Remote Direct Data Placement Work Group
INTERNET-DRAFT
draft-ietf-rddp-mpa-01.txt

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

Expires: January 2005

July 13, 2004

Marker PDU Aligned Framing for TCP Specification

Status of this Memo

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, or will be disclosed, and any of which I become aware will be disclosed, in accordance with RFC 3668.

By submitting this Internet-Draft, I accept the provisions of Section 4 of RFC 3667.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html. The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Abstract

A framing protocol is defined for TCP that is fully compliant with applicable TCP RFCs and fully interoperable with existing TCP implementations. The framing mechanism is designed to work as an "adaptation layer" between TCP and the Direct Data Placement [DDP] protocol, preserving the reliable, in-order delivery of TCP, while adding the preservation of higher-level protocol record boundaries that DDP requires.

Table of Contents

Status of this Memo....1
Abstract....1
1 Introduction....5
1.1 Motivation....5
1.2 Protocol Overview....5
2 Glossary....9
3 LLP and DDP requirements....11
3.1 TCP implementation Requirements to support MPA....11
3.1.1 TCP Transmit side....11
3.1.2 TCP Receive side....11
3.2 MPA's interactions with DDP....12
4 FPDU Formats....14
4.1 Marker Format....15
5 Data Transfer Semantics....16
5.1 MPA Markers....16
5.2 CRC Calculation....18
5.3 MPA on TCP Sender Segmentation....21
5.3.1 Effects of MPA on TCP Segmentation....21
5.3.2 FPDU Size Considerations....23
5.4 MPA Receiver FPDU Identification....24
5.4.1 Re-segmenting Middle boxes and non MPA-aware TCP senders....25
6 Connection Semantics....26
6.1 Connection setup....26
6.1.1 MPA Request Frame Format....30
6.1.2 Example Delayed Startup sequence....31
6.1.3 Use of "Private Data"....34
6.1.4 "Dual Stack" implementations....37
6.2 Normal Connection Teardown....38
7 Error Semantics....39
8 Security Considerations....40
8.1 Protocol-specific Security Considerations....40
8.2 Using IPsec With MPA....40
9 IANA Considerations....41
10 References....42
10.1 Normative References....42
10.2 Informative References....42
11 Appendix....44
11.1 Analysis of MPA over TCP Operations....44
11.1.1 Assumptions....44
11.1.2 The Value of Header Alignment....45
11.2 Receiver implementation....53
11.2.1 Network Layer Reassembly Buffers....53
11.2.2 TCP Reassembly buffers....54
12 Author's Addresses....55
13 Acknowledgments....56
14 Full Copyright Statement....59

Table of Figures

Figure 1 ULP MPA TCP Layering.....7
Figure 2 FPDU Format.....14
Figure 3 Marker Format.....15
Figure 4 Example FPDU Format with Marker.....17
Figure 5 Annotated Hex Dump of an FPDU.....20
Figure 6 Annotated Hex Dump of an FPDU with Marker.....20
Figure 7 "MPA Request/Reply Frame".....30
Figure 8: Example Delayed Startup negotiation.....32
Figure 9: Example Immediate Startup negotiation.....35
Figure 10: Non-aligned FPDU freely placed in TCP octet stream.....47
Figure 11: Aligned FPDU placed immediately after TCP header.....49

Revision history

[draft-ietf-rddp-mpa-02] workgroup draft with following changes:

Added the "R" bit (Rejected) to the "MPA Reply Frame" and described its semantics.

Added some comments on recent decisions regarding startup.

Updated RFC3667 boilerplate.

[draft-ietf-rddp-mpa-01] Alias of draft-ietf-rddp-map-00.

[draft-ietf-rddp-mpa-00] workgroup draft with following changes:

Changed "Start Key" to two separate startup frames to facilitate identification of incorrect Active/Active startup.

Changed Active/Passive nomenclature to Initiator/Responder to reduce confusion with TCP startup and verbs doc (which used opposite sense).

Added "Private Data" to the startup key sequences. This also required describing the motivation and expected usage models along with some interface hints. Removed the "Private data" stuff from appendix.

Added example "Immediate" startup with TCP and explanation.

[draft-culley-iwarp-mpa-03]

Add option to allow receivers to specify Marker use.

Add option that allows both sides to agree not to use CRC.

Added startup declaration "Start Key" with options and larger MPA mode recognition "key".

Updated MPA/DDP connection startup rules and sequence to deal with "Start Key".

Added Appendix that provides a more detailed analysis of the effects of MPA on TCP data streams.

Added appendix that describes a mechanism to deal with "private data" prior to full MPA/DDP operation.

[draft-culley-iwarp-mpa-02]

Enhanced descriptions of how MPA is used over an unmodified TCP.

Removed "No Packing" text.

Made MPA an adaptation layer for DDP, instead of a generalized framing solution.

Added clarifications of the MPA/TCP interaction for optimized implementations and that any such optimizations are to be used only when requested by MPA.

Note: a discussion of reasons for these changes can be found in [ELZER-MPA].

[draft-culley-iwarp-mpa-01] initial draft.

1 Introduction

This section discusses the reason for creating MPA on TCP and a general overview of the protocol. Later sections show the MPA headers (see section 4 on page 14), and detailed protocol requirements and characteristics (see section 5 on page 16), as well as Connection Semantics (section 6 on page 25), Error Semantics (section 7 on page 39), and Security Considerations (section 8 on page 40).

