Internet Draft
draft-pinkerton-iwarp-sdp-00

James Pinkerton
  Microsoft Corporation
Ellen Deleganes
  Intel Corporation
Michael Krause
  Hewlett-Packard Company

Expires: May 2004

Sockets Direct Protocol (SDP) for iWARP over TCP

1 Status of this Memo

2 Abstract

SDP is a byte-stream transport protocol that closely mimics TCP's stream semantics. SDP utilizes iWARP's advanced protocol offload, kernel bypass, and zero copy capabilities. Because of this, SDP can have lower CPU and memory bandwidth utilization when compared to conventional implementations of sockets over TCP, while preserving the familiar byte-stream oriented semantics upon which most current network applications depend.

3 Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

Table of Contents

Table of Figures

4 Introduction

This document defines a transport layer protocol called Sockets Direct Protocol (SDP) for iWARP over TCP.

SDP is a byte-stream transport protocol that closely mimics TCP's stream semantics. SDP utilizes iWARP's advanced protocol offload, kernel bypass, and zero copy capabilities. SDP allows existing sockets applications to gain the performance benefits of RDMA for data transfers without requiring any modifications to the application. Because of this, SDP can have lower CPU and memory bandwidth utilization when compared to conventional implementations of sockets over TCP, while preserving the familiar byte-stream oriented semantics upon which most current network applications depend.

The SDP implementation described in this specification is intended to emulate sockets semantics over TCP, and to be layered on iWARP mapped over TCP. Support for emulating SCTP sockets semantics may be done in a future specification. This layering is shown in Figure 1.

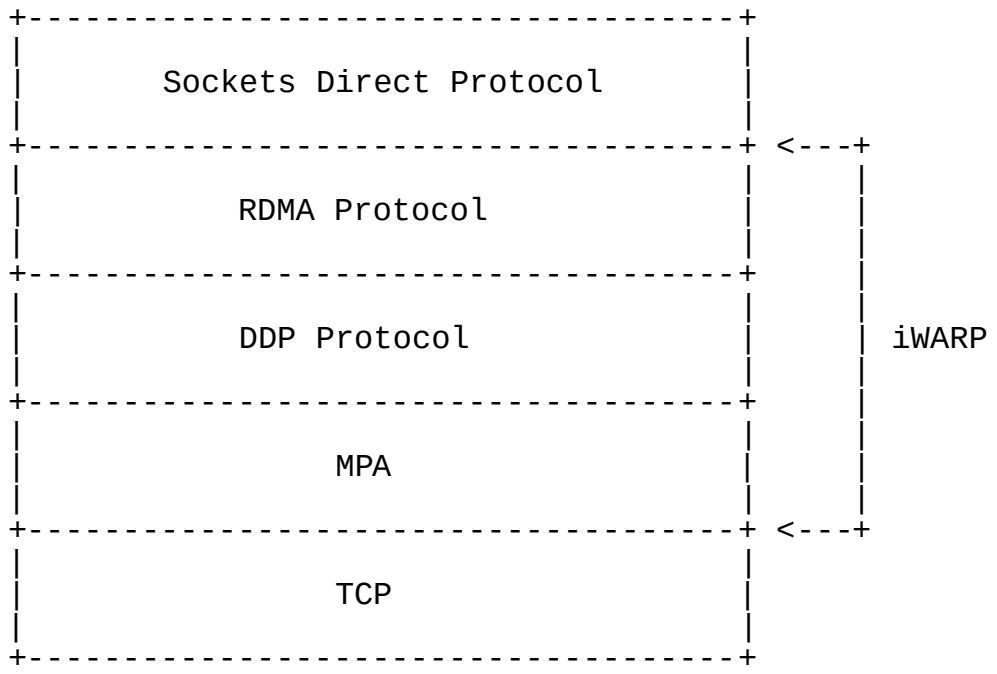

Figure 1 SDP Relation to iWARP Layers

4.1 Architectural Goals

SDP has the following architectural goals:

- Maintain traditional sockets SOCK_STREAM semantics as commonly implemented over TCP/IP. Some specific issues that are addressed include:
  - Graceful close, including half-closed sockets
  - Ability to use TCP port space
  - IP addressing (IPv4 or IPv6)
  - Connecting/accepting connect model
  - Out-of-band (OOB) data
  - Support for common socket options
- Support for byte-streaming over a message passing protocol
  - Capable of supporting kernel bypass data transfers
  - Capable of supporting Zero-copy data transfers from send upper-layer-protocol (ULP) buffers to receive ULP Buffers.

This specification focuses specifically on the wire protocol, finite state machine, and packet semantics. Operating system specific issues and other implementation-specific issues are outside the scope of this specification, including application programming interfaces (APIs), ULP completion mechanisms, kernel bypass capabilities, etc. These issues are left up to each implementation.

Note that SDP only supports SOCK_STREAM semantics (i.e., byte-stream), not SOCK_DGRAM (i.e., datagram) semantics. In addition, the socket emulation description assumes that the desired behavior is to emulate SOCK_STREAM with TCP semantics.

4.2 Overview of the Byte-Stream Protocol

SDP’s ULP interface is a byte-stream interface that is layered on top of iWARP’s message-oriented transfer model. The mapping of the byte-stream protocol to iWARP Message-oriented semantics is designed to enable ULP data to be transferred by one of two methods - through intermediate Private Buffers (Buffer-copy, also referred to as Bcopy) or directly between ULP Buffers (Zero-copy, also referred to as Zcopy).

A mix of iWARP Send and RDMA mechanisms are used to transfer ULP data. Zcopy uses iWARP RDMA Reads or Writes, transferring data between RDMA Buffers. Bcopy uses iWARP Sends, transferring data between send buffers and receive Private Buffers. An implementation is expected, but not required, to have RDMA Buffers be ULP Buffers, enabling the RDMA path to perform a true Zero-copy. An implementation is expected, but not required, to use the receive Private Buffer pool to buffer data and eventually copy the data from the receive Private Buffer pool into the receive ULP Buffer. An implementation may choose to implement different ULP Buffering semantics.

SDP has two types of buffers:

* Private Buffers - used for transmission of all SDP Messages and ULP data that is to be copied into the receive ULP Buffer. The Bcopy Data Transfer Mechanism is used to perform the copy of SDP Messages and ULP data into these buffers.

* RDMA Buffers - used when performing Zcopy data transfers. ULP data is intended to be transferred (using RDMA Writes or RDMA Reads) directly from the Data Source's ULP Buffer to the Data Sink's ULP Buffer.

The policy that controls when to use Private Buffers versus RDMA Buffers is outside the scope of this specification. An implementation-dependent parameter defined as the Bcopy Threshold is used to abstractly define the results of the policy decision.

As noted in the introduction, SDP is a wire protocol for iWARP over TCP. Thus, when this specification uses the term "send" in relation to an SDP message, it refers to when the iWARP message has been sent. The end-to-end acknowledgement may or may not have occurred. This implies that when an SDP message is sent, there is no guarantee that the Remote Peer has actually received the message. When this specification uses the term "receive" in relation to an SDP message, it refers to when the SDP protocol layer has been handed the message by the LLP.

5 Definitions

Accepting Peer - The peer that sent the reply to the connection establishment request during connection establishment.

Bcopy - See Buffer-copy.

Buffered Mode - One of three modes that can be used for an SDP half-connection. This Mode uses the Bcopy Data Transfer Mechanism exclusively. When used in the context of the Port Mapper Protocol, the Accepting Peer can also be a Port Mapper Service Provider (PMSP) acting on behalf of the Accepting Peer.

Bcopy Threshold - A locally defined threshold existing separately for each peer of a half-connection, which helps the peer determine whether it will attempt to use the Bcopy Data Transfer Mechanism versus the Zcopy Data Transfer Mechanism to transfer a given size ULP Buffer.

Buffer-copy - A Data Transfer Mechanism where the transfer of ULP payload between peers is done through an SDP-managed receive Private Buffer pool. The received ULP data may require a copy into the receive ULP Buffer.

Combined Mode - One of three modes that can be used for an SDP half-connection. This Mode enables the use of the Bcopy and Read Zcopy Data Transfer Mechanisms.

Connecting Peer - The peer that sent the connection establishment request. When used in the context of the Port Mapper Protocol, a Connecting Peer can also be a management agent acting on behalf of the Connecting Peer.

Connection Context - The endpoint state needed for the LLP and the iWARP protocol suite. This might include protocol control state, mappings from STags to buffers, queues for transmission and reception of data, etc.

Controlling Address Space - The address space in which the socket currently exists. This is relevant for socket duplication where a Non-Controlling Address Space may request control of the socket.

Data Sink - The peer receiving ULP payload. The Data Sink can be required to both send and receive iWARP and/or SDP Messages to complete a Data Transfer Mechanism.

Data Source - The peer sending ULP payload. The Data Source can be required to both send and receive iWARP and/or SDP Messages to complete a Data Transfer Mechanism.

Data Transfer Mechanism - A sequence of iWARP and/or SDP Messages (with or without ULP payload) used to transfer data from a Data Source to a Data Sink with flow control. Four Data Transfer Mechanisms are defined - Buffer-copy (Bcopy), Zero-copy with RDMA Write (Write Zcopy), Zero-copy with RDMA Read (Read Zcopy), and Transaction.

Flow Control Mode - The Mode of the half-connection that determines which Data Transfer Mechanisms may be used. Three Flow Control Modes are defined - Buffered, Pipelined, and Combined.

In-Process - An SDP Message sequence is In-Process if it is actively being worked on. For example, if a SrcAvail Message is In-Process, RDMA Reads may have been issued or completed, but a RdmaRdCompl Message or SendSm Message has not been sent. See also Unprocessed and Processed.

Inbound RDMA Read Queue Depth (IRD) - The Inbound RDMA Read Queue Depth is the number of simultaneous outstanding inbound RDMA Read Requests.

Incomplete - See In-Process.

LLP - see Lower Layer Protocol.

Lower Layer Protocol - The protocol layer(s) beneath the protocol layer currently being referenced. For example, the LLP underneath SDP is the iWARP protocol family after the Connection Management sequence (see [RDMAP]) has been completed. Before the Connection Management sequence has been completed, the LLP is expected to be TCP.

LLP Connection - Corresponds to an LLP transport-level connection between peer LLP layers on two nodes. For SDP, the LLP Connection is initially established in Streaming Mode, and then iWARP communication is enabled.

LLP Stream - Corresponds to a single LLP transport-level Stream between peer LLP layers. One or more LLP Streams may map to a single transport-level LLP connection. For transport protocols that support multiple Streams per connection (e.g. SCTP), a LLP Stream corresponds to one transport-level Stream.

Message - See SDP Message.

Mode - See Flow Control Mode.

Mode Master - The side of an SDP half-connection that can initiate a Mode transition by sending a ModeChange Message. This is either

the Data Source or the Data Sink, depending upon the Flow Control Mode.

Mode Slave - The side of an SDP half-connection that reacts to Mode changes. The Mode Slave can advise the Mode Master to change modes (by using SendSm Messages or setting the REQ_PIPE flag in the BSDH), but it cannot force a Mode change. This is either the Data Source or the Data Sink, depending upon the Flow Control Mode.

Non-Controlling Address Space - An address space that does not currently have control of the socket. See Controlling Address Space.

OOB - See Out-Of-Band Data.

Out-Of-Band Data - Out-of-Band Data is a single byte of data in the data stream whose handling should be expedited.

Outbound RDMA Read Queue Depth (ORD) - The number of simultaneous outstanding RDMA Read Requests that can be issued on any given connection.

Pipelined Mode - One of three modes that can be used for an SDP half-connection. This Mode enables the use of the Bcopy, Read Zcopy, Write Zcopy, and Transaction Data Transfer Mechanisms.

Private Buffer - Buffers owned by SDP and not exposed to the ULP. Receive Private Buffers are used for reception of all SDP Messages and ULP data transfer using the Bcopy or Transaction Data Transfer Mechanisms. All receive Private Buffers must be at least the current advertised size, and are posted to the iWARP Receive Queue.

Processed - An SDP Message sequence has been completed by sending the last SDP Message of the sequence. Note that the completing SDP Message may not have been received. For example, a SinkAvail advertisement is said to have been Processed when the corresponding RdmaWrCompl Message has been sent. See also Unprocessed and In-Process.

RDMA Buffer - A buffer that is exposed by the SDP protocol for RDMA access. It is used by the Write Zcopy, Read Zcopy, and Transaction Data Transfer Mechanisms.

Receiver - Destination of an SDP Message.

SDP Message - An iWARP Send Type Message that contains an SDP Base Sockets Direct Header (BSDH). This specifically does not include iWARP RDMA Write and RDMA Read Messages.

Sender - Source of an SDP Message.

Streaming Mode - The term used for the transport used to transmit messages that are exchanged before the iWARP protocol has been established on the LLP Stream. The SDP Hello Message is transferred using Streaming Mode.

Transaction - A Data Transfer Mechanism that collapses a Data Message and a SinkAvail Message into a single SDP Message.

ULP - Upper Layer Protocol.

ULP Buffer - Buffers owned by and visible to the ULP. A ULP Buffer may serve as an RDMA source buffer, an RDMA sink buffer, or a send buffer.

Unprocessed - An SDP Message is Unprocessed if it has been sent, and possibly received, but it has not been operated on. For example, a SinkAvail Message is Unprocessed if it has been sent by the Data Sink, received by the Data Source, but no RDMA Writes have begun. In addition, processing of flow control information by the receiver may have been done. See also In-Process and Processed.

WrapSubtract - WrapSubtract represents a function that subtracts the second argument (arg2) from the first argument (arg1). Both arguments are unsigned integers that wrap from a value of 0xFFFFFFFF to 0x0. Mathematical operations on wrapping unsigned integers can be done using a variety of methods, including methods defined in RFC1982. The following equation is an example implementation of the function that casts the unsigned integers into two’s complement integers, and then takes the absolute value of the result:

```
result = abs((int)arg1 - (int)arg2)
```

Zcopy - See Zero-copy.

Zero-copy - Three Data Transfer Mechanisms (Read Zcopy, Write Zcopy, and Transactions), where the transfer of ULP payload between peers is done directly into the ULP Buffer using an RDMA Read or RDMA Write, thus avoiding a Buffer-copy on receive.

6 SDP Message Formats

Sockets Direct Protocol defines several types of SDP Messages to transfer data and control the state of a connection. Each SDP Message MUST contain a Base Sockets Direct Header (BSDH). Some SDP Message types may also contain an extended header and/or ULP payload. The extended header (if present) MUST immediately follow the BSDH. The ULP payload (if present) MUST immediately follow the headers (BSDH and extended header, if any).

All SDP Message headers MUST use network byte order (i.e. big-endian byte ordering).

6.1 Base Sockets Direct Header (BSDH)

All SDP Messages contain the Base Sockets Direct Header, as specified in this chapter.

The BSDH format MUST be as defined in Figure 2 Base Sockets Direct Header.

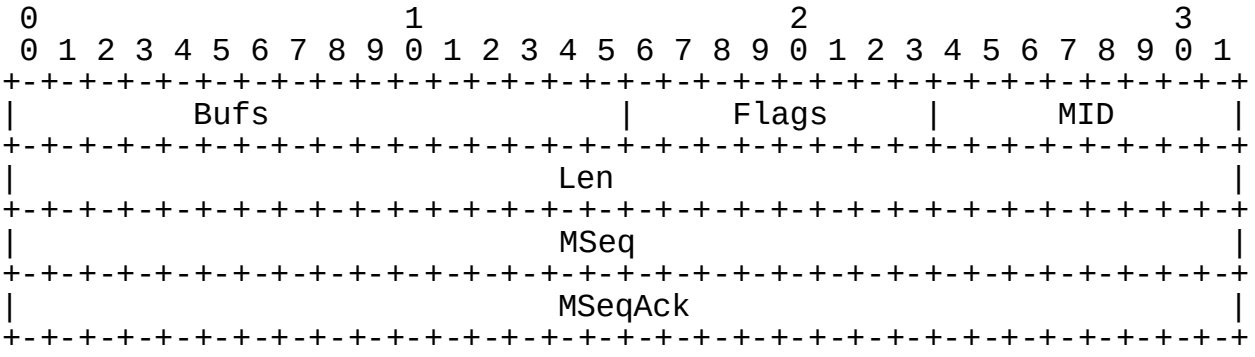

Figure 2 Base Sockets Direct Header

6.1.1 Message Identifier (MID)

The MID specifies the type of the SDP Message. The type of SDP Message indicates whether an extension header is present.

Figure 3 MUST be used to define the MID parameter in the BSDH, the type of SDP Message, and the extended headers and payload that follow the BSDH for a specific SDP Message.

MID[7-0]	Message Name	Extended Header following the Base SDP Header	Packet Contents following the Extended Header (if any)
00000000	Hello	HH	none
00000001	HelloAck	HAH	none
00000010	DisConn	none	none
00000011	AbortConn	none	none
00000100	SendSm	none	none
00000101	RdmaWrCompl	RWCH	none
00000110	RdmaRdCompl	RRCH	none
00000111	ModeChange	MCH	none
00001000	SrcAvailCancel	none	none
00001001	SinkAvailCancel	none	none
00001010	SinkCancelAck	none	none
00001011	ChRcvBuf	CRBH	none
00001100	ChRcvBufAck	CRBAH	none
00001101	SuspComm	SuspCH	none
00001110	SuspCommAck	none	none
00001111	Reserved	n/a	n/a
00010000-00111111	Reserved	n/a	n/a
01000000-01111111	Experimental	n/a	opt. Payload
10000000-11111100	Reserved	n/a	n/a
11111101	SinkAvail	SinkAH	opt. ULP payload
11111110	SrcAvail	SrcAH	opt. ULP payload

11111111	Data	none	opt. ULP payload

Figure 3 SDP Message Definitions

Reserved MID values may be assigned in future versions of the protocol. Experimental values SHOULD NOT be used for permanent assignment.

6.1.2 Flags

Figure 4 MUST be used to define the BSDH Flags field.

Bit Position	Name	Description
0	OOB_PRES	Out-Of-Band Data is present
1	OOB_PEND	Out-Of-Band Data is pending
2	REQ_PIPE	Request change to Pipelined Mode
3-7	reserved	Transmitted as zero and not checked at receiver

Figure 4 BSDH Flags

The SDP implementation MUST set the OOB_PRES bit to one in a Data Message when the last byte of the ULP payload in the SDP Message is OOB data. The OOB_PRES bit is used only in Data Messages and MUST be zero in all other SDP Message types. The Data Message MAY also contain normal ULP payload data before the OOB data. See section 9.5.3 Processing Out-Of-Band Data on page 68.

If the OOB_PEND bit is set to one, then Out-Of-Band data has been sent by the ULP. This bit MAY be set to one in any SDP Message. The actual Out-Of-Band data is not required to be in the current SDP Message. See section 9.5.3 Processing Out-Of-Band Data on page 68 for details on setting and using this bit.

The REQ_PIPE bit is a hint from the Mode Slave to the Mode Master to convey which Flow Control Mode the Mode Slave would prefer the Mode Master to use. If the Mode Slave prefers to be in Combined Mode, it should set the bit to zero. If the Mode Slave prefers to be in Pipelined Mode, it should set the bit to one. The Mode Master is NOT REQUIRED to follow the recommendation from the Mode Slave. The Mode

Master MUST ignore the REQ_PIPE bit if the current Flow Control Mode for this half-connection is Buffered Mode.

The Mode Slave MUST only indicate a Flow Control Mode recommendation using the REQ_PIPE bit (i.e., by setting the bit to either zero or one) in an RdmaRdCompl or RdmaWrCompl Message. The Mode Slave MUST set REQ_PIPE to zero in all other messages. The REQ_PIPE bit MUST only be examined by the Mode Master in an RdmaRdCompl or RdmaWrCompl Message header. The Mode Master MUST NOT check the REQ_PIPE bit in any other SDP Message.

Note that the RdmaRdCompl Message is generated by the Data Sink, and thus setting the REQ_PIPE bit applies to that half-connection. REQ_PIPE in an RdmaWrCompl Message applies to the opposite half-connection. (i.e., the Data Source for this half-connection is currently the Mode Slave in the opposite half-connection).

6.1.3 Buffers (Bufs)

The number of Private Buffers that were currently posted after the last SDP Message was received by the Local Peer, in units of Private Buffers. More precisely, Bufs MUST equal the total number of Private Buffers posted over the lifetime of the connection minus the number of SDP Messages received over the lifetime of the connection. A maximum of 65535 (2^16-1) buffers may be posted at any one time.

6.1.4 Length (Len)

SDP Message length in bytes.

The SDP Message Len MUST be equal to the sum of the sizes of the BSDH, extended header (if present), and ULP payload (if present).

6.1.5 Message Sequence Number (MSeq)

The first SDP Message sent after SDP connection establishment (i.e., the Hello and HelloAck Messages) MUST set the MSeq value to zero. Each successive SDP Message MUST increase the MSeq value by one, and the value MUST wrap to zero after reaching 0xFFFFFFFF.

6.1.6 Message Sequence Number Acknowledgement (MSeqAck)

MSeqAck MUST be set to the sequence number of the last SDP Message received by the Local Peer. See section 9.5.4 SrcAvail Revocation on page 69 for additional constraints.

Until an SDP Message from the Remote Peer is received by the Local Peer, the Local Peer MUST transmit a MSeqAck value of zero.

6.2 Connection Management Messages

SDP connection setup MUST use the Hello and HelloAck Messages to establish an SDP connection. In addition to establishing the connection, these SDP Messages also set up SDP parameters such as the initial credits for the receive Private Buffers, local IRD, and local ORD. See section 8 Connection Setup on page 47 for additional information on how these SDP Messages are used to establish a connection.

SDP connection teardown uses the LLP disconnect mechanisms, plus the DisConn and AbortConn Messages to emulate TCP connection teardown semantics. See section 8.2 Connection Teardown on page 50 for more information.

6.2.1 Hello Message (HH)

The Hello Message MUST contain only a BSDH and a Hello Header(HH). See section 8 Connection Setup on page 47.

The Hello Header format MUST be as defined in Figure 5.

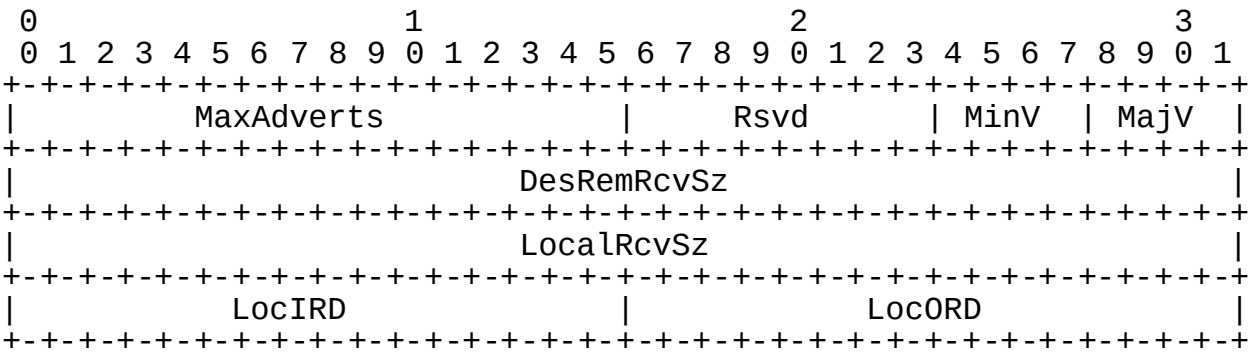

Figure 5 Hello Header

The Hello Message BSDH fields MUST be set as follows:

* MID = Hello.
* Len = size of the BSDH, plus the size of the HH.
* Flags = 0x0.
* Bufs = see 8 Connection Setup on page 47 and section 10.3 Initialization of Send Credit on page 74.
* MSeq = set to zero and not checked on receive.

* MSeqAck = set to zero and not checked on receive.

6.2.1.1 Major Protocol Version Number (MajV) - 4 Bits

For this version of the specification, MajV MUST be set to 1.

The Accepting Peer MUST terminate the connection if MajV in the HH does not match its locally supported value(s).

6.2.1.2 Minor Protocol Version Number (MinV) - 4 Bits

For this version of the specification, MinV MUST be set to 1.

The Accepting Peer should not terminate the connection if the only reason is that MinV in the HH does not match its local value. This enables future protocol extensions that are upwardly compatible.

6.2.1.3 Maximum Advertisements (MaxAdverts) - 16 Bits

The maximum number of concurrent Zcopy advertisements that can be outstanding at the local connection at any one time. This includes SrcAvail advertisements for data transfer from the Remote Peer to the Local Peer and SinkAvail advertisements for data transfer from the Local Peer to the Remote Peer. MaxAdverts MUST be between 1 and 2^16-1, inclusive.

The Accepting Peer MUST terminate the connection if MaxAdverts of the HH is zero.

6.2.1.4 Desired Remote Receive Size (DesRemRcvSz) - 32 Bits

DesRemRcvSz is a hint to the Remote Peer specifying the Local Peer's desired size for the Remote Peer’s receive Private Buffers, in units of bytes (maximum = 2^31 bytes). This field is usually set to the initial size of the local send buffers (assuming the send buffers are all the same size). The Remote Peer SHOULD take this value into consideration when choosing the size of its receive Private Buffers, but it is free to select a different size.

6.2.1.5 Local Receive Size (LocalRcvSz) - 32 Bits

Initial size of the local receive Private Buffers, in units of bytes (maximum = 2^31 bytes).

6.2.1.6 Local IRD (LocIRD) - 16 Bits

LocIRD MUST be set to less than or equal to the depth of the Connection Context's IRD at the Local Peer and MUST be greater than zero.

6.2.1.7 Local ORD (LocORD) - 16 Bits

LocORD MUST be set to less than or equal to the depth of the Connection Context's ORD at the Local Peer and MUST be greater than zero.

6.2.1.8 Rsvd

This field is reserved for future use. These bits MUST be transmitted as zeroes and MUST NOT be checked on receive.

6.2.2 HelloAck Message (HAH)

The HelloAck (Hello Acknowledgement) Message MUST contain only a BSDH and a HelloAck Header (HAH). The HelloAck Message contains a subset of the information sent in the Hello Message. See section 8 Connection Setup on page 47.

The HelloAck Header format MUST be as defined in Figure 6.

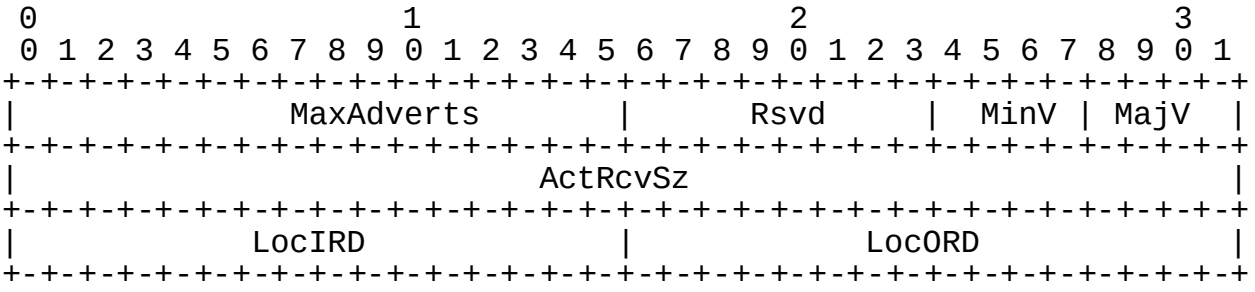

Figure 6 HelloAck Header

The HelloAck Message BSDH fields MUST be set as follows:

* MID = HelloAck.
* Len = size of the BSDH, plus the size of the HAH.
* Flags = 0x0.
* Bufs = see section 8 Connection Setup on page 47 and section 10.3 Initialization of Send Credit on page 74.
* MSeq = set to zero and not checked on receive.
* MSeqAck = set to zero and not checked on receive.

6.2.2.1 Major Protocol Version Number (MajV) - 4 Bits

For this version of the specification, MajV MUST be set to 1.

The Connecting Peer MUST terminate the connection if MajV in the HAH does not match the value that the Connecting Peer supplied in the HH.

6.2.2.2 Minor Protocol Version Number (MinV) - 4 Bits

For this version of the specification, MinV MUST be set to 1.

The Connecting Peer should not terminate the connection if the only reason is that MinV in the HAH does not match its local value. This enables future protocol extensions that are upwardly compatible.

6.2.2.3 Maximum Advertisements (MaxAdverts) - 16 Bits

The maximum number of concurrent Zcopy advertisements that can be outstanding to the Local Peer at any one time. This includes SrcAvail advertisements sent to the Local Peer for data transfer from the Remote Peer to the Local Peer and SinkAvail advertisements for data transfer from the Local Peer to the Remote Peer. MaxAdverts MUST be between 1 and 2^16-1, inclusive.

The Connecting Peer MUST terminate the connection attempt if MaxAdverts of the HAH is set to zero.

6.2.2.4 Actual Receive Size (ActRcvSz) - 32 Bits

The initial size of the local receive Private Buffers, in units of bytes (maximum = 2^31 bytes).

6.2.2.5 Local IRD (LocIRD) - 16 Bits

LocIRD MUST be set to less than or equal to the depth of the Connection Context's IRD at the Local Peer and MUST be greater than zero.

6.2.2.6 Local ORD (LocORD) - 16 Bits

LocORD MUST be set to less than or equal to the depth of the Connection Context's ORD at the Local Peer and MUST be greater than zero.

6.2.2.7 Rsvd

This field is reserved for future use. These bits MUST be transmitted as zeroes and MUST NOT be checked on receive.

6.2.3 DisConn Message

The DisConn (Disconnect Connection) Message informs the Remote Peer that the local ULP will not be sending any more data on this connection, and that the ULP has requested graceful teardown of the socket in the send direction. This is functionally equivalent to TCP sending a FIN packet. See section 8.2 Connection Teardown on page 50.

The DisConn Message MUST contain only a BSDH. It contains no ULP payload.

6.2.4 AbortConn Message

The AbortConn (Abort Connection) Message tells the Remote Peer to ignore an earlier DisConn Message (if received) and to consider socket teardown as abortive. This SDP Message should be sent only if a DisConn Message has been sent earlier. AbortConn is functionally equivalent to TCP setting the RST bit to reset a connection. See section 8.2 Connection Teardown on page 50.

The AbortConn Message MUST contain only a BSDH. It contains no ULP payload.

6.3 Data Transfer and Flow Control Messages

6.3.1 Data Message

The Data Message MUST contain a BSDH and MAY contain ULP payload.

The Data Message is normally used to send ULP payload. A Data Message without ULP payload is a gratuitous credit update. Usually a gratuitous update is used by the Local Peer to update the Remote Peer that there are additional receive Private Buffers available. A gratuitous credit update is done by setting the Bufs field in the BSDH as described in Section 10.4 Gratuitous Update of the Remote Peer’s Send Credit on page 74.

The Data Message is one of three SDP Message types that MAY contain ULP data. The other types are SrcAvail and SinkAvail Messages. For more information on Data Messages, see section 9.1 Bcopy on page 55.

6.3.2 SrcAvail Message (SrcAH)

The SrcAvail (Data Source Available) Message is sent by the Data Source to the Data Sink to inform the latter of the availability of an RDMA Buffer that can be transferred through an RDMA Read operation.

This SrcAvail Message MUST contain a BSDH, a SrcAvail Header (SrcAH), and MAY include a copy of the initial portion of the send RDMA Buffer as payload of the SDP Message, depending upon the Flow Control Mode. (See section 9.2 Read Zcopy on page 56).

The SrcAvail Header format MUST be as defined in Figure 7.

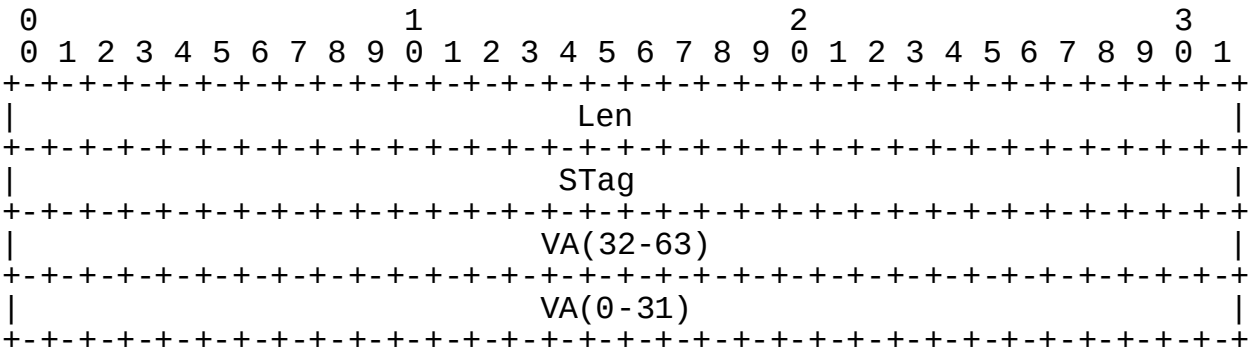

Figure 7 SrcAvail Header (SrcAH)

Note that the word order for VA is designed to be compatible with the InfiniBand Trade Association’s SDP [IBTA-SDP] definition of VA.

6.3.2.1 Length (Len) - 32 bits

Length is the size of the advertised buffer referenced by the STag and VA.

The value of Len MUST be greater than zero and less than or equal to 2^31 bytes.

6.3.2.2 Virtual Address (VA) - 64 bits

The VA defines the beginning of a buffer referenced by the STag, where VA corresponds to Tagged Offset (TO) for iWARP protocols. An SDP implementation MUST support any byte alignment for the VA start address.

The buffer addressed by the RDMA VA MUST include the initial ULP data that was copied into the ULP payload of the SrcAvail Message (if any).

6.3.2.3 STag - 32 bits

The STag field MUST contain the STag corresponding to the VA, which the Data Sink MUST use when retrieving the data from the Data Source into the Data Sink's memory via an RDMA Read.

6.3.3 SinkAvail Message (SinkAH)

The SinkAvail (Data Sink Available) Message is sent by the Data Sink to the Data Source to inform the latter of the availability of an RDMA Buffer that can be filled through an RDMA Write operation. See section 9.3 Write Zcopy on page 61.

The SinkAvail Header format MUST be as defined in Figure 8.

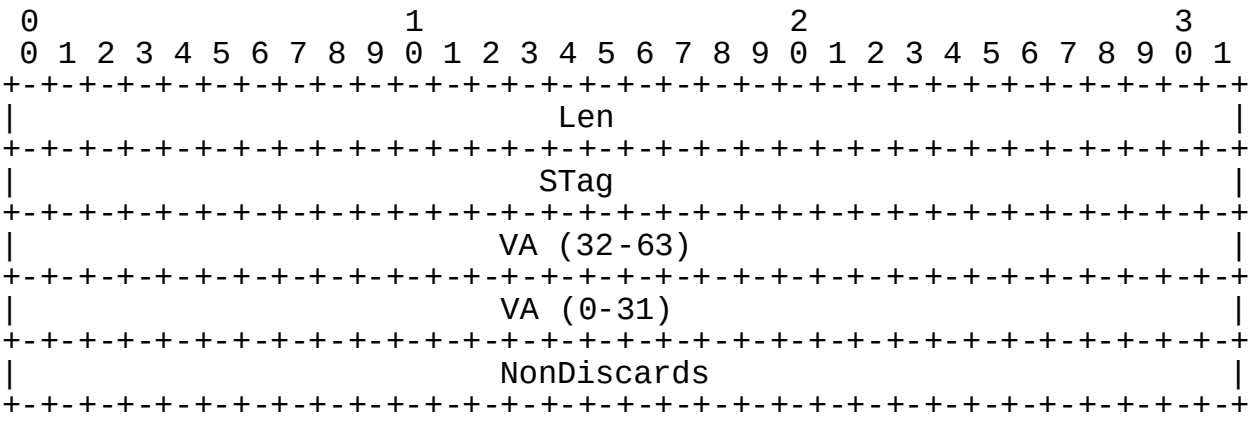

Figure 8 SinkAvail Header (SinkAH)

The SinkAvail Message MUST contain a BSDH and SinkAH, and MAY include some send data as ULP payload for data flow in the opposite direction. See section 9.4 Transaction Mechanism on page 63.

Note that the word order for VA is designed to be compatible with the InfiniBand Trade Association's SDP definition [IBTA-SDP] of VA.

6.3.3.1 Length (Len) - 32 bits

The Len field MUST contain the size of the advertised buffer referenced by the STag and VA.