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. A probabilistic approach is characterized by a detectable value embedded in the octet stream. 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. A 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, the 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

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 6.1 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, when it is available, or chooses a reasonable value. This information is already supported on many TCP implementations, including all modern flavors of BSD networking, through the TCP_MAXSEG socket option.

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 pre-pending 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 TCP Sender is MPA-aware, 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 TCP receiver may be MPA-aware or may not be MPA-aware. If it is MPA-aware, 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.

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

MPA-aware TCP is a TCP layer which potentially contains some additional semantics as defined in this document. MPA is implemented as a data stream ULP for TCP and is therefore RFC compliant. MPA-aware TCP is RFC compliant.

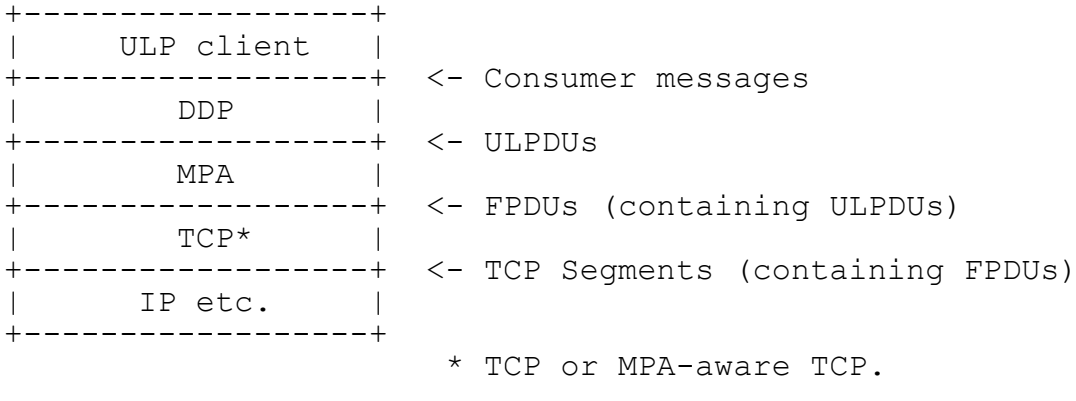

Figure 1 ULP MPA TCP Layering

An MPA-aware 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).

MPA with an MPA-aware TCP receiver allows a DDP on MPA implementation to recover ULPDUs that may be received out of order. 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.

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. Many users of TCP have noted that the TCP checksum is not as strong as could be desired [CRCTCP]. Studies 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 [RFC2960]. This check may be disabled with agreement by providers and administrators at both ends of a connection. This disabling of CRCs should only be done when it is clear that the connection through the network has data integrity at least as good as a CRC (for example when IPSEC is implemented end to end). DDP's ULP expects this level of data integrity and therefore the ULP SHOULD NOT have to provide its own duplicate data integrity and error recovery for lost data.

2 Glossary

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.

Delivery - (Delivered, Delivers) - For MPA, Delivery is defined as the process of informing DDP that a particular PDU is ordered for use. 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 Transfer Unit (MTU) [RFC1191].

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

FPDU Alignment - the property that a TCP segment begins with an FPDU.

Header Alignment - the property that a TCP segment begins with an FPDU and the TCP segment includes an integer number of FPDUs.

PDU - protocol data unit

MPA-aware TCP - a TCP implementation that is aware of the receiver efficiencies of MPA Header 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 - 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.

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.

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".

3 LLP and DDP requirements

3.1 TCP implementation Requirements to support MPA

The TCP implementation MUST inform MPA when the TCP connection is closed or has begun closing the connection (e.g. received a FIN).

3.1.1 TCP Transmit side

To provide optimum performance, an MPA-aware transmit side TCP 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 to the MPA transmit layer.

An MPA-aware TCP transmit side implementation MUST continue to use the method of segmentation expected by non-MPA applications (and described in TCP RFCs) when MPA is not enabled on the connection. When MPA is enabled above an MPA-aware TCP, it SHOULD specifically enable the segmentation rules described above for the DDP segments (FPDUs) posted for transmission.

If the transmit side TCP implementation is not able to segment the TCP stream as indicated above, MPA SHOULD make a best effort to achieve that result. For example, using the TCP_NODELAY socket option to disable the Nagle algorithm will usually result in many of the segments starting with an FPDU.

If the transmit side TCP implementation is not able to report the EMSS, MPA may assume that TCP will use 1460 octet segments in creating FPDUs. If the implementation has reason to believe that the TCP segment size is actually smaller than 1460, it may instead use a 536 octet FPDU.

3.1.2 TCP Receive side

When an MPA receive implementation and the MPA-aware receive side TCP implementation support handling out of order ULPDUs, the TCP receive implementation SHOULD be enabled to:

* Pass incoming TCP segments to MPA as soon as they have been received and validated, even if not received in order. The TCP layer MUST have committed 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 not be a

duplicate, must be part of the same epoch (if timestamps are used to verify this) and any other checks required by TCP RFCs. The segment MUST NOT be passed to MPA more than once unless explicitly requested (see Section 7).