The value of Len MUST be greater than zero and less than or equal to 2^31 bytes.

6.3.3.2 Virtual Address (VA) - 64 bits

The VA MUST contain the start address of the receive RDMA Buffer, where VA corresponds to Tagged Offset (TO) for iWARP protocols. The VA MAY start on any byte boundary.

6.3.3.3 STag - 32 bits

The STag field MUST contain the iWARP STag corresponding to the VA, which the Data Source MUST use when sending the data into the memory of the Data Sink via an RDMA Write.

6.3.3.4 NonDiscards - 32 bits

The NonDiscards field in the SinkAvail Message contains the Data Sink's current local value for NonDiscards. After connection setup, the Data Sink MUST initialize the local value for NonDiscards to zero. The Data Sink MUST increment its local value for NonDiscards when an SDP Message that is carrying ULP payload is received and the SDP Message did not cause the Data Sink to discard a previously sent SinkAvail Message. This count MUST wrap around to zero after reaching 0xFFFFFFFF.

See section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65 for additional information.

6.3.4 RDMA Messages

SDP uses the iWARP RDMA Write and RDMA Read Messages to transfer Zero-copy data.

6.3.5 SendSm Message

The Data Source uses a SrcAvail Message to inform the Data Sink of data that can be transferred using RDMA. If the Data Sink is unable or unwilling to transfer this data using RDMA, it MUST use the SendSm (Send Small) Message to force the Data Source to send the data using the Bcopy Transfer Mechanism. See section 9.5.2 Mechanisms for Forcing Bcopy on page 66.

The SendSm Message MUST contain only a BSDH. It contains no ULP payload.

6.3.6 RdmaWrCompl Message (RWCH)

The RdmaWrCompl (RDMA Write Complete) Message is sent by the Data Source to inform the Data Sink of completion of an RDMA Write transfer. See section 9.3 Write Zcopy on page 61.

The RdmaWrCompl Header format MUST be as defined in Figure 9.

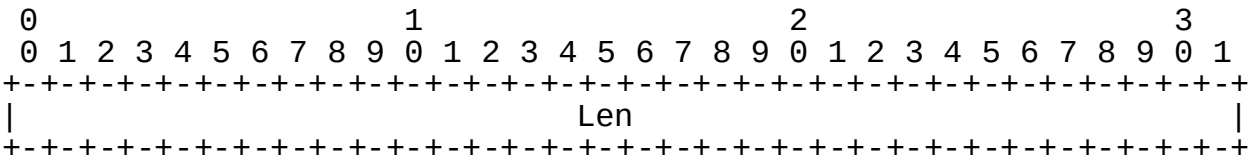

Figure 9 RdmaWrCompl Header (RWCH)

The RdmaWrCompl Message MUST contain only a BSDH and a RWCH.

In addition, the sender (Data Source) must indicate its preferred Flow Control Mode to the Data Sink via the REQ_PIPE bit in the Flags field of the BSDH in the RdmaWrCompl Message. See section 6.1.2 Flags on page 15 for a description of REQ_PIPE usage.

6.3.6.1 Length (Len) - 32 Bits

Len MUST contain the number of bytes transferred to the Data Sink's RDMA Buffer through RDMA Write(s) for the oldest outstanding SinkAvail. Len MAY be less than the size of the RDMA Buffer advertised by the Data Sink in the SinkAvail Message.

6.3.7 RdmaRdCompl Message (RRCH)

The RdmaRdCompl (RDMA Read Complete) Message is sent by the Data Sink to inform the Data Source of completion of an RDMA Read transfer. See section 9.2 Read Zcopy on page 56.

The RdmaRdCompl Header format MUST be as defined in Figure 10.

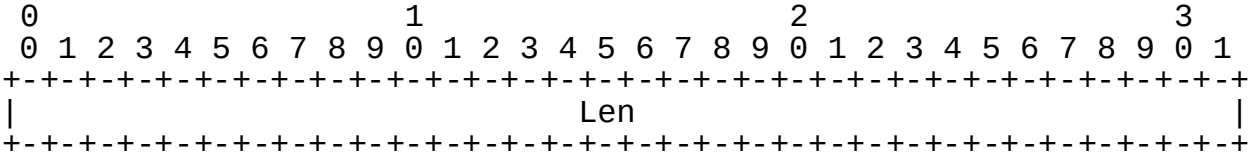

Figure 10 RdmaRdCompl Header (RRCH)

The RdmaRdCompl Message MUST contain only the BSDH and RRCH.

In addition, the sender (Data Sink) must indicate its preferred Flow Control Mode to the Data Source via the REQ_PIPE bit in the Flags field of the BSDH in the RdmaRdCompl Message. See section 6.1.2 Flags on page 15 for a description of REQ_PIPE usage.

6.3.7.1 Length (Len) - 32 Bits

Len MUST contain the size (in bytes) transferred to the Data Sink's RDMA Buffer through RDMA Read(s) for the oldest outstanding SrcAvail. Len MUST NOT include the portion of the buffer (if any) transferred within the SrcAvail Message as ULP data payload. If there is ULP data in the SrcAvail Message, Len MUST be less than the size of the Data Source RDMA Buffer advertised in the SrcAvail Message by exactly the number of ULP payload bytes included in the SrcAvail Message.

6.3.8 ModeChange Message (MCH)

The ModeChange Message is used to inform the Remote Peer of a Flow Control Mode transition. See section 11 SDP Flow Control Modes on page 78.

The ModeChange Header format MUST be as defined in Figure 11.

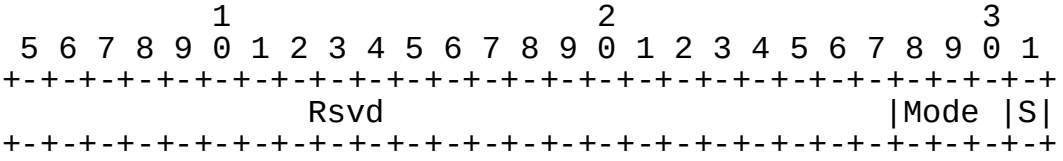

Figure 11 ModeChange Header (MCH)

The ModeChange Message MUST contain only a BSDH and MCH.

The receiver of the ModeChange Message MUST change its send or receive Flow Control Mode to the new Mode specified in the ModeChange Message.

6.3.8.1 S - 1 bit

Specifies whether the peer receiving the ModeChange Message needs to change its Flow Control Mode for its send half-connection (S = 1) or receive half-connection (S = 0), for the connection over which the ModeChange Message was received.

6.3.8.2 Mode - 3 bits

The Mode field specifies the new Mode.

The ModeChange MCH field value MUST be as defined in Figure 12.

Mode Value	Name	Description
0	BUFF_MODE	New Mode should be Buffered Mode
1	COMB_MODE	New Mode should be Combined Mode
2	PIPE_MODE	New Mode should be Pipelined Mode
3-7	reserved	Reserved value

Figure 12 MCH Mode Values

6.3.8.3 Rsvd - 28 Bits

This field is reserved for future use. These bits MUST be transmitted as zeroes and MUST NOT be checked on receive.

6.3.9 SrcAvailCancel Message

The SrcAvailCancel (Data Source Available Cancel) Message is sent by the Data Source to ask the Data Sink to ignore all SrcAvail advertisements sent by the Data Source that are Unprocessed by the Data Sink. See section 9.5.4 SrcAvail Revocation on page 69.

The SrcAvailCancel Message MUST contain only a BSDH.

6.3.10 SinkAvailCancel Message

The SinkAvailCancel (Data Sink Available Cancel) Message is sent by the Data Sink to ask the Data Source to ignore all SinkAvail advertisements sent by the Data Sink that are Unprocessed by the Data Source. See section 9.5.5 SinkAvail Revocation on page 70.

The SinkAvailCancel Message MUST contain only a BSDH.

6.3.11 SinkCancelAck Message

The SinkCancelAck (Data Sink Available Cancel Acknowledgement) Message is sent by the Data Source in response to the SinkAvailCancel Message. The Data Source MUST send a SinkCancelAck after it has canceled all Unprocessed SinkAvail advertisements. See section 9.5.5 SinkAvail Revocation on page 70.

The SinkCancelAck Message MUST contain only a BSDH.

6.4 Private Buffer Resizing Messages

6.4.1 ChRcvBuf Message (CRBH)

The ChRcvBuf (Change Receive Private Buffer Size) Message is sent by the Data Source to the Data Sink to request a change in the size of the latter's receive Private Buffers. See section 10.6 Receive Buffer Resizing on page 75.

The ChRcvBuf Message MUST contain only a BSDH and a CRBH.

The ChRcvBuf Header format MUST be as defined in Figure 13.

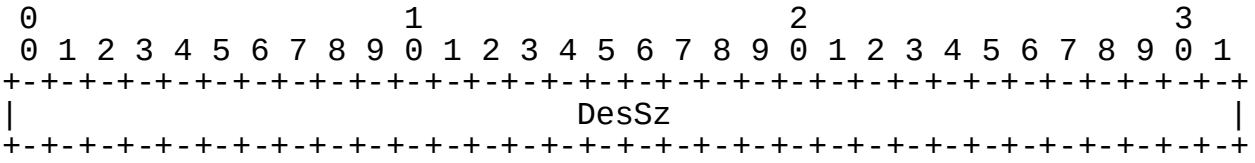

Figure 13 ChRcvBuf Header (CRBH)

6.4.1.1 Desired Size (DesSz) - 32 bits

Desired size (in bytes) of the Data Sink's receive Private Buffers.

6.4.2 ChRcvBufAck Message (CRBAH)

The ChRcvBufAck (Change Receive Private Buffer Size Acknowledgement) Message is sent in response to the ChRcvBuf Message. The ChRcvBufAck Message informs the Data Source of the new size of the receive Private Buffers. See section 10.6 Receive Buffer Resizing on page 75.

The ChRcvBufAck Message MUST contain only a BSDH and a CRBAH.

The ChRcvBufAck Header format MUST be as defined in Figure 14.

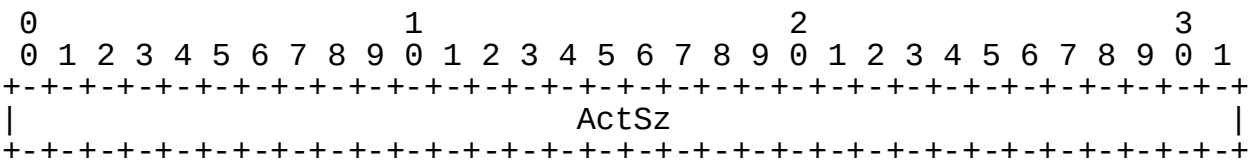

Figure 14 ChRcvBufAck Header (CRBAH)

6.4.2.1 Actual Size (ActSz) - 32 bits

This is the actual or new size (in bytes) of the local receive Private Buffers. The actual size MAY be the same as the size prior

to receipt of the ChRcvBuf Message if the protocol implementation does not wish to resize its receive Private Buffers.

6.5 Socket Duplication Messages

6.5.1 SuspComm Message

The SuspComm (Suspend Communication) Message is sent to ask the Remote Peer to suspend communication as part of preparing the socket for duplication. See section 13 Socket Duplication on page 92.

The SuspComm Message MUST contain only a BSDH and a SuspCH.

The SuspComm Header format MUST be as defined in Figure 15.

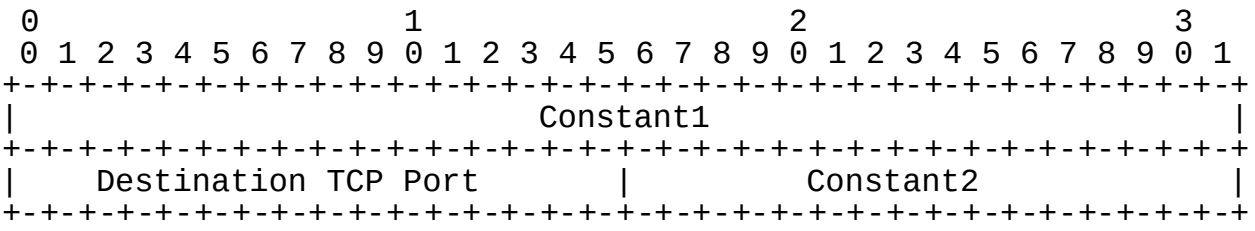

Figure 15 SuspComm Header (SuspCH)

To facilitate interoperability and simplified bridging between SDP over iWARP and SDP as defined by the InfiniBand Trade Association [IBTA-SDP], the SuspComm header defines a 64-bit field composed of three components: a 16-bit TCP port and two constants. The two constants when concatenated together [Constant1, Constant2] form a 48-bit constant value that corresponds to the 48-bit constant of the SuspComm header defined in the InfiniBand Trade Association SDP specification.

6.5.1.1 Destination TCP Port - 16 Bits

The Destination TCP port that the Remote Peer should try to connect with to re-establish the connection with the Local Peer after duplication has completed.

6.5.1.2 Constant1 - 32 bits

Constant1 MUST be set to 0x0.

6.5.1.3 Constant2 - 16 bits

Constant2 MUST be set to 0x1.

6.5.2 SuspCommAck Message

The SuspCommAck (Suspend Communication Acknowledgement) Message is sent in response to the SuspComm Message. This SDP Message informs the peer that all communication has been suspended as requested by the peer in its SuspComm Message. See section 13 Socket Duplication on page 92.

The SuspCommAck Message MUST contain only a BSDH.

7 Address Resolution using the SDP Port Mapper Protocol

A key objective of SDP is to transparently operate underneath SOCK_STREAM applications. SDP is intended to allow an application to advertise a service using its application-defined listen port and transparently connect using an SDP RDMA-capable listen port. However, if the SDP Connecting Peer does not know the port and IP address to use when creating a connection for SDP communication, it must resolve the TCP port and IP address used for traditional SOCK_STREAM communication to a TCP port and IP address that can be used for SDP communication.

This section defines the SDP Port Mapper protocol, which enables the Connecting Peer, through a PM Client, to negotiate with an SDP Port Mapper Service to find the TCP port and IP address which the Connecting Peer should use to connect to the SDP mapping of the TCP application.

Figure 16 depicts the relationships between the different entities involved in the SDP Port Mapper protocol, and the terminology used in this chapter to refer to those entities. The PMSP (PM Server) and the PM Client communicate using the Port Mapper protocol. The Accepting Peer and Connecting Peer use the results from the Port Mapper protocol to initiate LLP Connection Setup.

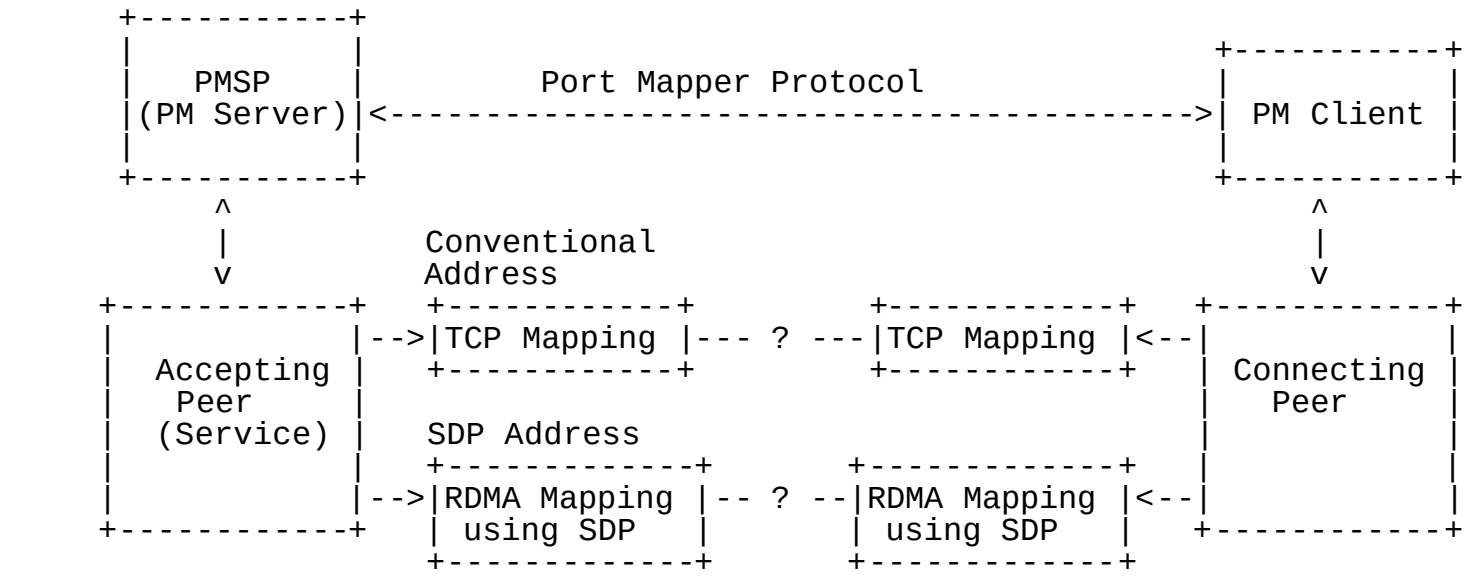

Figure 16 Port Mapper Protocol Entities

Setting up an SDP Connection using the Port Mapper protocol is done in two stages. From the Client’s perspective, the first stage is performed by the PM Client to discover what address should be used for LLP Connection setup (either the SDP Address or the Conventional

Address) to the service. Note that the mechanism for discovery of the original service's address is outside the scope of this specification. The second stage occurs when the Connecting Peer attempts to connect to the service using the address negotiated in the first stage. Thus the Connecting Peer, as a result of the Port Mapper protocol, will attempt to setup an LLP Connection to the SDP Address, which will cause SDP Connection Setup to be initiated, or it will attempt to setup an LLP Connection to the Conventional Address, which will cause traditional streaming mode communication to be used.

7.1 Definitions for Address Resolution

Conventional Address - the original port and IP address, which the client attained by some means outside the scope of this paper. If the Port Mapper Service Provider denies access via SDP, then it is intended that the client fall back to the Conventional Address for connection. After connection setup to the Conventional Address, all communication is done in streaming mode (i.e. SDP is not used).

Final SDP Address - a port and IP address, which was returned as a result of the Port Mapper protocol (in the PMAccept Message).

PMSP - Port Mapper Service Provider.

Port Mapper Service Provider (PMSP) - A service that returns the SDP listen port and IP address (i.e. SDP Address), if any, that the Connecting Peer may use to establish an SDP connection with the specified Accepting Peer.

SDP Address - the port and IP address, which the client uses to create an LLP connection, initialize SDP, and then transfer ULP data using SDP. All communication after the initial SDP Hello Message is in iWARP mode.

7.2 Port Mapper Service Requirements

The following list of requirements drove the design of the SDP Port Mapper Protocol:

Main Goals:

1. When the SDP Port Mapper protocol has completed, the net result will be that both the Connecting Peer and the Accepting Peer will unambiguously have either an SDP over iWARP connection, or have a standard TCP connection.

* All of the state at both ends of the connection is recovered after the protocol completes, regardless of the number of lost packets or timeouts.

2. The Port Mapper must enable SDP to use the TCP port name space to establish an LLP connection between a Connecting Peer application instance and an Accepting Peer application instance.

3. The Port Mapper protocol must enable interoperable implementations that support timeouts and retransmissions of Port Mapper Messages.

Other goals of the protocol:

1. Enable the Port Mapper Service Provider to provide load balancing and failover capabilities by manipulating the returned IP address and TCP port number.

2. Enable the Port Mapper Service to direct the Connecting Peer to target the Accepting Peer's advertised listen port or target a dynamically mapped SDP listen port for connection establishment.

3. A Connecting Peer may transparently invoke the SDP Port Mapper without requiring application modification.

4. Address denial of service issues that can occur because UDP is used to encapsulate the Port Mapper Protocol.

5. Enable, but do not require, that the Port Mapper Service can, through some private mechanism, be able to verify that the service being negotiated is actually running on the target machine to be returned to the Port Mapper Client via the IP address.

6. Enable the Port Mapper Service to allow the Connecting Peer to cache the returned TCP Port and IP address for a specific amount of time. This reduces the overhead associated with creating an SDP connection because the Port Mapper protocol does not have to be run before the SDP connection is setup. However, it only helps if the Connecting Peer is attempting to connect to a specific service at a specific IP address multiple times before the cached entry times out.

7. The PM Server may be implemented using either a centralized (e.g., a central management agent acting on behalf of one or more Accepting Peers) or a distributed mechanism (e.g. point-to-point Connecting Peer to Accepting Peer).

8. The PM Client may be implemented as an agent acting on behalf of the Connecting Peer or be implemented as part of the Connecting Peer.

9. Provide support for either IPv4 (IETF RFC 791) or IPv6 (IETF RFC 2460) based SDP services.

10. Ensure that if a host crashes and comes back up, that all state is resynchronized.

The protocol analysis in this specification assumes that the client TCP port namespace for a specific source IP address is unique within the host. If this is not true, additional analysis should be done to ensure relaxing this assumption maintains robust behavior.

7.3 SDP Port Mapper Message Format

The SDP Port Mapper uses a single message format for the four message types that are exchanged. All Port Mapper messages MUST have the format defined in Figure 17 Port Mapper Message Format.

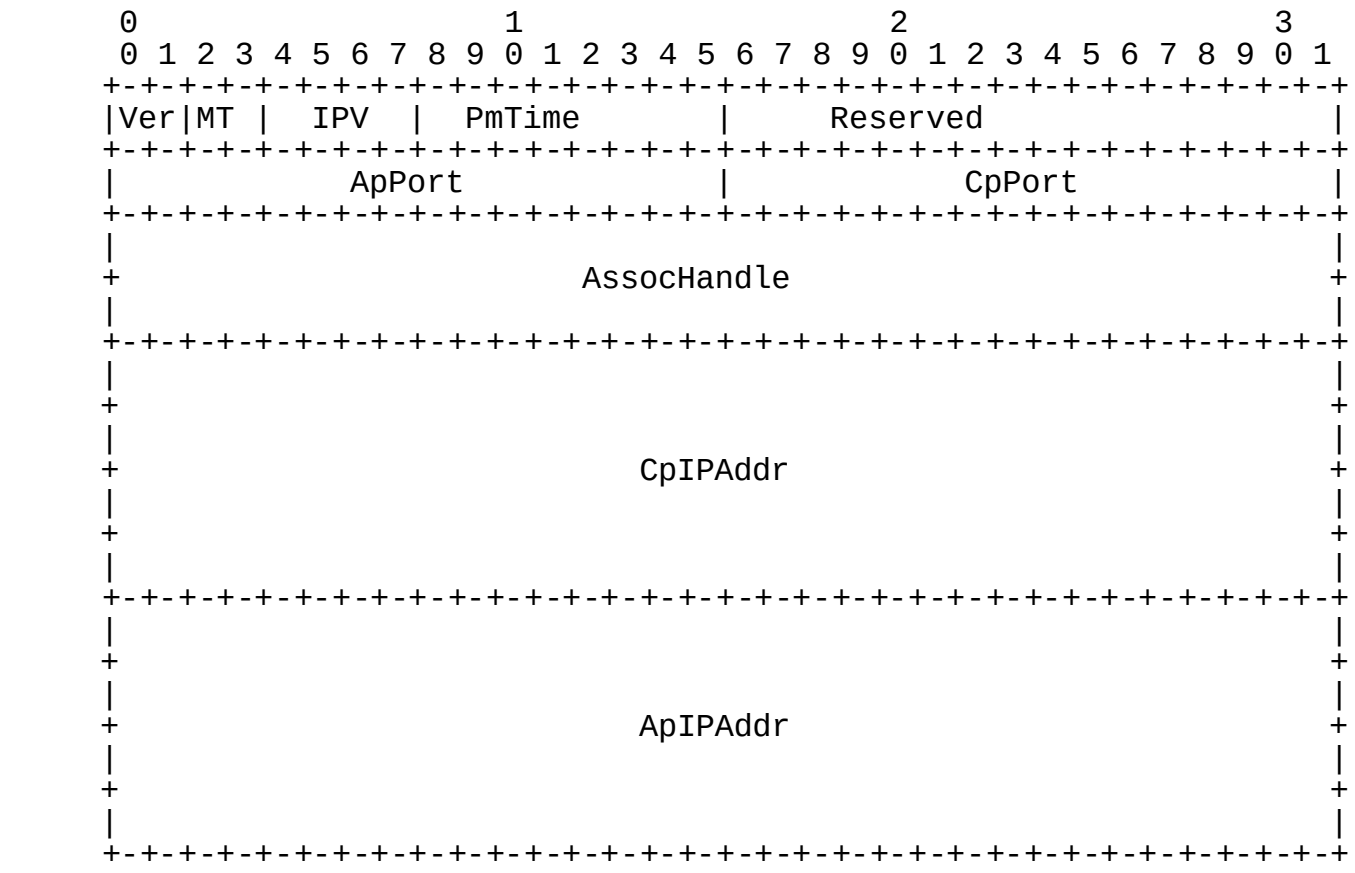

Figure 17 Port Mapper Message Format

7.3.1.1 Version (Ver) - 2 Bits

Version indicates the version of the SDP Port Mapper protocol. For this version of the specification, Ver MUST be set to zero.

7.3.1.2 Message Type (MT) - 2 Bits

Message Type indicates the Port Mapper message being exchanged. A Port Mapper message MUST use one of the following values:

* MT = 0x0 - Port Mapper Request Message (PMReq)

* MT = 0x1 - Port Mapper Accept Message (PMAccept)

* MT = 0x2 - Port Mapper Acknowledgement Message (PMAck)

* MT = 0x3 - Port Mapper Deny Message (PMDeny)

7.3.1.3 IP Version (IPV) - 4 Bits

IPV indicates the version of the IP address used by the Connecting Peer. Only 0x4 and 0x6 are valid values for IPV, as defined below. All other values are reserved.

* IPV = 0x4 indicates an IPv4 address is used, and only the first 32-bits, in network byte order, of the CpIPAddr and the ApIPAddr fields are valid. The remaining 96 bits are transmitted as zeroes and are ignored on receipt.

* IPV = 0x6 indicates an IPv6 address is used, i.e., all 128-bits of the CpIPAddr and the ApIPAddr fields are valid.

7.3.1.4 PmTime - 8 Bits

PmTime indicates the total time since the receipt of the PMAccept Message that the PM Client may cache the response data. When the PmTime has elapsed, the PM Client MUST flush the response data. If the Connecting Peer requests an SDP Address from the PM Client after the PM Client's cached response data has been flushed, the PM Client MUST issue a new PMReq Message.

* PmTime = 0 indicates the Connecting Peer is only allowed to use the returned service port for a single connection operation and MUST NOT cache the PMAccept Message ApPort and ApIPAddr. The Accepting Peer implementation may bound the time it treats the mapping as valid; therefore, it is strongly recommended that the Connecting Peer initiate the LLP connection establishment subsequent to the receipt of the PMAccept and issuance of the PMAck Message.

* PmTime = [1,127], (PmTime * 250) is the number of milliseconds that the Connecting Peer MAY cache the PMAccept Message ApPort.

* PmTime = [128, 254], (PmTime * 1000) is the number of milliseconds that the Connecting Peer MAY cache the PMAccept Message ApPort.

* PmTime = 255 indicates that the Connecting Peer MAY permanently cache the PMAccept Message ApPort.

When determining PmTime, the PM Server and Accepting Peer should account for the expected total time to process the PMAccept, issue a PMAck, initiate LLP connection establishment, and transmit the LLP connection request.

7.3.1.5 Reserved - 16 Bits

Reserved field. Reserved MUST be transmitted as zero by the sender and MUST NOT be checked by the receiver.

7.3.1.6 Accepting Peer Port (ApPort) - 16 Bits

* For the PMReq Message, ApPort = requested server port number, i.e. the known listen port of the associated service to be mapped.

* For the PMAccept Message, ApPort = mapped server port, i.e. the SDP listen port.

* For the PMAck Message, ApPort = mapped server port, i.e. the SDP listen port.

* For the PMDeny Message, ApPort is undefined, and MUST be transmitted as zeroes and ignored on receive.

7.3.1.7 Connecting Peer Port (CpPort) - 16 Bits

* For the PMReq Message, CpPort = local TCP port number for the Connecting Peer.

* For the PMAccept Message, CpPort = the same value that was sent in the PMReq Message.

* For the PMAck Message, CpPort = the same value that was sent in the PMAccept Message.

* For the PMDeny Message, CpPort = the same value that was sent in the PMReq Message.

7.3.1.8 Association Handle (AssocHandle) - 64 Bits

The AssocHandle is an opaque identifier that MUST be set by the PM Client in PMReq Message. The AssocHandle is reflected in the associated PMAccept Message, PMAck Message, and PMDeny Message. The AssocHandle may be used to delineate multiple in-flight Port Mapper transactions from one another - a transaction is defined as the Port Mapper two-way or three-way message exchange.

7.3.1.9 Connecting Peer IP Address (CpIPAddr) - 128 Bits

In the PMReq Message this field contains the Connecting Peer’s IP address to be used for SDP session establishment. The CpIPAddr is reflected in the associated PMAccept Message, PMAck Message, and PMDeny Message.

Note that the CpIPAddr format is defined by the IP version. See section 7.3.1.3 for the format.

7.3.1.10 Accepting Peer IP Address (ApIPAddr) - 128 Bits

In the PMReq Message, this field contains the IP address of the Accepting Peer that will be used to establish a communication session with the Connecting Peer (e.g. the client is the Connecting Peer and the server is the Accepting Peer). The ApIPAddr may be changed by the PMSP in the corresponding PMAccept Message. The ApIPAddr value in the PMAck Message is the same as was specified in the corresponding PMAccept Message. The ApIPAddr value in the PMDeny Message is the same as was specified in the corresponding PMReq Message.

The ApIPAddr format is defined by the IP version. See section 7.3.1.3 for the format.

7.4 Operational Overview

The Port Mapper protocol uses either a two-way or a three-way UDP/IP (datagram) [UDP] message exchange between the PM Client and the Port Mapper service provider (PMSP) acting on behalf of the Accepting Peer or the Accepting Peer itself. The PMReq Message's destination UDP port number MUST by default be as defined in Section 16 IANA Considerations on page 100.

The first message exchanged in either case is a PMReq Message, which MUST be issued by the PM Client and MUST be encapsulated within UDP. The PMReq Message fields MUST be set by the PM Client as follows:

* Ver - MUST be set to 0x0 for this version of the specification.
* IPV - MUST be set to either 0x4 if the CpIPAddr and ApIPAddr are an IPv4 address or 0x6 if the CpIPAddr and ApIPAddr are IPv6 addresses.
* MT - MUST be set to 0x0.
* PmTime - MUST be set to zero and ignored on receive.
* ApPort - MUST be set to the listen port for the associated service.
* CpPort - MUST be set to the local TCP Port number that the Connecting Peer will use when connecting to the service.
* AssocHandle - MUST be set by the Connecting Peer to a unique value to differentiate in-flight transactions.

* CpIPAddr - MUST be set to the Connecting Peer’s IP address that will initiate LLP connection establishment.

* ApIPAddr - MUST be set to the target Accepting Peer’s IP address to be used in connection establishment.

* A PMReq Message MUST be transmitted by the PM Client using UDP/IP to target the IANA Port Mapper service provider port as defined in Section 16 IANA Considerations on page 100.

If the port mapping operation is successful, the PM Server MUST return a PMAccept Message.

The PMAccept Message MUST be encapsulated within UDP using the UDP Ports and IP Address information contained within the corresponding PMRequest Message. The PMAccept fields MUST be set by the PM Server and as follows:

* Ver - MUST be set to 0x0 for this version of the specification.

* IPV - MUST be set to the same value as the IPV field in the PMReq Message.

* MT - MUST be set to 0x1.

* PmTime - MUST be set to a value within the defined range. See section 7.3.1.4.

* ApPort - MUST be set to the SDP listen port.

* CpPort - MUST be set to the same value as the CpPort field in the corresponding PMReq Message.

* AssocHandle - MUST be set to the same value as the AssocHandle field in the corresponding PMReq Message.

* CpIPAddr - MUST set to the same value as the CpIPAddr field in the corresponding PMReq Message.

* ApIPAddr - MUST be set to the Accepting Peer's IP address to be used in connection establishment. The Accepting Peer MAY return a different ApIPAddr than requested in the corresponding PMReq Message.

* A PMAccept Message MUST be transmitted using the address information contained in the UDP/IP headers used to deliver the corresponding PMReq Message.

Upon receipt of a PMAccept Message, the PM Client MUST return a PMAck Message.

The PMAck Message MUST be encapsulated within UDP using the UDP Ports and IP Address information contained within the corresponding PMAccept or PMDeny Message. The PMAck Message fields MUST be set by the PM Client as follows:

* Ver - MUST be set to 0x0 for this version of the specification.

* IPV - MUST be set to the same value as the IPV field in the corresponding PMAccept Message.

* MT - MUST be set to 0x2.

* PmTime - MUST be set to zero and ignored on receive.

* ApPort - MUST be set to the same value as the ApPort field in the corresponding PMAccept Message.

* CpPort - MUST be set to the same value as the CpPort field in the corresponding PMAccept Message.

* AssocHandle - MUST be set to the same value as the AssocHandle field in the corresponding PMAccept Message.

* CpIPAddr - MUST be set to the same value as the CpIPAddr field in the corresponding PMAccept Message. An Accepting Peer implementation may use the CpIPAddr to validate the subsequent LLP connection request through association of the CpIPAddr with the ApPort returned in the corresponding PMAccept Message.

* ApIPAddr - MUST be set to the same value as the ApIPAddr field in the corresponding PMAccept Message.

* A PMAck Message MUST be transmitted using the address information contained in the UDP/IP headers used to deliver the PMAccept Message.

A PMAck Message MUST be transmitted by the PM Client using the address information contained in the UDP/IP headers used to deliver the PMAccept Message. The three-way message exchange is illustrated in Figure 18 Three-way Port Mapper Message Exchange:

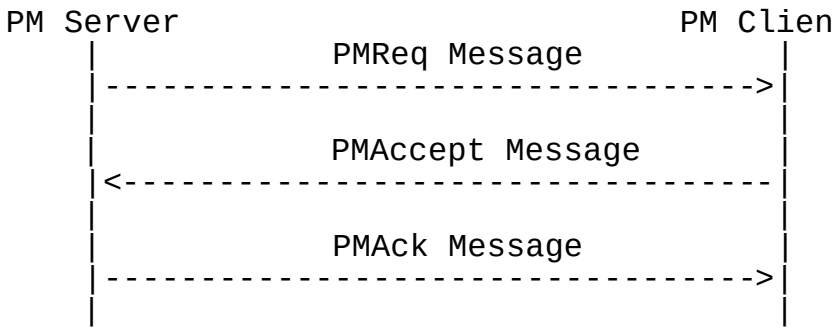

Figure 18 Three-way Port Mapper Message Exchange

A three-way message exchange is used for the following reasons:

1. It supports either centralized or distributed (peer-to-peer) Port Mapper implementations while minimizing the number of packets exchanged between the Connecting Peer and the Accepting Peer.

   The flexibility afforded by the Port Mapper messages enables a wide variety of interoperable implementation options. For example:

   * The PM Client may be implemented as an agent acting on behalf of the Connecting Peer or be implemented as part of the Connecting Peer.

   * The PM Server may be implemented as an agent acting on behalf of the Accepting Peer or be implemented as part of the Accepting Peer.

   * The ApIPAddr field within the PMAccept Message may be different than the requested IP Address (i.e. the ApIPAddr field in the PMRequest) due to local policy decisions. For example, if the Accepting Peer contains multiple network interfaces, and its local policy supports network interface load balancing, then the Accepting Peer may return a different ApIPAddr for the selected target interface than was requested in the PMReq Message.

2. It allows an Accepting Peer to dynamically create the SDP listen port and know that the Connecting Peer will utilize this port only within the specified time period. The Accepting Peer MAY release the associated resources upon the time period expiring, if a PMAck Message is not received. The ability to release resources minimizes the impact of a denial of service attack via consumption of a SDP listen port. For additional information see Section 15.2.1 Port Flooding on page 98.

If the port mapping operation is not successful, the Accepting Peer MUST return a PMDeny Message. The PMDeny Message MUST be encapsulated within UDP using the UDP Ports and IP Address information contained within the corresponding PMRequest Message. The PMDeny Message fields MUST be set by the Accepting Peer as follows:

* Ver - MUST be set to 0x0 for this version of the specification.