This is not to imply that the data must be completely ordered before use. An implementation may accept out of order segments, SACK them [RFC2018], and pass them to DDP when the reception of the segments needed to fill in the gaps arrive. Such an implementation can "commit" to the data early on, and will not overwrite it even if (or when) duplicate data arrives. MPA expects to utilize this "commit" to allow the passing of ULPDUs to DDP when they arrive, independent of ordering.

* Provide 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.

* Provide 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.

  DDP on MPA MUST utilize these 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.

An MPA-aware TCP receive side implementation MUST continue to buffer TCP segments until completely ordered and then deliver them as expected by non-MPA applications (and described in TCP RFCs) when MPA is not enabled on the connection. When MPA is enabled above an MPA-aware TCP, TCP SHOULD enable the in and out of order passing of data, and the separate ordering information as described above.

When an MPA receive implementation is coupled with a TCP receive implementation that does not support the preceding mechanisms, TCP passes and Delivers incoming stream data to MPA in order.

3.2 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 combined with an MPA-aware TCP can only ensure FPDU Alignment with the TCP Header if the FPDU is less than or equal to TCP's EMSS. Since FPDU alignment is generally desired by the receiver, DDP must cooperate with MPA to ensure FPDUs' lengths do not exceed the EMSS under normal conditions. This is done with the MULPDU mechanism.

MPA provides 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.

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. 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).

4 FPDU Formats

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.

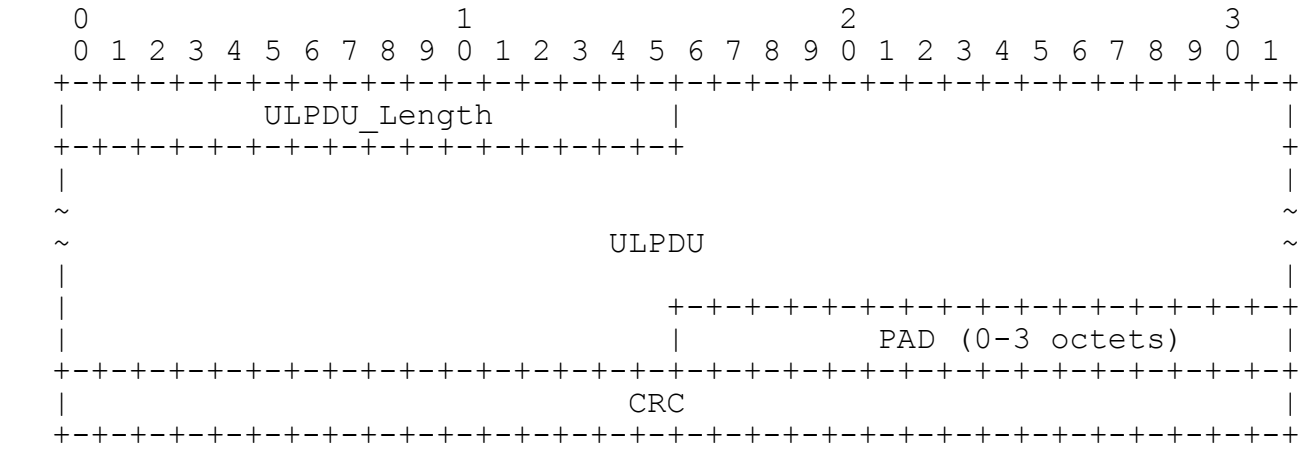

Figure 2 FPDU Format

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 zero and three 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 5.2 CRC Calculation on page 18. 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.

4.1 Marker Format

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

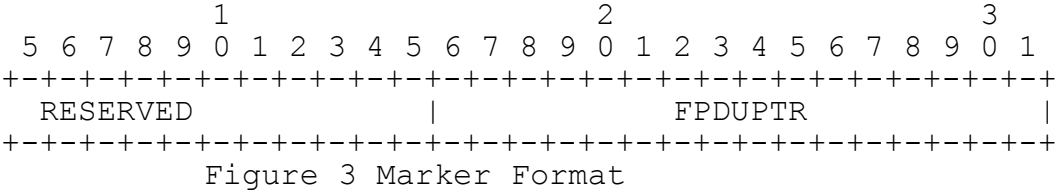

Figure 3 Marker Format

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 FPDU to the first octet of the entire marker.

5 Data Transfer Semantics

This section discusses some characteristics and behavior of the MPA protocol as well as implications of that protocol.

5.1 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.

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 6.1 Connection setup on page 26).

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 current FPDU 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. In this case, the marker MUST be placed in the following FPDU and viewed as being part of that FPDU (e.g. for CRC calculation). Thus an FPDUPTR value of 0x0000 means that immediately following the marker is an FPDU header.

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 6.1 Connection setup on page 26), the MPA markers MUST be inserted immediately following MPA

connection establishment, 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 FPDU.

If a marker interval contains multiple FPDUs (the FPDUs are small), the marker MUST point to the start of 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.