* IPV - MUST be set to the same value as the IPV field in the PMReq Message.

* MT - MUST be set to 0x3.

* PmTime - MUST be set to zero and ignored on receive.

* ApPort - MUST be set to the same value as the ApPort field in the corresponding PMReq Message.

* CpPort -MUST be set to the same value as the CpPort field in the corresponding PMReq Message.

* AssocHandle - MUST be set to the same value as the AssocHandle field in the corresponding PMReq Message.

* CpIPAddr - MUST be set to the same value as the CpIPAddr field in the corresponding PMReq Message.

* ApIPAddr - MUST be set to the same value as the ApIPAddr field in the corresponding PMReq Message.

* A PMDeny Message MUST be transmitted using the address information contained in the UDP/IP headers used to deliver the PMReq Message.

Upon receipt of a PMDeny Message, the PM Client MUST treat the associated Port Mapper transaction as complete and MUST not issue a PMAck Message. A Port Mapper operation may fail for a variety of reasons, e.g., there is no such service mapping, resource exhaustion, etc.

The two-way message exchange is illustrated in Figure 19:

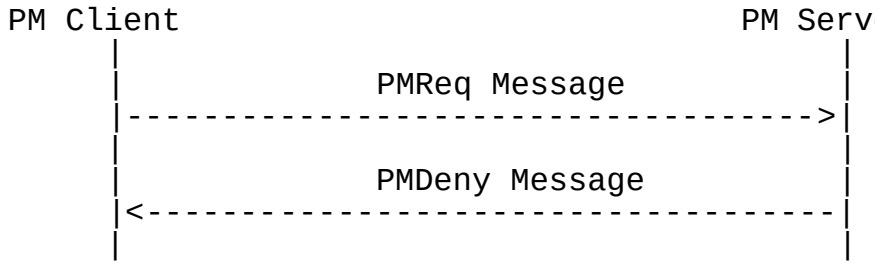

Figure 19 Two-way Port Mapper Message Exchange

7.5 Lost Messages, Timeouts, and Other Error Cases

The Port Mapper Protocol requires the PM Client to implement timeouts on transactions, and it is RECOMMENDED the PM Server implement timeouts to recover state because the Port Mapper protocol is encapsulated on top of UDP - an unreliable protocol. The behavior requirements in this section ensure that regardless of whether the PM Server implements timeouts, both Peers will interoperate. Because timeouts can occur, retransmissions of Port Mapper Messages are possible.

The PM Server detects duplicate PMReq Message by comparing the 5-tuple (AssocHandle, ApPort, CpPort, ApIPAddr, CpIPAddr) of the message with the five-tuples within its internal PM transaction state. If the five values match the internal state, then the PM Server MUST treat the incoming PMReq Message as a duplicate message.

The PM Client detects duplicate PMAccept or PMDeny messages by comparing the 3-tuple (AssocHandle, CpPort, CpIPAddr) of the message with the 3-tuples within its internal PM transaction state. If the three values match the internal state, then the PM Client MUST treat the incoming PMAccept or PMDeny Message as a duplicate message.

7.5.1 PM Client Behavior

The combination of the PM Client and the Connecting Peer MUST select the combination of the AssocHandle, CpIPAddr, and CpPort to ensure that it is unique within the maximum lifetime of a packet on the network. This ensures that the PMSP will not see delayed duplicate messages.

The PM Client MUST arm a timeout when transmitting a PMReq Message. If a timeout occurs for the reply to the PMReq message (i.e. neither a corresponding PMAccept nor a PMDeny Message was received before the timeout occurred), the PM Client MUST retransmit the PMReq Message and re-arm the timeout, up to a maximum number of retransmissions (timeouts).

The PM Client MUST use the same AssocHandle, ApPort, ApIPAddr, CpPort, and CpIPAddr on any retransmissions of PMReq. The initial AssocHandle chosen by a host SHOULD be chosen at random to make it harder for a third party to interfere with the protocol. The combination of the AssocHandle, ApPort, CpPort, ApIPAddr, and CpIPAddr MUST be unique within the host associated with the Connecting Peer. This enables the PMSP to differentiate between client requests.

If the PM Client does not get an answer from the PMSP after the maximum number of timeouts, the PM Client SHOULD stop attempting to connect to an SDP Address and instead use the Conventional Address for LLP connection setup. Conventional LLP connection setup will cause streaming mode data transfer to be initiated. The specifics of how the fall back to the Conventional Address is done are outside the scope of this specification.

If the PM Client receives a LLP Connection Reset (e.g. TCP RST segment) when attempting to connect to the SDP Address, it SHOULD view this as equivalent to receiving a PMDeny Message, and thus attempt to connect to the service using the Conventional Address.

If the PM Client receives a reply to a PMReq Message, and later receives another reply for the same request, the PM Client MUST discard any additional replies (PMAccept or PMDeny) to the request.

If the PM Client receives a PMAccept or PMDeny and has no associated state for the Message, the Message MUST be discarded.

7.5.2 PM Server Behavior

The PMSP SHOULD arm a timer when it sends a PMAccept Message, to be disabled when either a PMAck or LLP connection setup request (e.g. TCP SYN) to the SDP Address has occurred. If a PMAck Message or LLP Connection setup request is not received before the end of the timeout interval, all resources associated with the PMReq MUST be deleted. This protects against certain denial-of-service attacks. Note that if the PM Server was implemented on a different host than the Accepting Peer and the PMAck was lost, the PM Server would not be able to observe the LLP Connection setup request - thus the timer could expire after an SDP connection has been setup to the SDP Address. Therefore if the timer expires the only state that should be cleaned up is state associated with the connection setup request - and an existing, established LLP connection to the SDP Address should be unaffected.

If the PMSP detects a duplicate PMReq Message, it MUST reply with either a PMAccept or a PMDeny Message. In addition, if the PMSP armed a timer when it sent the previous PMAccept Message for the

duplicated PMReq Message, it should reset the timer when resending the PMAccept Message.

Because the PM Client can retransmit PMReq Messages, and the PMSP will send a reply for each request, it is possible to get duplicate PMAccept or PMDeny Messages. Because the PM Client will discard additional replies from the PMSP for a specific transaction, presumably there will not be duplicate PMAck Messages. However, to cover the condition where there is a race in the client that could cause multiple PMAck Messages to be sent for a specific transaction (or an early timeout by the PMSP), if a PMSP receives a PMAck Message which does not relate to any known state, it MUST discard the PMAck message. Note that per TCP semantics, if a TCP SYN segment is received at the SDP Port Address which that does not relate to any known state, TCP will send back to the Connecting Peer a TCP Reset segment. Thus the Connecting Peer is able to recover its state and tear down the connection request.

When the PMSP is attempting to attach the Connecting Peer to a service, the service can have one of two states - available or unavailable. If a PMSP receives a duplicate PMReq Message, the PMSP SHOULD use the most recent state of the requested service to reply to the PMReq (either with a PMAccept or a PMDeny).

The above rules mean that the PMSP will always attempt to communicate the most current state information about the requested service. However, because the Port Mapper protocol is mapped onto UDP/IP, it is possible that messages can be re-ordered upon reception. Thus when the PMSP receives a duplicate PMReq Message, and the PMSP changes its reply from a PMAccept to a PMDeny or a PMDeny to a PMAccept, the reply can be received out-of-order. To keep the Port Mapper protocol simple, rather than add a sequence number to detect this sequence of events, the approach is to keep the protocol response deterministic and require the PM Client to use the first reply it gets from the PMSP (see section 7.5.1 PM Client Behavior on page 43).

If the PMSP receives a PMReq for a transaction that it has already sent back a PMAccept, but the AssocHandle does not match the prior request, the PMSP MUST discard and cleanup the state associated with the prior request and process the new PMReq normally.

Note that if a duplicate message arrives after the PMSP state for the request has been deleted, the PMSP will view it as a new request, and generate a reply. If the prior reply was acted upon by the Connecting Peer, then the latest reply would presumably have no matching context and be discarded by the PM Client. The PMSP state will be recovered in one of the following ways:

* after the timeout interval (waiting for PMAck or TCP SYNLLP connection setup request to the SDP Address), or

* if the client issues a new request, with a new AssocHandle, but for the same four-tuple (ApPort, CpPort, ApIPAddr, CpIPAddr) the PMSP will discard the prior state and answer the current request.

The Hello and HelloAck Messages defined in sections 6.2.1 Hello Message (HH) on page 17 and 6.2.2 HelloAck Message (HAH)on page 19 are used to establish the SDP connection.

SDP connection teardown uses the LLP connection teardown mechanisms, plus two additional SDP Messages types to emulate TCP connection semantics for abortive and graceful connection teardown.

8 Connection Setup

8.1.1 iWARP Connection Setup

SDP communications MUST use the iWARP protocol suite for all SDP and RDMA Messages, except for the Hello Message, which MUST be sent in Streaming Mode.

Each socket corresponds to a single iWARP Connection Context.

SDP connection setup MUST include the following numbered steps in order, after the LLP connection is setup:

1. The Connecting Peer prepares to send a Hello Message.

    a. The Connecting Peer MUST configure the number of local receive Private Buffers that will be posted and the size of those receive Private Buffers. See section 10 Private Buffer Management on page 73 for constraints on Private Buffers.

    b. The Connecting Peer MUST create a Connection Context, initialize any required attributes and associate that Connection Context to the LLP Stream if it has not done so already.

    c. The Connecting Peer MUST post the number of receive Private Buffers that it advertises in the BSDH Bufs field at this time or before sending the Hello Message. See section 10 Private Buffer Management on page 73.

    d. The Connecting Peer MUST set the values of the LocIRD and LocORD fields in the Hello Message to greater than or equal to one and less than or equal to the local IRD and local ORD values, respectively, that the connection is able to support.

    e. The Connecting Peer MUST be able to receive an incoming iWARP Message immediately after sending the Hello Message. Note that the transition to iWARP mode and sending the Hello Message must appear to be performed atomically. Failure to do so may result in race conditions.

2. The Connecting Peer MUST send a Hello Message to the Accepting Peer in Streaming Mode. The Connecting Peer MUST NOT send any iWARP Messages until a HelloAck Message is received from the Accepting Peer. See section 6.2.1 Hello Message (HH) on page 17 and section 10.3 Initialization of Send Credit on page 74 for additional information on filling in the SDP parameters.

3. The Accepting Peer, upon receipt of the Hello Message, determines whether to accept or terminate the connection attempt as follows:

   a. The Accepting Peer MUST terminate the connection attempt when any of the following conditions occur:

      * The Accepting Peer does not support the Major Protocol Version Number contained in the Hello Message.

      * The MaxAdverts field in the Hello Message is equal to zero.

      * The LocIRD or LocORD field in the Hello Message is equal to zero.

   b. If none of the above conditions occur, the Accepting Peer may accept the connection.

      * The Accepting Peer SHOULD NOT reject the connection request based solely on a mismatch of the Minor Protocol Version Number.

      * The Accepting Peer MUST use the protocol specified by the minimum of the locally supported Minor Protocol Version Number and the value of the MinV field received in the Hello Message.

4. If the connection is accepted, the Accepting Peer MUST send a HelloAck Message back to the Connecting Peer. See section 6.2.2 HelloAck Message (HAH) on page 19 for additional information. The steps prior to sending a HelloAck Message are as follows:

   a. The Accepting Peer MUST create a Connection Context, initialize required attributes and associate that Connection Context with the LLP connection if it has not done so already.

   b. The Accepting Peer MUST post the number of its receive Private Buffers that it advertises in the BSDH Bufs field before sending the HelloAck Message. See section 10 Private Buffer Management on page 73.

   c. The Accepting Peer MUST set the values of LocIRD and LocORD fields in the HelloAck Message to greater than or equal to one and less than or equal to the local IRD and local ORD values, respectively, that the connection is able to support.

d. If the LocORD in the incoming Hello Message is less than the Accepting Peer's local LocIRD in the associated Connection Context, the Accepting Peer MAY reduce its local IRD to a value that is greater than or equal to the LocORD contained in the Hello Message.

e. If the LocIRD in the incoming Hello Message is less than the Accepting Peer's local ORD in the associated Connection Context, the Accepting Peer MUST modify that local ORD to a value less than or equal to the LocIRD contained in the Hello Message.

5. Send the HelloAck Message using the iWARP Send with SE Message.

6. The Accepting Peer MUST set the Flow Control Mode to Combined Mode and MAY immediately commence data transfer.

7. The Connecting Peer receives the HelloAck Message.

   a. The Connecting Peer MUST terminate the connection attempt when any of the following conditions occur:

      * The Major Protocol Version Number sent in the Hello Message does not match the Major Protocol Version Number in the HelloAck Message. Note that this should not happen because the Accepting Peer should have terminated the SDP connection due to a MajV mismatch.

      * The MaxAdverts field in the HelloAck Message is equal to zero.

      * The LocORD or LocIRD field in the HelloAck Message is equal to zero.

   b. If the above conditions do not cause a connection termination:

      * The Connecting Peer SHOULD NOT terminate the connection request based solely on a mismatch of the Minor Protocol Version Number sent in the Hello Message and the MinV value received in the HelloAck Message.

      * The Connecting Peer MUST use the protocol specified by the minimum of the Minor Protocol Version Number sent in the Hello Message and the value received in the HelloAck Message.

   c. If the LocORD in the incoming HelloAck Message is less than the Connecting Peer's local IRD in the associated Connection Context, the Connecting Peer MAY reduce its local IRD to a

value that is greater than or equal to the LocORD contained in the HelloAck Message.

d. If the LocIRD in the incoming HelloAck Message is less than the Connecting Peer's local ORD in the associated Connection Context, the Connecting Peer MUST modify its local ORD to a value less than or equal to the LocIRD contained in the HelloAck Message.

e. The Connecting Peer MUST set the Flow Control Mode to Combined Mode and MAY immediately commence data transfer. All SDP Messages from this point on are sent using iWARP.

8.1.2 Aborting Connection Setup

If the LLP connection is torn down during connection setup, the implementation MUST abort the SDP connection setup.

An SDP implementation should clean up any resources associated with an aborted connection.

8.2 Connection Teardown

SDP emulates TCP connection teardown functionality. TCP provides two ways to close a connection - a graceful close, where any data that has been posted by the ULP to the transport is transferred before the connection is torn down, and abortive close, where the connection is immediately torn down.

8.2.1 Graceful Close

TCP's graceful close (also known as graceful disconnect or half-closed connections) is an agreement between the transport and ULP that:

* Before the connection is terminated, all data accepted for transmission by the transport before the close occurred is guaranteed to be sent out (under reasonable limitations) and reliably acknowledged.

* Data reception can continue normally until the Remote Peer performs a close.

Sockets Direct Protocol provides the same behavior over iWARP.

The Local Peer SHOULD NOT close the LLP connection at the time of the ULP's call to gracefully close the half-connection. The Local Peer MUST reject any send data posted by the ULP after the ULP close call occurred.

The Local Peer SHOULD continue to receive ULP data through any of the SDP Data Transfer Mechanisms until the Remote Peer gracefully closes the connection, or the connection is abortively closed (see section 8.2.2 Abortive Close on page 52 for the abortive close protocol).

The Local Peer MUST also perform the following operations in the order specified:

1. The Local Peer MUST complete the transmission of all outbound data posted by the ULP before the ULP requested the graceful close. This means that all Bcopy transfers, Write Zcopy transfers, Read Zcopy transfers, and Transaction transfers from this Data Source have been completed (see section 9 Data Transfer Mechanisms on page 54). Completions may be successful or unsuccessful (e.g., the LLP connection was torn down). Unsuccessful completions MUST cause the Local Peer to perform an abortive close (see section 8.2.2 Abortive Close on page 52).

2. The Local Peer MUST send a DisConn Message to the Remote Peer. This informs the Remote Peer that the connection is being terminated gracefully, allowing the Remote Peer to inform the ULP of this fact, as appropriate. If the transmission of the DisConn Message completed with an error, the connection tear down was abortive. The DisConn Message provides similar semantics to a TCP segment with the FIN bit set. Because the LLP is required to provide reliable in-order delivery, no ULP data will be received by the Remote Peer after the Remote Peer receives the DisConn Message. The Local Peer MUST continue to receive SDP Messages to enable ULP data transfer on the opposite half-connection.

3. The Local Peer MUST wait for one of the following events:

    * The Local Peer receives a DisConn Message. The Remote Peer gracefully closed the opposite half-connection, unless an AbortConn Message is received before the connection is terminated.

    * The LLP connection is torn down (this is due to an abortive close).

    * The local ULP abortively closes the connection (see section 8.2.2 Abortive Close on page 52).

    * When no forward progress is being made, the connection MAY be abortively closed (see section 8.2.2 Abortive Close on page 52).

4. When the SDP implementation is informed that the LLP connection was torn down, the Local Peer MUST determine whether a DisConn Message, or a DisConn Message followed by an AbortConn Message, was received. If a DisConn Message was received without an AbortConn, the graceful close was successfully completed. If a DisConn Message was not received, or a DisConn Message and an AbortConn Message were received, then the close was abortive (see section 8.2.2 Abortive Close on page 52). In either case, the Local Peer MUST complete all ULP receive buffers with information about how much of the buffer was filled.

5. The Local Peer MUST also clean up all iWARP resources associated with the connection (Connection Context, buffers, etc.).

The Remote Peer MUST perform the following operations for a graceful close:

1. Upon receipt of a DisConn Message, the Remote Peer MUST consider all its outstanding SinkAvail advertisements as canceled, complete all ULP receive buffers, and wait for the ULP to close the connection. The Remote Peer MUST continue to allow normal ULP send data transfer, but MUST complete any new ULP receive buffers and inform the ULP (as appropriate) that the receive half-connection has been gracefully closed.

2. If the ULP issues an abortive close, the Remote Peer MUST use the abortive close protocol (see section 8.2.2 Abortive Close on page 52). If the ULP issues a graceful close, the Remote Peer MUST complete the transmission of all send ULP data that was posted before the ULP posted the graceful close. Completions may be successful or unsuccessful (e.g., the LLP connection was torn down). The Remote Peer MUST reject any send data posted by the ULP after the ULP close call occurred.

3. The Remote Peer MUST send a DisConn Message to the Local Peer. If the DisConn Message completed without error and no AbortConn Message was received, then the graceful teardown was successful. If the DisConn Message completed with an error, or an AbortConn was received, the connection teardown was abortive.