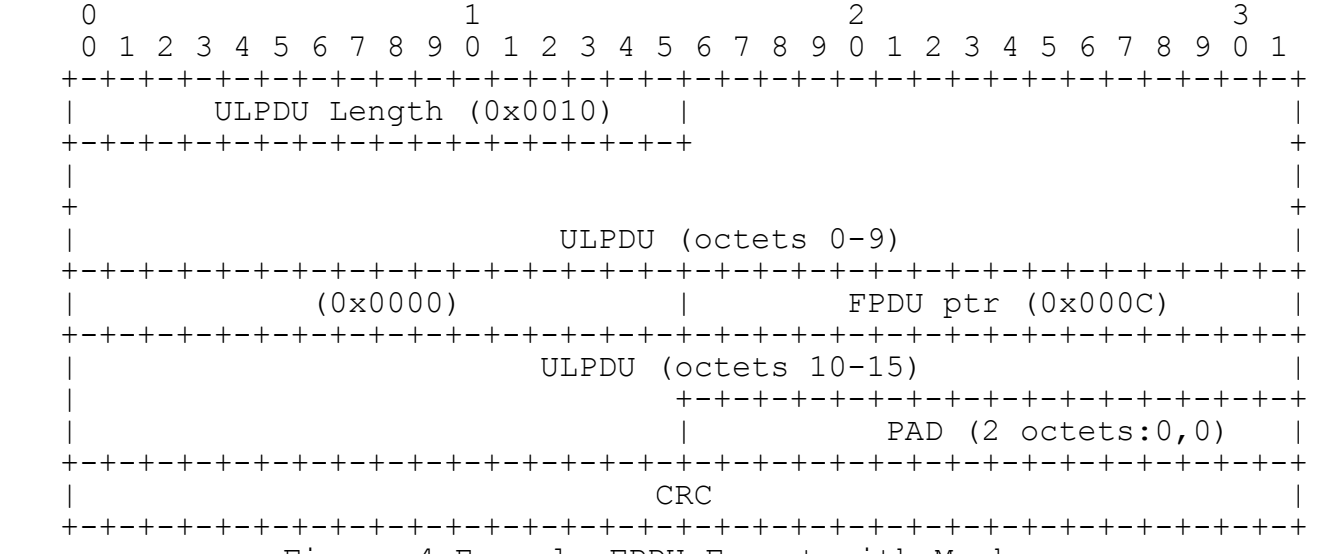

Figure 4 Example FPDU Format with Marker

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.

5.2 CRC Calculation

An MPA implementation MUST implement CRC support and MUST either:

(1) always use CRCs

or

(2) only negotiate the non-use of CRC 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.

(3) The MPA provider at either peer MAY ignore its administrator's request that CRCs not be used.

The decision for one host to request CRC suppression MAY be made on an administrative basis for any path that provides equivalent protection from undetected errors as an end-to-end CRC32c.

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 6.1 Connection setup on page 26 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 the first octet of the FPDU is the "ULPDU Length" field, the CRC-32c is calculated from the first octet of the "ULPDU Length" header, 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 the first octet of the FPDU is a marker, (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 the Error Section (see section 7 on page 39)

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 24 octets of the contained ULPDU, which are all zeros. The CRC32c has been correctly calculated and can be used as a reference. See the [DDP] and [RDMA] specification for definitions of the DDP Control field, Queue, MSN, MO, and Send Data.

```
    Octet Contents  Annotation
    Count

    0000    00 00   Marker: Reserved
    0002    00 00           FPDUPTR
    0004    00 2a   Length
    0006    40 03   DDP Control Field, Send with Last flag set
    0008    00 00   Reserved (STag position with no STag)
    000a    00 00
    000c    00 00   Queue = 0
    000e    00 00
    0010    00 00   MSN = 1
    0012    00 01
    0014    00 00   MO = 0
    0016    00 00
    0018    00 00
                    Send Data (24 octets of zeros)
    002e    00 00
    0030    4C 86   CRC32c
    0032    B3 84
```

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 2a   Length
    01ee    40 03   DDP Control Field: Send with Last Flag set
    01f0    00 00   Reserved (STag position with no STag)
    01f2    00 00
    01f4    00 00   Queue = 0
    01f6    00 00
    01f8    00 00   MSN = 2
    01fa    00 02
    01fc    00 00   MO = 0
    01fe    00 00
    0200    00 00   Marker: Reserved
    0202    00 14           FPDUPTR
    0204    00 00
                    Send Data (24 octets of zeros)
    021a    00 00
    021c    A1 9C   CRC32c
    021e    D1 03
```

Figure 6 Annotated Hex Dump of an FPDU with Marker

5.3 MPA on TCP Sender Segmentation

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

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

An MPA-aware TCP sender SHOULD, when enabled for MPA, on TCP implementations that support this, and with an EMSS large enough to contain at least one FPDU, segment the outbound TCP stream such that each TCP segment begins with an FPDU, and fully contains all included FPDUs.

> Implementation note: To achieve the previous segmentation rule, TCP's Nagle [RFC0896] algorithm SHOULD be disabled.

There are exceptions to the above rule. Once an ULPDU is provided to MPA, the MPA on TCP sender MUST transmit it or fail 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.

The sender MUST still format 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 MPA-aware TCP sender SHOULD try to preserve original TCP segmentation boundaries on a retransmission.

5.3.1 Effects of MPA on TCP Segmentation

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. Therefore, the application is expected to set TCP parameters such that it can trade off latency and wire efficiency. This is accomplished by setting the TCP_NODELAY socket option.

When latency is not critical, and the application provides data in chunks larger than EMSS at one time, the TCP implementation may "pack" any available stream data 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 (discussed below).

DDP/MPA 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. If a "message tail", small DDP messages, or the start of a larger DDP message are available, MPA MAY "pack" the resulting 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.

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

TCP implementations often utilize the "Nagle" [RFC0896] 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].

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.

5.3.2 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 7.3.1. 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, the MULPDU value is changed accordingly.

In certain rare situations, the EMSS may shrink to very small sizes. 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 5.3 MPA on TCP Sender Segmentation on page 21.

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.

5.4 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 receivers that support out of order reception of FPDUs (see section 5.1 MPA Markers on page 16) 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, following contiguous FPDUs can be found by using the ULPDU Lengths (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 5.2 on page 18 above.

5.4.1 Re-segmenting Middle boxes and non MPA-aware TCP senders

Since MPA on MPA-aware TCP senders start FPDUs on TCP segment boundaries, a receiving DDP on MPA on 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 if the sender is MPA-aware, the network may contain "middle boxes" 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 MPA 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.

6 Connection Semantics

6.1 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 9: Example Immediate Startup negotiation on page 35.

  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 8: Example Delayed Startup negotiation on page 32.

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 draft). 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 actinos, 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 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 the side which starts in "Initiator" and which starts in "Responder" MPA mode.