4. The Remote Peer MUST use the LLP connection teardown protocol to complete the teardown. The Remote Peer MUST also close and clean up all iWARP resources associated with the connection (Connection Context, buffers, etc.).

8.2.2 Abortive Close

If the ULP specifies an abortive disconnect or an abortive disconnect is required for some other reason, an SDP implementation MUST comply with the following rules:

* If the LLP connection is still valid and a DisConn Message was previously sent, then an implementation MUST send an AbortConn Message and process its completion before terminating the LLP connection.

* If the LLP connection is not valid, or if the LLP connection is valid and no DisConn Message was previously sent, an implementation MUST NOT attempt to send additional SDP Messages and MUST immediately terminate the LLP connection.

An SDP implementation MUST consider the connection abortively torn down if the LLP connection is torn down without receiving a DisConn Message, or if both an AbortConn and a DisConn Message were received. An implementation MUST discard any unsent ULP data.

If an SDP protocol violation occurs, an implementation SHOULD abortively close the connection. A protocol violation includes but is not limited to LLP errors, invalid SDP Messages, or incorrectly formatted SDP Messages.

Certain ULP behaviors can lead to a situation under which the ULP initiates graceful teardown in the send direction (causing the DisConn Message to be sent), and then some error occurs that requires the connection to be abortively closed. The AbortConn Message is used for this purpose. The AbortConn Message is sent if the DisConn Message has already been sent, but the LLP connection has not been terminated yet, and some error condition arises that calls for abortive teardown of the socket under TCP semantics. Sending out the AbortConn Message informs the Remote Peer to ignore the earlier DisConn Message and inform the ULP (as appropriate) that the connection was closed abortively. In this case, the AbortConn Message provides similar semantics to TCP sending a segment with the RST bit set after it has already sent a segment with the FIN bit set.

Once the AbortConn Message completion event occurs, an SDP implementation MUST use the normal LLP connection teardown protocol to complete the teardown.

9 Data Transfer Mechanisms

SDP employs four Data Transfer Mechanisms:

* Bcopy - transfer of ULP data from send buffers into receive Private Buffers.

* Read Zcopy - transfer of ULP data through RDMA Reads, preferably directly from ULP Buffers into ULP Buffers.

* Write Zcopy - transfer of ULP data through RDMA Writes, preferably directly from ULP Buffers into ULP Buffers.

* Transaction - an optimized ULP data transfer model for transactions. It piggy-backs ULP data transfer using Private Buffers on top of the Write Zcopy mechanism used to transfer ULP data on the opposite half-connection.

The policy that controls when to use the Bcopy Data Transfer Mechanisms versus a Zcopy Data Transfer Mechanism is outside the scope of this specification. An implementation dependent parameter defined as the Bcopy Threshold is used to abstractly define the results of the policy decision. No constraints are placed on the Data Source Bcopy Threshold values, and the value of the Data Source Bcopy Threshold may be static or dynamic. The Data Sink Bcopy Threshold has a single constraint: it MUST be greater than or equal to the size of the receive Private Buffers. Its value MAY also be static or dynamic.

Note that some socket implementations do not provide deterministic results if overlapping receive buffers are posted.

An SDP implementation MUST support the Bcopy Data Transfer Mechanism, both as a Data Source and as a Data Sink.

It is strongly RECOMMENDED that an SDP implementation support the ability to initiate all of the Data Transfer Mechanisms.

It is strongly RECOMMENDED that an SDP implementation support the ability to carry out all of the Data Transfer Mechanism requests.

If an SDP implementation does not support carrying out a received request for a given optional Data Transfer Mechanism, the implementation MUST still be able to parse the received request, and to force the use of the Bcopy Data Transfer Mechanism by sending a SendSm Message.

For example, if an SDP implementation does not support carrying out the Read Zcopy Data Transfer Mechanism request, when a SrcAvail Message is received, the implementation must be able to respond with

a SendSm Message to force the Data Source to use the Bcopy Data Transfer Mechanism.

9.1 Bcopy

SDP maintains a small set of receive Private Buffers for receiving data that the Data Source transfers using iWARP Sends. Each connection has a separate pool of receive Private Buffers.

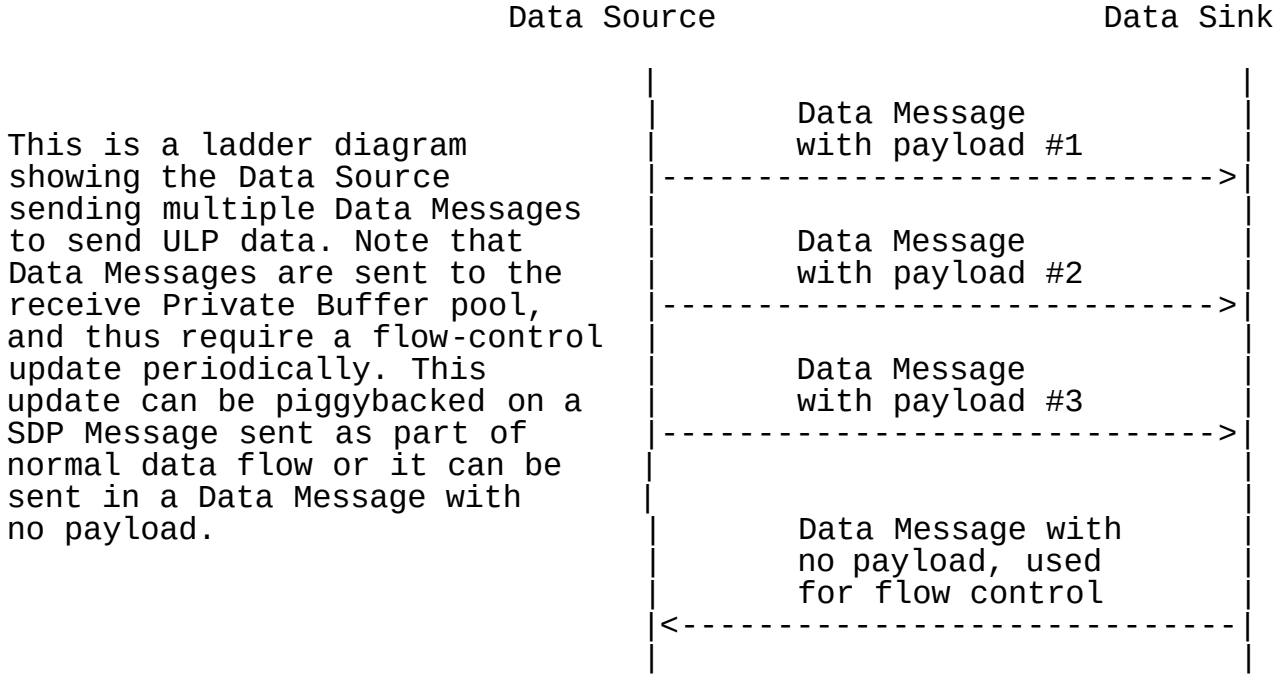

Figure 20 Ladder Diagram for Bcopy Mechanism

Each peer chooses its own sizes of send and receive Private Buffers and informs the other peer of the size of the receive Private Buffer during connection setup.

The Data Source MUST limit the amount of ULP data sent in an SDP Message (specifically a Data, SinkAvail, or SrcAvail Message) to ensure the ULP data plus SDP header(s) fits within the receive Private Buffer size advertised by the Data Sink.

SDP Message transfer is flow controlled as described in section 10 Private Buffer Management on page 73.

For the Data Source, data may be copied from the ULP's buffer to the payload sections of one or more of the send buffers, or the ULP Buffer may be referenced directly by the send work request. In the

header, the MID is set to type Data Message and the SDP Message size is set to the ULP payload size plus the size of the header.

The Data Sink receives the SDP Message in its posted receive Private Buffers. When a ULP receive buffer is completed is outside the scope of this specification.

9.2 Read Zcopy

This mechanism MUST transfer data through the following sequence of operations:

1. The Data Source sends a SrcAvail Message when a send ULP Buffer that the Source deems suitable has been posted (there are no protocol restrictions on the Data Source use of the Bcopy mechanism versus Read Zcopy mechanism for transfer of a specific ULP Buffer). For example, a SrcAvail Message may be sent if the ULP Buffer is larger than the Bcopy Threshold. If the Source chooses to advertise a ULP Buffer in a SrcAvail Message, the ULP Buffer is referred to as an RDMA Buffer (the RDMA Buffer may be a copy of the ULP Buffer).

   In Combined Mode the SrcAvail Message payload MUST contain at least one byte of ULP payload. However, in Pipelined Mode the SrcAvail Message MUST NOT contain ULP payload (see sections 11.2 Combined Mode on page 80 and 11.3 Pipelined Mode on page 81).

   The SrcAH Len, VA, and STag fields MUST describe the entire Data Source RDMA Buffer and MUST reference the same Data Source RDMA Buffer, regardless of whether a copy of the initial portion of the RDMA Buffer is included in the SrcAvail Message payload.

   After receiving a SrcAvail Message, the Data Sink MAY send a SendSm Message when ULP receive buffer(s) are not suitable for Read Zcopy.

2. The Data Sink receives the SrcAvail Message and waits for the ULP to post a receive buffer to SDP. If the Data Sink chooses to complete the data transfer:

   a. If the receive ULP Buffer is viewed as unsuitable for Read Zcopy, the Data Sink MUST send a SendSm Message (see section 9.5.2 Mechanisms for Forcing Bcopy on page 66).

   b. If the receive ULP Buffer is viewed as suitable for Read Zcopy, the Data Sink MUST use the Read Zcopy Data Transfer Mechanism. An implementation should use the ULP Buffer as the RDMA Buffer. An implementation may choose to create an intermediate buffer as the RDMA Buffer, and then copy the data into the ULP Buffer. If the initial portion of the send

RDMA Buffer is present in the SrcAvail advertisement, the Data Sink moves the data into the RDMA Buffer through one of the following mechanisms:

* Copy some or all of the ULP payload of the SrcAvail Message to the receive RDMA Buffer, and then perform one or more RDMA Read(s) to retrieve the rest of the data, offsetting the initial RDMA Read transfer by the number of bytes that were copied out of the SrcAvail Message ULP payload.

* Avoid the ULP payload copy and start the initial RDMA Read at the start of the send RDMA Buffer. Additional RDMA Reads may be used to transfer the rest of the buffer.

After the RDMA Read(s) complete(s), the Data Sink MUST send an RdmaRdCompl Message to the Data Source, unless the operation was canceled (see section 9.5.4 SrcAvail Revocation on page 69). The Data Sink MUST wait for completion of the RDMA Read before sending the RdmaRdCompl.

The RdmaRdCompl header MUST contain the size (in bytes) of ULP data transferred through the RDMA Read(s), excluding any portion of the ULP data that was originally transferred through the SrcAvail Message. The RdmaRdCompl Message MUST refer to data made available through a single SrcAvail advertisement.

A Data Sink MUST only send an RdmaRdCompl Message associated with the oldest incomplete SrcAvail Message.

The RdmaRdCompl Len does not include the portion of the data, if any, transferred within the SrcAvail Message as ULP payload. The size MAY be less than the size of the Data Source RDMA Buffer advertised in the SrcAvail Message minus the size of ULP data payload included in the SrcAvail Message. An implementation MAY loop performing a series of RDMA Read operations followed by RdmaRdCompl Messages to transfer the send RDMA Buffer contents.

It is expected (but not required) that protocol implementations would typically RDMA Read all the ULP data and then send a single RdmaRdCompl Message to inform the Data Source that the SrcAvail Message has been Processed. The facility to specify data transfer size less than the RDMA Buffer size advertised in the SrcAvail Message enables various transfer scenarios. For example, a protocol implementation could RDMA Read part of the data and then send a RdmaRdCompl Message followed by a SendSm Message to retrieve the rest of the data using the Bcopy mechanism. Only one SendSm Message can be used to complete data transfer for a given SrcAvail advertisement. See 3) below.

Note that if the Flow Control Mode from the Data Source to the Data Sink is Combined Mode, the Data Sink can set the REQ_PIPE bit in the BSDH Flags field of the RdmaRdCompl Message if it wishes to transition to Pipelined Mode (see section 6.1 Base Sockets Direct Header (BSDH)on page 13).

3. Upon receiving the RdmaRdCompl Message, the Data Source MUST compare the RRCH Len field against the length of the oldest, incomplete SrcAvail advertisement. If the send RDMA Buffer has not been completely transferred, the Data Source MUST wait until the SrcAvail Message has been Processed by an ensuing RdmaRdCompl Message or a SendSm Message. Once the send RDMA Buffer is completed, this may map to completion of a send ULP Buffer, as appropriate.

   Data advertised by SrcAvail MUST remain available for Read Zcopy by the Data Sink until its consumption has been acknowledged with RdmaRdCompl Message(s), a SendSm Message is received from the Data Sink, the SrcAvail Message has been overridden because of a SinkAvail Message (see section 11.3 Pipelined Mode on page 81), or the Data Source has Processed a SrcAvailCancel sequence (see section 9.5.4 SrcAvail Revocation on page 69).

   Note the portions of the RDMA Read buffer that have been completed by an RdmaRdCompl are no longer required to be available for RDMA Read.

   The Data Sink SHOULD send the final RdmaRdCompl Message for an RDMA Read Buffer with the Send with SE and Invalidate iWARP Message to invalidate the STag associated with that RDMA Read Buffer. If the Data Sink does not use the Send with SE and Invalidate iWARP Message for RdmaRdCompl, it MUST use the Send with SE iWARP Message.

   If the Data Sink used the iWARP remote invalidate feature, then the Data Source MUST verify that the invalidated STag was the same STag that was sent in the original SrcAvail Message - and if it was not the same STag, the Data Source MUST view this as a protocol violation. If the Data Sink did not use the remote invalidate feature, the Data Source should locally invalidate the advertised STag (If the Data Source does not invalidate the advertised STag, there are security implications. See [RDMAP-SECURITY] for additional details).

   If a SendSm Message is received at the Data Source, the Data Source MUST match the SendSm Message with the oldest, incomplete SrcAvail advertisement. The Data Source MUST view this SrcAvail as Processed and MUST send the remaining ULP data using Data Messages (see section 9.1 Bcopy on page 55). The Data Sink MUST consume this data before sending an RdmaRdCompl for other

SrcAvail Messages (this condition can only occur in Pipelined Mode - see section 11.3 Pipelined Mode on page 81).

Upon receiving a SendSm Message, the Data Source MUST complete the oldest incomplete SrcAvail advertised buffer using Data Message(s). After sending a SendSm Message, the Data Sink MUST wait until it has received all of the data that was advertised in the corresponding SrcAvail before sending any RdmaRdCompl Message, even for another advertised buffer.

Note that the Data Sink calculates the number of remaining bytes expected through Data Messages by subtracting from the SrcAH length field the sum of the number of bytes that the Data Sink has acknowledged with RdmaRdCompl Message(s) plus the amount of ULP payload included in the SrcAvail Message (if any).

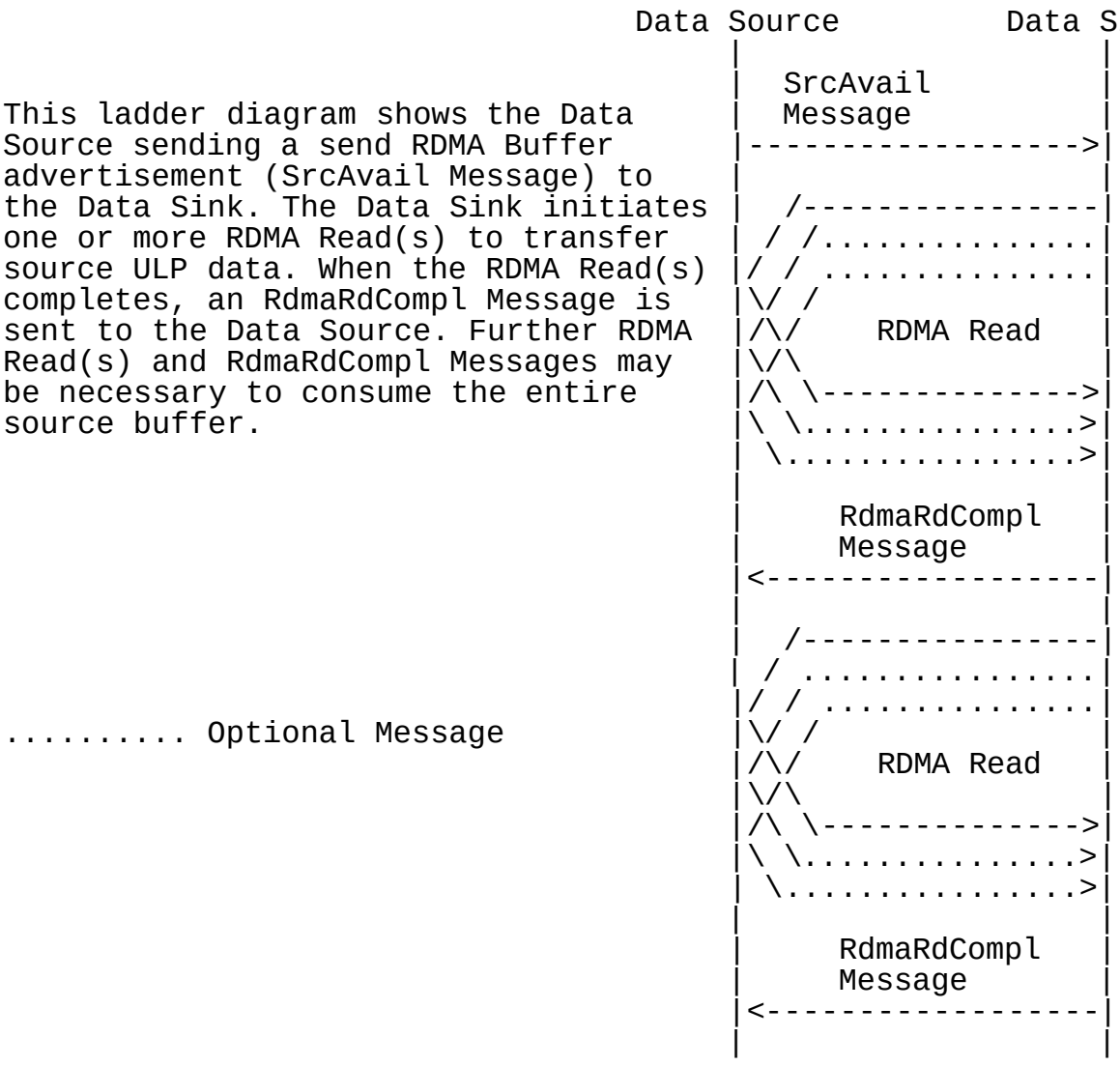

Figure 21 Ladder Diagram for Read Zcopy Mechanism

When the Data Sink sends multiple RdmaRdCompl Messages for a single SrcAvail advertisement, the Data Sink MUST set the RdmaRdCompl Message length field to the number of bytes transferred since the last RdmaRdCompl was sent for this SrcAvail advertisement.