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 a "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 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 a "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 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 mode; 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 6.1.1 MPA Request and Reply Frame Format below) 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 a "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 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.

6.1.1 MPA Request and Reply Frame Format

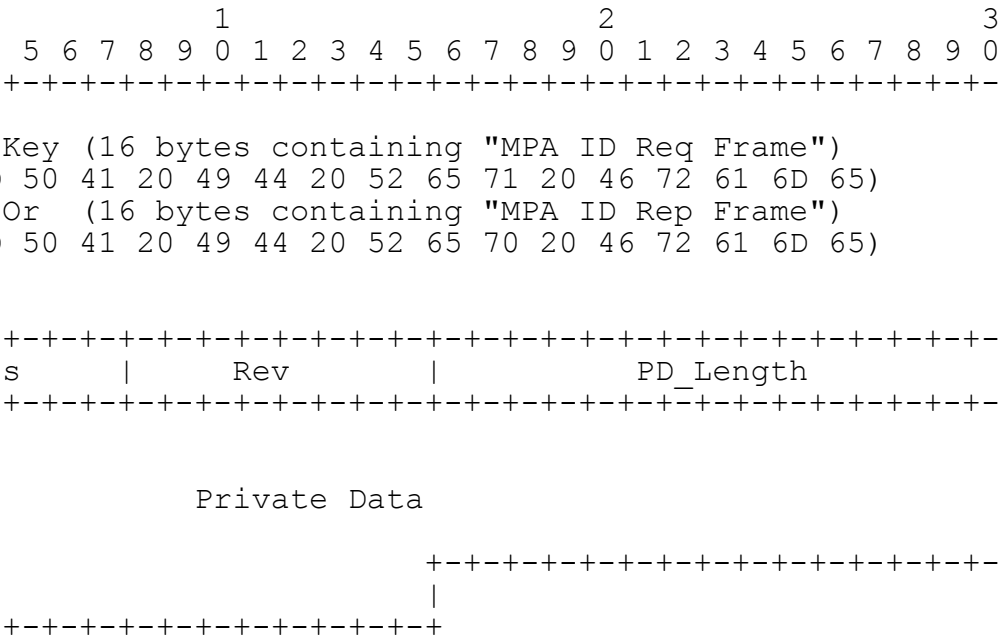

Figure 7 "MPA Request/Reply Frame"

Key: This field contains the "key" used to authenticate 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, when sent in an "MPA Request Frame" or an "MPA Reply Frame", declares a receiver's requirement for Markers. 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 the sender. When '1' markers MUST be added as described in section 5.1 MPA Markers on page 16.

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.

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 zero. MPA receivers compliant with this version of the specification MUST check this field for zero, and close the connection and report an error locally if any other value is detected.

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. The private data field may be as long as 65535 Octets.

Private Data: This field may contain any value defined by ULPs or may not be present. ULPs define how to set and validate this field.

6.1.2 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 6.1.3.2 Example Immediate Startup using Private Data on page 35), 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".

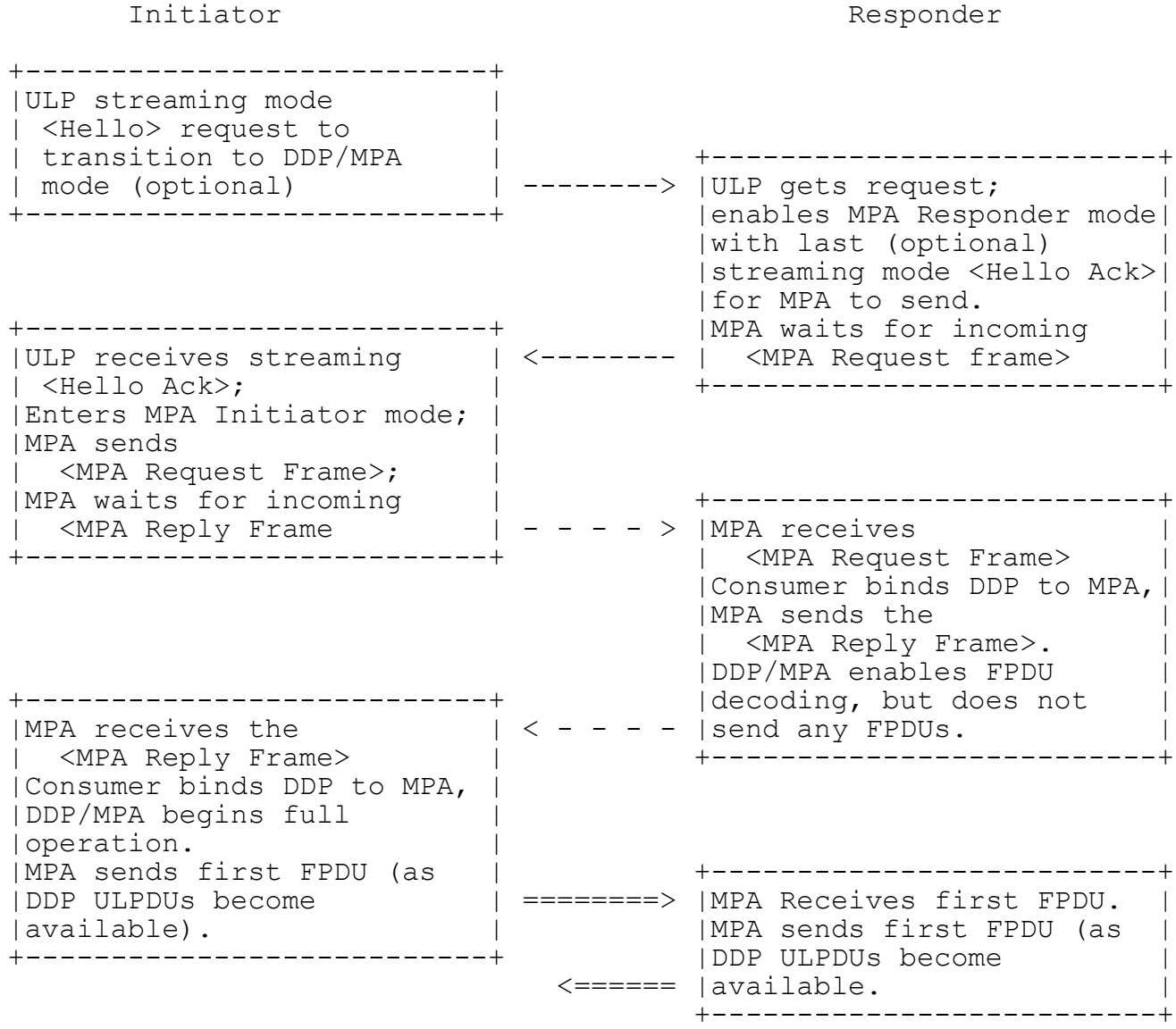

Figure 8: Example Delayed Startup negotiation

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, Lets 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.

6.1.3 Use of "Private Data"

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

6.1.3.1 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. For example,

An RDMA Endpoint (referred to as a Queue Pair, or QP, in InfiniBand and the [Verbs]) 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 iWARP verbs [Verbs] 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.

InfiniBand, the DAT APIs and the 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. 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 draft 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.

6.1.3.2 Example Immediate Startup using Private Data

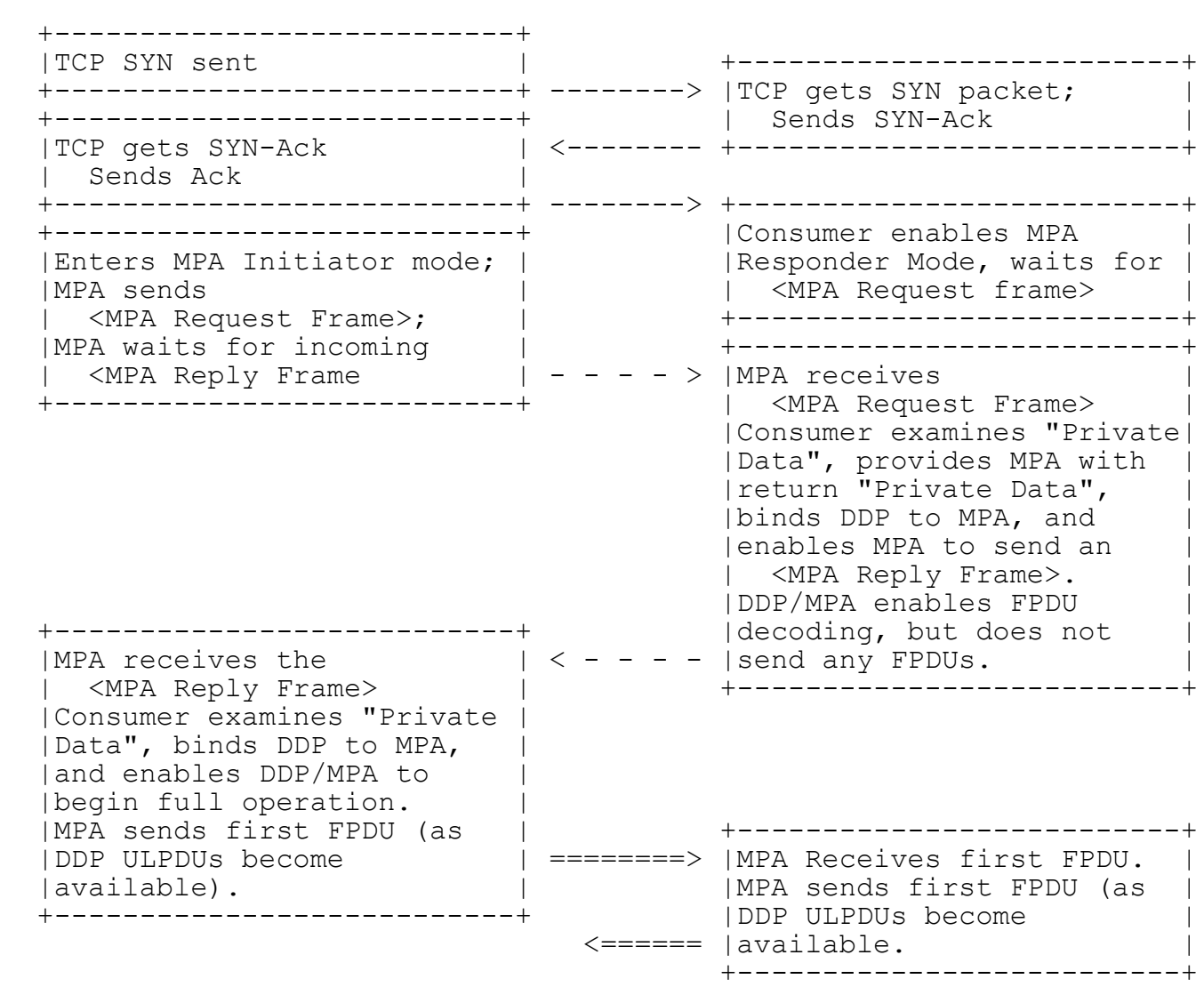

Figure 9: Example Immediate Startup negotiation

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.

6.1.4 "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. 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.

6.2 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 LLP connection has been received by the LLP (e.g. FIN is received).

7 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 a 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 MPA-aware 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.

8 Security Considerations

This section discusses the security considerations for MPA.

8.1 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. This would have a severe impact on performance. Communication security mechanisms such as IPsec [RFC2401] 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.

8.2 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.

9 IANA Considerations

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.

10 References

10.1 Normative References

[iSCSI] Satran, J., "iSCSI", draft-ietf-ips-iscsi-20.txt (work in progress), January 2003.

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

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

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[RFC793] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, September 1981.

10.2 Informative References

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