It is possible to create a deadlock if, at the same time, both ULP peers post send data suitable for Read Zcopy and both ULPs wait for the associated send to complete before posting a receive. A SrcAvail Message could be sent by each SDP peer, but no ULP receive buffer would be posted. This deadlock is possible when all of the following are true:

* A SrcAvail is received; and

* No ULP receive buffer is posted; and

* The local Data Source has a SrcAvail outstanding.

When these conditions are true, a deadlock can be avoided in a variety of ways:

* The Data Sink could send a SendSm Message to force the use of the Bcopy Data Transfer Mechanism.

* The Data Source could send a SrcAvailCancel Message and then complete the ULP write using the Bcopy Data Transfer Mechanism.

* The Data Sink could complete the Read Zcopy using a local buffer, holding that data until the ULP posts a receive.

Regardless of the method used, it is strongly RECOMMENDED that SDP implementations detect and recover from this deadlock situation.

9.3 Write Zcopy

This mechanism MUST transfer data through the following sequence of operations:

1. The Data Sink sends a SinkAvail Message to the Data Source when a suitable receive ULP Buffer is posted (note that the ULP Buffer MUST be larger than the size of the local receive Private Buffers - see section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65). If Write Zcopy is chosen, the ULP Buffer is referred to as a receive RDMA Buffer (the receive RDMA Buffer may actually be a Private Buffer from where receive data is copied to the ULP Buffer - this is implementation-dependent).

   The SinkAH Len, VA, and STag fields MUST describe the entire Data Sink RDMA Buffer and MUST reference the same Data Sink RDMA Buffer. The Data Sink MUST set a value in the NonDiscards field as specified in section 6.3.3.4 NonDiscards - 32 bits on page 24.

2. The Data Source receives the SinkAvail Message and waits for the ULP to post a send buffer. If the Data Source determines the buffer is suitable for Write Zcopy, it MUST use one or more RDMA Writes to transfer ULP data to the Data Sink. If the Data Source determines the buffer is unsuitable for Write Zcopy, it MUST use the protocol described under section 9.5.2.2 Data Source Forcing Bcopy on page 67.

   After the RDMA Write(s) complete, the Data Source MUST send a single RdmaWrCompl Message to the Data Sink, unless the

operation was canceled. The Data Source SHOULD send the RdmaWrCompl Message using Send with SE and Invalidate to invalidate the STag associated with the completed RDMA Buffers at the Data Sink. If the Data Source does not use the Send with SE and Invalidate iWARP Message for RdmaWrCompl, it MUST use the Send with SE iWARP Message.

If the Data Source used the iWARP remote invalidate feature, then the Data Sink MUST verify that the invalidated STag was the same STag that was sent in the original SinkAvail Message - and if it was not the same STag, the Data Sink MUST view this as a protocol violation. If the Data Source did not use the remote invalidate feature, the Data Sink should locally invalidate the advertised STag (If the Data Sink does not invalidate the advertised STag, there are security implications. See [RDMAP-SECURITY] for additional details).

The RdmaWrCompl header MUST contain the size (in bytes) of data transferred through the RDMA Write(s).

3. Upon receiving the RdmaWrCompl Message, the Data Sink MUST match the RdmaWrCompl Message to the oldest incomplete SinkAvail advertisement and MUST consider the SinkAvail advertisement Processed. Once the receive RDMA Buffer is Processed, this may map to completion of a receive ULP Buffer, as appropriate. If the RdmaWrCompl Message did not include a remote invalidate, the Data Sink may invalidate the advertised STag.

A Data Sink RDMA Buffer advertised by SinkAvail MUST remain available for Write Zcopy from the Data Source until it has been acknowledged with an RdmaWrCompl Message, the SinkAvail was canceled due to a Data Message (see section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65), or the advertisement has been revoked, (and the revoke request has been Processed - see section 9.5.5 SinkAvail Revocation on page 70).

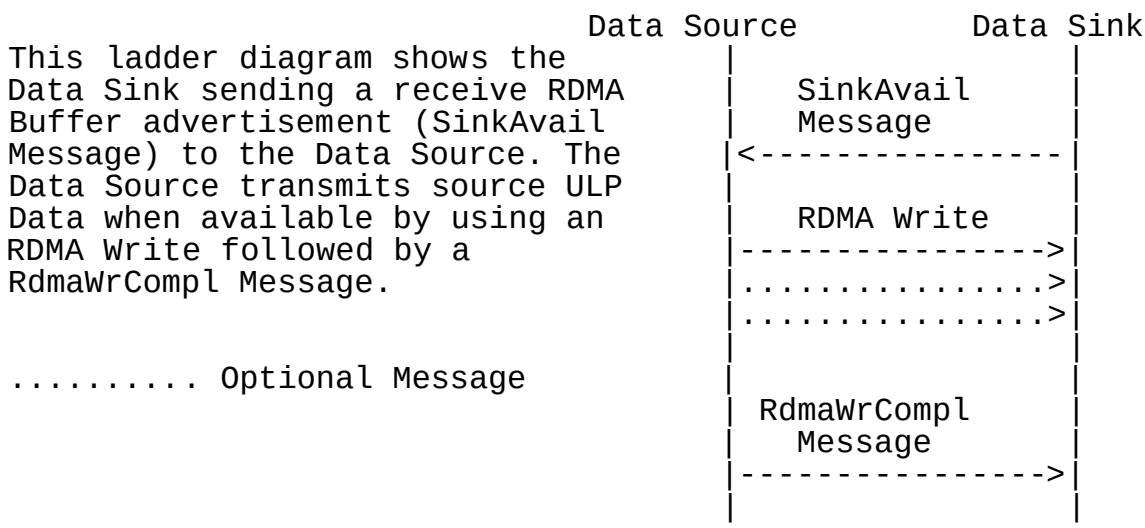

Figure 22 Ladder Diagram for Write Zcopy Mechanism

Some socket implementations support an option to ensure that receive ULP Buffers are completely filled before they are returned to the ULP. This is typically implemented as a flag called MSG_WAITALL that is specified when a receive ULP Buffer is posted. If the MSG_WAITALL socket option is supported by the Data Sink implementation, the Data Sink MUST disable Write Zcopy for ULP Buffers that have MSG_WAITALL set by not sending SinkAvail advertisements to the Data Source. Enabling Write Zcopy for buffers with MSG_WAITALL breaks the ULP Buffer accounting algorithm that addresses crossing SinkAvail and Data Messages (see section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65). If the ULP Buffer accounting algorithm is used, then the Data Sink must partially complete the ULP Buffer when a Data Message and SinkAvail Message cross. Disabling SinkAvail prevents this condition, thus enabling the receive ULP Buffer to be completely filled in all scenarios. Note that the setting of MSG_WAITALL flag for a receive buffer does not restrict the use of Read Zcopy or Bcopy Data Transfer Mechanisms.

9.4 Transaction Mechanism

If the ULP is transaction oriented, typically one peer is sending short command messages and medium to long reply messages are expected. It is possible to optimize this transfer model by collapsing the SinkAvail advertisement for the reply's receive RDMA Buffer with the Data Message for the command. This enables Zero-copy receives on potentially smaller replies as well as reducing control traffic. Note that the SinkAvail Message is used to transfer ULP payload that is being sent in the opposite direction of the Data Message. In order for that SinkAvail to be generated, the Flow Control Mode needs to be Pipelined Mode (see section 11.3 Pipelined Mode on page 81). Furthermore, the receive RDMA Buffer for the

SinkAvail advertisement needs to be larger than the local receive Private Buffer size (see section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65).

If an end point receives a SinkAvail Message with ULP payload, the Data Source SHOULD use RDMA Writes to fill the advertised RDMA Buffer unless prior ULP payload-carrying SDP Messages effectively canceled this SinkAvail advertisement (see section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65).

Figure 23 Ladder Diagram of Transaction Mechanism on page 64 shows the collapsing of the Data Message into the SinkAvail advertisement. ULP peer A is communicating with ULP peer B, with a traffic pattern that appears as though it is transactional. Peer A is repetitively sending peer B a single small ULP message (this is the command) and then immediately posting a receive ULP Buffer (this is the reply). Without the piggyback mechanism, two SDP Messages could potentially be generated by peer A.

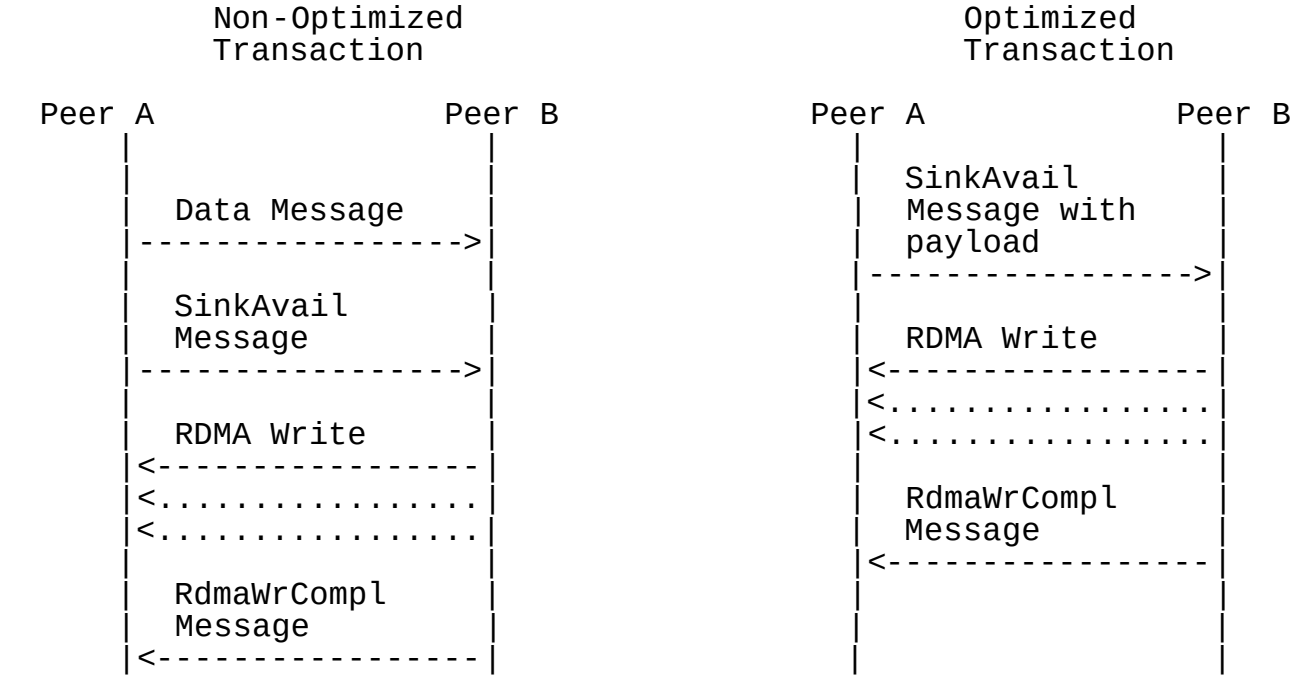

Figure 23 Ladder Diagram of Transaction Mechanism

ULP peer B is consistently waiting for reception of a command before posting a send ULP Buffer for the reply. If the Transaction

mechanism was not available and the SinkAvail advertisement was not received before the reply was posted by the ULP, the reply Data Source would have to choose whether to wait for the SinkAvail advertisement, generate a SrcAvail advertisement, or transfer the ULP data using the Bcopy mechanism. With the Transaction mechanism, the logic is straightforward. The reply Data Source MAY use the Write Zcopy mechanism to transfer the reply. If the reply ULP Buffer is not suitable for RDMA Write, the Data Source MAY send the reply data using the Bcopy mechanism. All Pipelined Mode rules apply (see section 11.3 Pipelined Mode on page 81).

9.5 Miscellaneous Data Transfer Issues

9.5.1 Detecting Stale SinkAvail Advertisements

SDP allows the Data Source to send ULP data through SDP Messages. This creates an issue in Pipelined Mode because a SinkAvail advertisement could cross an SDP Message containing ULP data that is destined for the same receive ULP Buffer that was advertised in the SinkAvail Message. The receive ULP Buffer could be at least partially satisfied through the Data Message. This effectively requires the Data Source to view the receive RDMA Buffer advertisement as outdated or stale.

The Data Sink MUST only send a SinkAvail advertisement if the RDMA Buffer is larger than the local receive Private Buffer size.

This also means that the Data Sink MUST set its Bcopy Threshold to be larger than or equal to the receive Private Buffer size.

SDP MUST use the following algorithm to enable the Data Source to detect and recover from stale SinkAvail advertisements:

1. If a ULP receive RDMA Buffer R has been advertised through a SinkAvail Message and one or more SDP Messages with ULP payload (Data or SinkAvail Message with ULP payload) arrives at the Data Sink, then the Data Sink MUST copy the ULP payload of exactly one SDP Message into R and MUST return R to the ULP. In other words, R will not consume the ULP payload of more than one SDP Message.

   In Pipelined Mode, the Data Sink MUST use the ULP payload of only one SDP Message (Data Message or SinkAvail) to complete any one RDMA Buffer advertised through SinkAvail.

   If a receive ULP Buffer has not been advertised through a SinkAvail Message, that buffer MAY consume the ULP payload of more than one SDP Message.

2. The Data Source MUST keep a 32-bit counter, PotentialNonDiscards, which tracks the number of SDP Messages carrying ULP payload that the Data Source has sent that might not cause a SinkAvail Message to be discarded. The PotentialNonDiscards counter MUST be initialized to zero and MUST wrap to zero after reaching 0xFFFFFFFF.

3. The Data Source, upon sending an SDP Message carrying ULP payload, MUST increment PotentialNonDiscards by one.

4. At the Data Source:

   The Data Source MUST execute the following pseudo-code for each received SinkAvail advertisement before initiating a Write Zcopy data transfer using the advertised buffer. If ULP payload is present in the SinkAvail, the Data Source MUST process the ULP payload normally regardless of whether the SinkAvail was discarded.

```
If (SinkAvail.NonDiscards != PotentialNonDiscards)
  Discard(SinkAvail)
  PotentialNonDiscards--

Else
  Process SinkAvail normally
```

   For example, if PotentialNonDiscards=2 and the Data Source has three SinkAvail advertisements, all with NonDiscards=0, then the first two advertisements are discarded as stale and RDMA is initiated on the third advertisement.

Note that this algorithm can cause receive ULP Buffers to be partially filled when completed. If a ULP Buffer is required to be completely filled, an SDP implementation should not advertise the ULP Buffer with a SinkAvail Message. See section 9.3 Write Zcopy on page 61 for additional details.

Detecting stale SinkAvail advertisements is one mechanism that causes a SinkAvail advertisement to be discarded. Section 12.4 Transition From Pipelined Mode to Combined Mode on page 86 defines a different circumstance when a SinkAvail Message must be discarded.

9.5.2 Mechanisms for Forcing Bcopy

9.5.2.1 Data Sink Forcing Bcopy

While in Combined or Pipelined Modes, if the Data Sink determines that its buffer is unsuitable for use with Read Zcopy from an RDMA Buffer advertised by the Data Source, the Data Sink can use the SendSm Message to force the Data Source to send data through the

Bcopy mechanism (i.e., through Data Messages). The Data Sink MAY send the SendSm Message after receiving a SrcAvail Message.

Upon receiving the SendSm Message, the Data Source MUST send all remaining ULP data (advertised in the associated SrcAvail Message) using Data Messages.

The Data Sink MUST process SrcAvail Messages in MSeq order as described in section 10.1 SDP Message Ordering on page 73. For each SrcAvail Message received, the Data Sink MUST either proceed with the appropriate RDMA Data Transfer Mechanism (Read Zcopy if a SinkAvail advertisement did not cross, or wait for an RdmaWrCompl if a crossing occurred - see section 11.3 Pipelined Mode on page 81) or it MUST respond with a SendSm Message.

The Data Source MUST respond to the SendSm request by matching it to the oldest incomplete SrcAvail advertisement and then sending the remaining ULP data for the SrcAvail advertisement (i.e., that has not been Processed by RdmaRdCompl Message(s) from the Data Sink or already sent as ULP payload in the SrcAvail Message) through the Bcopy mechanism.

Implementation note: in some cases the Data Source and Data Sink could have different Bcopy Threshold values. When the Data Source advertises an RDMA Buffer whose size is greater than the Data Source’s Bcopy Threshold but less than the Data Sink’s Bcopy Threshold, the Data Sink could choose to force a Bcopy. However, since the Data Source has already invested in setting up a Read Zcopy data transfer, the Data Sink should give special consideration to cooperating with the Data Source’s attempt to use Read Zcopy. Note that the Data Sink is free to force a Bcopy if it determines that for any reason its buffer is unsuitable for Read Zcopy.

9.5.2.2 Data Source Forcing Bcopy

While in Pipelined Mode, if the Data Source determines that its buffer is unsuitable for use with Write Zcopy to an RDMA Buffer advertised by the Data Sink, the Data Source MAY choose to not use the Data Sink’s RDMA Buffer advertisement, and instead use Data Messages to send data using the Bcopy Data Transfer Mechanism. In this case, the Data Source and Data Sink MUST follow the protocol described in section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65.

Implementation note: in some cases the Data Source and Data Sink could have different Bcopy Threshold values. When the Data Sink advertises an RDMA Buffer whose size is greater than the Data Sink’s Bcopy Threshold but less than the Data Source’s Bcopy Threshold, the Data Source could choose to force a Bcopy. However, since the Data Sink has already invested in setting up a Write Zcopy data transfer,

the Data Source should give special consideration to cooperating with the Data Sink's attempt to use Write Zcopy. Note that the Data Source is free to force a Bcopy if it determines that for any reason its buffer is unsuitable for Write Zcopy.