[DDP] H. Shah et al., "Direct Data Placement over Reliable Transports", draft-ietf-rddp-ddp-02.txt (Work in progress), February 2004

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

[RFC0896] J. Nagle, "Congestion Control in IP/TCP Internetworks", RFC 896, January 1984.

[NagleDAck] Minshall G., Mogul, J., Saito, Y., Verghese, B., "Application performance pitfalls and TCP's Nagle algorithm", Workshop on Internet Server Performance, May 1999.

[RDMA] R. Recio et al., "RDMA Protocol Specification", draft-ietf-rddp-rdmap-02.txt, May 2004

[RFC2960] R. Stewart et al., "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC792] Postel, J., "Internet Control Message Protocol". September 1981

[RFC1122] Braden, R.T., "Requirements for Internet hosts - communication layers". October 1989.

[ELZUR-MPA] Elzur, U., "Analysis of MPA over TCP Operations" draft-elzur-iwarp-mpa-tcp-analysis-00.txt, February 2003.

[Verbs] J. Hilland et al., "RDMA Protocol Verbs Specification" draft-hilland-rddp-verbs-00.txt, April 2003.

11 Appendix

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

11.1 Analysis of MPA over TCP Operations

This appendix analyzes the impact of MPA (Marker PDU Aligned Framing for TCP [MPA]) on the TCP sender, receiver, and wire protocol.

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 require the TCP sender to do "Header Alignment", where a TCP segment is required to 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 Header 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.

11.1.1 Assumptions

11.1.1.1 MPA is layered beneath DDP [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.

11.1.1.2 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.

11.1.1.3 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

11.1.1.4 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.

11.1.2 The Value of Header 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/S] * Delay [S].

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 Gbps link with 10,000 connections and an EMSS of 1500B would require 15 MB of memory. Often the number of connections used scales with the network speed, aggravating the situation for higher speeds.

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

BufferSizeAF = K2 * EMSS

for the same conditions. A Header 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 NIC and is placed directly in the destination buffer. Assuming most of the connections support Header 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 Header Aligned FPDU 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 Header Aligned FPDU approach is used, receive buffering is expected to scale gracefully (i.e. less than linear scaling) as network speed is increased.

11.1.2.1 Impact of lack of Header 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 a FPDU is carried in no more than 2 TCP segments. In reality, with no Header Alignment, an FPDU can be carried by more than 2 TCP segments (e.g., if the PMTU was reduced).

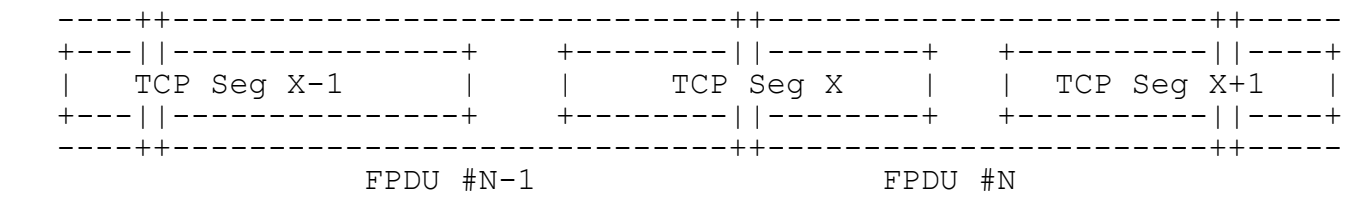

Figure 10: Non-aligned FPDU freely placed in TCP octet stream

The receiver algorithm for processing TCP segments (e.g., TCP segment #X in Figure 10: Non-aligned FPDU freely placed in TCP octet stream) 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 TCP out-of-order, 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 Header 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 Header Alignment for every TCP segment received.

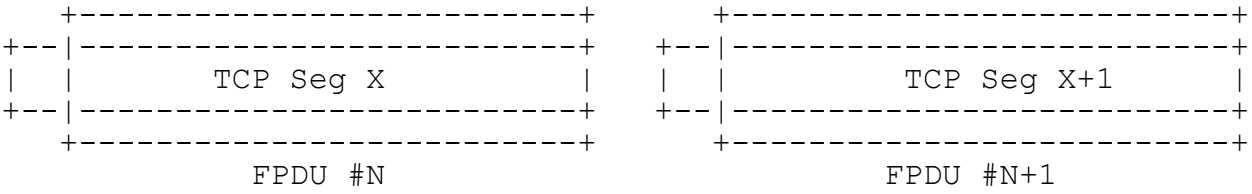

Figure 11: Aligned FPDU placed immediately after TCP header

The receiver algorithm for Header 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 [MPA] section 7.4). 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 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 Header Alignment, the receiver is forced to classify incoming traffic before it can calculate the FPDU CRC. In the case of Header Alignment the operations order is left to the implementer.

The Header 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 a 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 Header Alignment is not found, the receiver should follow the algorithm for non aligned FPDU reception which may be slower and less efficient.

11.1.2.2 Header Alignment effects on TCP wire protocol

> An MPA-aware 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 few bytes to multiple EMSS (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 a 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 Header 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 [RFC0793] section 2.6, [RFC0793] section 3.7 "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 middle box, as documented in 5.4.1 Re-segmenting Middle boxes and non MPA-aware TCP senders on page 25), a better available bandwidth utilization may be possible by re-transmitting only the missing octets. If an MPA-aware 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 MPA-aware 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, MPA-aware 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 MPA-aware TCP will behave similarly to any TCP sender and allow an MPA-aware 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.

11.2 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 middle box is present, or where the TCP sender is not MPA-aware, the presence of markers significantly reduces the amount of buffering needed.

Recovery from IP Fragmentation must be transparent to the MPA Consumers.

11.2.1 Network Layer Reassembly Buffers

Most IP implementations set the IP Don't Fragment bit. Thus upon a path MTU change, intermediate devices drop the IP datagram if it is too large and reply with an ICMP message which 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 RFC792.

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 which 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].

11.2.2 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, the buffer MUST be at least 140 octets long to support the minimum FPDU size. The 140 octets allows 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 may provide better performance.

Note that if the TCP segment were not stored, it is 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.

12 Author's Addresses

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

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

Uri Elzur
Broadcom
16215 Alton Parkway
CA, 92618
Phone: 949.585.6432
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

John Carrier
Adaptec Inc.
691 South Milpitas Blvd.
Milpitas, CA 95035
Phone: 360-378-8526
Email: John Carrier@adaptec.com

13 Acknowledgments

Dwight Barron
Hewlett-Packard Company
20555 SH 249
Houston, Tx. USA 77070-2698
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
Hewlett-Packard Company
19333 Vallco Parkway
Cupertino, Ca. USA 95014
Phone: 408.285.6116
Email: dave.garcia@hp.com

Hari Ghadia
Adaptec, Inc.
691 S. Milpitas Blvd.,
Milpitas, CA 95035 USA
Phone: +1 (408) 957-5608
Email: hari ghadia@adaptec.com

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

Jeff Hilland
Hewlett-Packard Company
20555 SH 249
Houston, Tx. USA 77070-2698
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, USA 98052
Email: jpink@microsoft.com

Hemal Shah
Intel Corporation
MS PTL1
1501 South Mopac Expressway, #400
Austin, Texas 78746
Phone: 512-732-3963
Email: hemal.shah@intel.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
375 Totten Pond Road
Waltham, MA 02451 USA
Phone: +1 (781) 768-5329
EMail: thomas.talpey@netapp.com

Patricia Thaler
Agilent Technologies, Inc.
1101 Creekside Ridge Drive, #100
M/S-RG10
Roseville, CA 95678
Phone: +1-916-788-5662
email: pat thaler@agilent.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

14 Full Copyright Statement

This document and the information contained herein is provided on an "AS IS" basis and ADAPTEC INC., AGILENT TECHNOLOGIES INC., BROADCOM CORPORATION, CISCO SYSTEMS INC., DUKE UNIVERSITY, EMC CORPORATION, EMULEX CORPORATION, HEWLETT-PACKARD COMPANY, INTERNATIONAL BUSINESS MACHINES CORPORATION, INTEL CORPORATION, MICROSOFT CORPORATION, NETWORK APPLIANCE INC., SANDBURST CORPORATION, THE INTERNET SOCIETY, AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright (C) The Internet Society (2004). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.