9.5.3 Processing Out-Of-Band Data

When the Data Source ULP posts Out-Of-Band data (a single byte) to be transmitted, the SDP implementation MUST preserve the ordering of the Out-Of-Band data in the output byte stream.

The precise mechanism for conveying OOB data requests from the ULP to the SDP implementation is outside the scope of this specification.

Once the ULP has indicated that a particular byte in its output stream should be marked as Out-Of-Band data, the SDP implementation MUST notify the Remote Peer that Out-Of-Band data is pending by setting the OOB_PEND flag on an outgoing SDP Message. It is RECOMMENDED that this notification be accomplished in an expeditious fashion; however, the only requirements levied by the specification are as follows:

* The OOB_PEND flag MUST be sent exactly once for each OOB data indication.

* The OOB_PEND flag MUST be set on an SDP Message that is sent no later than the SDP Message containing the Out-Of-Band data byte.

* The implementation MUST, if necessary, delay sending the OOB_PEND flag to ensure that no more than 65,535 (2^16-1) bytes of data are sent between the flag and its associated Out-Of-Band data byte. This includes all ULP data sent by any SDP Data Transfer Mechanism, including any data sent in or advertised by the SDP Message containing the OOB_PEND flag.

Note that an implementation MAY send an SDP Message with the OOB_PEND flag using a reserved credit. See section 10.5 Use of Send Credits on page 74.

When the output byte-stream advances to the point where the Out-Of-Band data was inserted into the data stream by the ULP, the Data Source MUST send the Out-Of-Band data using a Data Message with the OOB_PRES bit set and the Out-of-Band data byte as the last byte of the ULP payload in the Data Message.

Upon receipt of an SDP Message with the OOB_PEND flag set, it is RECOMMENDED that the SDP implementation expeditiously notify the ULP

that OOB data is pending; however, the precise mechanism for conveying OOB notifications from the SDP implementation to the ULP is outside the scope of this specification.

9.5.4 SrcAvail Revocation

To revoke all incomplete SrcAvail Messages sent by the Data Source to the Data Sink, the Data Source MUST send a SrcAvailCancel Message. This is needed, for example, if the ULP performs a socket write and a timeout capability is supported. If the timeout interval passes without successful completion of the transfer, the Data Source needs to cancel all RDMA Buffers advertised on behalf of the socket write. Rather than create a new SDP Message type to explicitly acknowledge the SrcAvailCancel Message, the SendSm Message is used because it can be unambiguously understood to complete the cancel operation.

The Data Sink, upon receiving the SrcAvailCancel Message, MUST discard all Unprocessed SrcAvail Messages (SrcAvail Messages that have not been operated on), and SHOULD discard all In-Process SrcAvail Messages (RDMA Read processing has started, but an RdmaRdCompl or SendSm Message to complete the SrcAvail advertisement has not been sent) -- see details below. If all SrcAvail Messages have been Processed, then the Data Sink MUST ignore the SrcAvailCancel Message.

Note that if a SrcAvail Message is In-Process at the Data Sink (i.e., it has initiated one or more RDMA Reads), the RDMA Read cannot be canceled. Because of this and potential head-of-queue blocking due to the mix of control and data on the same connection, it may be some time before the SrcAvail Message is actually canceled.

The Data Sink MUST NOT update the value of MSeqAck to be sent in SDP Messages to greater than or equal to the MSeq value, with wrap, in the SrcAvailCancel Message until all In-Process SrcAvail advertisements have been completed with the following sequence of events:

1. The Data Sink MUST NOT initiate any new RDMA Reads.

2. After completion of all In-Process RDMA Reads, the Data Sink MUST send any relevant RdmaRdCompl Messages (this may or may not complete the SrcAvail Message, depending on how many bytes have been consumed).

3. If there is more ULP data that has not been transferred from the original SrcAvail Message, the Data Sink MUST cancel the remainder of the SrcAvail advertisement.

If the Data Sink canceled one or more SrcAvail advertisements (either Unprocessed or In-Process), the Data Sink MUST send exactly one SendSm Message associated with the oldest incomplete SrcAvail Message.

The Data Source, after sending a SrcAvailCancel Message, MUST NOT send any new SrcAvail Messages until the SrcAvailCancel Message has been Processed as defined below.

This enables the Data Sink to implement simpler accounting (i.e., not have to account for whether a SrcAvail Message was sent before or after the SrcAvailCancel Message).

The Data Source MUST consider the SrcAvailCancel Message Processed if any of the following occur:

* All Unprocessed or In-Process SrcAvail Messages have been moved to the Processed state with an RdmaRdCompl Message or have been overridden by a SinkAvail Message (see section 11.3 Pipelined Mode on page 81).

* A SendSm Message is received with an MSeqAck value greater than or equal to the MSeq value in the SrcAvailCancel Message.

9.5.5 SinkAvail Revocation

To revoke all incomplete SinkAvail advertisements sent by the Data Sink to the Data Source, the Data Sink MUST send the SinkAvailCancel Message. This is needed, for example, if the ULP performs a socket read and a timeout capability is supported. If the timeout interval passes without successful completion of the transfer, all RDMA Buffers advertised on behalf of the socket read need to be canceled.

The Data Source, upon receiving the SinkAvailCancel Message, MUST discard all Unprocessed SinkAvail advertisements (SinkAvail Messages that have not been operated on) and SHOULD cancel all In-Process SinkAvail Messages (RDMA Write processing has started, but an RdmaWrCompl Message that completes the SinkAvail advertisement has not been sent) - see details below. If all SinkAvail advertisements have been Processed, the Data Source MUST ignore the SinkAvailCancel Message.

Because an RDMA Write cannot be canceled when a SinkAvail Message is In-Process at the Data Source (i.e., it has initiated one or more RDMA Writes), the Data Source MUST send an RdmaWrCompl Message after the RDMA Write completes. Because of this and potential head-of-queue blocking due to the mix of control and data on the same connection, it may be some time before the SinkAvail Message is actually canceled.

The Data Source MUST complete the buffer with the following sequence of events:

1. The Data Source MUST NOT initiate any new RDMA Writes.

2. After completion of the In-Process RDMA Writes, the Data Source MUST send any relevant RdmaWrCompl Messages (this may or may not complete the SinkAvail Message, depending on how many bytes have been consumed).

3. If there is more ULP data that has not been transferred into the RDMA Buffer advertised by the SinkAvail Message, the Data Sink MUST discard the remainder of the SinkAvail advertisement.

If the Data Source canceled one or more SinkAvail advertisements (either Unprocessed or In-Process), the Data Source MUST send exactly one SinkCancelAck Message.

The Data Sink, after sending the SinkAvailCancel Message, MUST NOT send a new SinkAvail or SinkAvailCancel Message until all previous SinkAvail Messages have been Processed, as defined below.

This enables the Data Source to implement simpler accounting (i.e., not have to account for whether a SinkAvail Message was sent before or after the SinkAvailCancel Message).

The Data Sink MUST consider the SinkAvailCancel Message Processed if any of the following occur:

* All Unprocessed or In-Process SinkAvail Messages have been moved to the Processed state with an RdmaWrCompl Message (i.e., the byte count returned in the RdmaWrCompl completely consumed the buffer).

* A Data Message that adhered to the stale advertisement rules is received (see section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65).

* A SinkCancelAck Message is received.

9.5.6 Buffering ULP Payload

Under certain conditions it is possible for a sockets application to deadlock unless ULP payload is buffered by the underlying sockets implementation. For example, if all of the following occur:

1. Both ULP peers perform a sockets send followed by a sockets receive,

2. both ULP peers do not post the receive buffer until the send is completed, and

3. the underlying sockets implementation does not buffer the send data

then the send will never complete - thus creating deadlock. To solve deadlock conditions in the most general case (i.e., infinite length sends) is intractable, thus existing sockets applications bound the amount of buffering required by the transport layer through the use of the socket options SO_RCVBUF and SO_SNDBUF.

An application whose behavior is similar to the above example will not deadlock if the application ensures that a send is never larger than the size of the Local Peer's SO_RCVBUF plus the Remote Peer's SO_SNDBUF, and the SDP implementation ensures there is SO_RCVBUF plus SO_SNDBUF amount of buffering in the local and Remote Peer respectively.

An application may post buffers larger than SO_RCVBUF plus SO_SNDBUF - but to remain deadlock free it must ensure that it does not exhibit the above behavior (e.g., a backup application could post large sends in one direction after it is sure the Remote Peer is posting receives).

Specification of the exact buffering algorithm is beyond the scope of this specification, but care must be taken if the receive Private Buffer pool is used as part of the SO_RCVBUF buffers. This is because an entire receive Private Buffer may, in some situations, contain only one byte of ULP data instead of being filled completely.

Thus an SDP implementation SHOULD provide at least SO_RCVBUF amount of buffering for ULP data at the Data Sink. An SDP implementation SHOULD also provide at least SO_SNDBUF amount of ULP data buffering at the Data Source.

10 Private Buffer Management

SDP uses credit-based flow control on a per-socket connection basis. Each peer, for each connection, posts some number of Private Buffers as receive requests to the Receive Queue of the QP associated with the socket. The number of currently posted receive Private Buffers is advertised by the Local Peer to the Remote Peer in the Bufs field in the BSDH of each SDP Message.

Private Buffers MUST obey the following enumerated constraints:

1. All receive Private Buffers MUST be at least as large as the advertised buffer size. See section 10.6 Receive Buffer Resizing on page 75 for receive Private Buffer constraints when resizing.

2. The total number of receive Private Buffers MUST be at least 3 per-connection for normal data flow. See section 10.5 Use of Send Credits on page 74 for detail on how send credits are used. It is RECOMMENDED the number of receive Private Buffers be substantially greater than 3 to enable practical data transfer using the Bcopy mechanism.

3. A Local Peer MUST NOT send SDP Messages larger than the size of the Remote Peer’s receive Private Buffers.

4. The sizes of both send and receive Private Buffers MUST be at least the size of the BSDH plus the size of the largest extended header in an SDP Message (which is SinkAH) plus one byte.

5. The Data Sink Bcopy Threshold MUST be greater than or equal to the size of the Local Peer’s receive Private Buffers.

In addition, send buffers MAY obey the following constraint:

* If the Local Peer’s send buffer size is larger than the Remote Peer’s receive Private Buffer size, the Local Peer MAY reduce the size of its send buffers or leave them unmodified. The latter approach may be advantageous if the Remote Peer enlarges its receive Private Buffers at a later time.

10.1 SDP Message Ordering

The SDP sender MUST insert SDP Messages into the Send Queue in BSDH MSeq order.

This means the SDP Message MSeq value in the BSDH will be monotonically increasing in the Send Queue. The SDP receiver MUST process all SDP Messages in BSDH MSeq order.

10.2 Send Credit Calculation

Send credit is calculated using information in the BSDH included with each SDP Message.

Consider the case of peer 1 sending an SDP Message to its connected peer, peer 2. The header of the SDP Message includes the number of receive Private Buffers peer 1 currently has posted on that connection (in the Bufs field of the BSDH). The header also includes the sequence number of the last SDP Message peer 1 has received before sending this SDP Message (in the MSeqAck field of the BSDH - see section 9.5.4 SrcAvail Revocation on page 69 for additional constraints on MSeqAck). Upon receiving this SDP Message, peer 2 uses this information to update its send credit for that connection:

New send credit = bufs - WrapSubtract(LSSeq - MSeqAck)

where LSSeq (“Last Sent Sequence number”) is the MSeq of the last SDP Message sent by peer 2.

See section 10.5 below for the detailed rules governing usage of available send credits.

10.3 Initialization of Send Credit

Initial send credit advertisements are exchanged during connection setup in the Buf field of the BSDH within the Hello and HelloAck Messages. The initial send credit advertisements from each peer MUST be greater than or equal to three. Either before or after connection setup, the receiver MAY post additional receive Private Buffers and increase the advertised window.

10.4 Gratuitous Update of the Remote Peer’s Send Credit

As previously mentioned, credit updates are included in the header of each SDP Message. Therefore, when ULP data flow is such that SDP Message flow is bi-directional (e.g., when doing Zcopy data transfer), credits are refreshed as part of the data transfer process. In some scenarios, bi-directional SDP Message flow does not occur. Under these circumstances, SDP MUST send gratuitous Data Messages (Data Messages with no ULP payload) as required to update the Remote Peer's send credit.

10.5 Use of Send Credits

The sender MUST reserve two receive Private Buffer credits to ensure the SDP connection operates correctly under flow controlled conditions.

The sender MUST reserve one credit for an SDP Message that provides additional credits. If this credit is not reserved, a deadlock scenario is possible if both peers become flow controlled. Reserving a receive Private Buffer for the flow control update ensures that the sender can always update the receiver when more receive Private Buffers are posted.

The sender MUST reserve one additional credit for sending any SDP Message that does not contain ULP payload. This ensures that the credit can be refreshed by the Remote Peer without depending upon ULP receive behavior. If ULP payload is allowed to be present in the SDP Message, it is possible to have protocol deadlock.

Before sending any SDP Message over the connection, an SDP implementation MUST compute its available send credit as detailed in section 10.2, and MUST then obey the following rules:

* If no credits are available, an implementation MUST NOT send any type of SDP Message.

* If one credit is available, an implementation MUST only send SDP Messages that provide additional credits and do not contain ULP payload.

* If two credits are available, an implementation MUST only send SDP Messages that do not contain ULP payload.

* An SDP implementation MUST send an SDP Message that provides additional credit(s) if the Remote Peer's credits drop to one or fewer credits. The sending of this SDP Message by the Local Peer MUST NOT be contingent upon the Local Peer first receiving some other SDP Message from the Remote Peer.

Note that if three or more credits are available, an implementation can send any type of SDP Message that would otherwise be legal.

10.6 Receive Buffer Resizing

The Local Peer MAY request the Remote Peer to change its receive Private Buffer pool buffer size by sending a Change Receive Buffer Message (ChRcvBuf) with the desired new size. This enables the Local Peer to increase or decrease the maximum size of its outgoing SDP Messages if the Remote Peer agrees to the change. See section 10 Private Buffer Management on page 73 for restrictions on the size of the receive Private Buffers.

If the Local Peer requests a smaller receive Private Buffer in the ChRcvBuf Message, the Local Peer MUST begin using the smaller size immediately after sending the ChRcvBuf Message. If the Local Peer requests a larger receive Private Buffer in the ChRcvBuf Message,

the Local Peer MUST NOT change the local value for the size of the Remote Peer's receive Private Buffers until it receives a ChRcvBufAck Message. When the Local Peer receives the ChRcvBufAck Message, it SHOULD begin using the returned value for the size of the Remote Peer's receive Private Buffers immediately.

The Remote Peer SHOULD change the size of its receive Private Buffers to the desired size specified in the ChRcvBuf Message; it MAY make them larger than the desired size, for example for alignment or performance optimization. If the Remote Peer is unable or unwilling to change its receive Private Buffer size in this manner, it SHOULD change the size to be as close as possible.

Upon receipt of the ChRcvBuf Message, if the Local Peer requests a decrease, the Remote Peer MUST either decrease the size or leave it unchanged, and it MUST NOT decrease the Private Buffer size to be smaller than that requested. Conversely, if the Local Peer requests an increase, the Remote Peer MUST either increase the size or leave it unchanged.

To confirm the change, the Remote Peer MUST send a ChRcvBufAck Message with the new size of its receive Private Buffers. The Remote Peer MAY send a ChRcvBufAck Message immediately if the new size is smaller than or equal to the old size. If the new size is larger than the old size, the Remote Peer MUST send a ChRcvBufAck Message after all receive Private Buffers of the old size have been consumed. If the Remote Peer is unable to resize its receive Private Buffers, it MUST specify in the ChRcvBufAck Message the original receive Private Buffer size.

The Remote Peer MUST continue to use the old receive Private Buffer size to determine whether a SinkAvail Message can be sent for a specific ULP Buffer until it has sent the ChRcvBufAck Message. At that time, the Remote Peer MUST use the new receive Private Buffer size to determine whether a SinkAvail Message may be sent.

If the ChRcvBuf Message requested an increased size and the ChRcvBufAck Message contains a size that is the same as the original size before the ChRcvBuf Message was sent, then the Local Peer MUST NOT request any further size increases for this connection.

If the ChRcvBuf Message requested a decreased size and the ChRcvBufAck Message contains a size that is the same as the original size before the ChRcvBuf Message was sent, then the Local Peer MUST NOT request any further size decreases for this connection.

The Local Peer MUST NOT send a new ChRcvBuf Message if there is an unacknowledged ChRcvBuf Message.

10.6.1 Conflict Resolution

If both peers concurrently send each other ChRcvBuf Messages, then the Accepting Peer MUST disregard the ChRcvBuf Message.

The Connecting Peer MUST respond to the ChRcvBuf Message. The Connecting Peer MAY re-send its ChRcvBuf Message after sending the ChRcvBufAck Message in response to the Accepting Peer's ChRcvBuf Message.

10.6.2 Flow Control Issues During Resizing

When a peer receives the ChRcvBuf Message and it decides to change its receive Private Buffer size in response to this request, the peer MUST allocate new receive Private Buffers of the desired size and post these Private Buffers to the Receive Queue. The peer MUST NOT wait for completion of all posted receive Private Buffers of the previous size before allocating and posting the new (different size) receive Private Buffers. This is required to enable the Remote Peer to continue sending SDP Messages that will cause the old-size receive Private Buffers to complete; otherwise the Remote Peer will stop sending SDP Messages once the channel becomes stalled and the reserved receive Private Buffers will not be consumed.

11 SDP Flow Control Modes

SDP Flow Control Modes control how ULP Buffers larger than the Bcopy Threshold are transferred. Data Source ULP Buffers less than or equal to the Bcopy Threshold MUST be sent using the Bcopy or Transaction mechanism (note that, because the Bcopy Threshold is locally defined, it may be fixed, variable, or be defined as infinite - which would cause the Data Source to always use the Bcopy mechanism). ULP Buffers larger than the Bcopy Threshold are sent in a variety of ways depending upon the current Flow Control Mode. The three Flow Control Modes are:

* Combined Mode - the initial Mode. This Mode enables both Bcopy and Read Zcopy Data Transfer Mechanisms, but with only one outstanding Read Zcopy operation at a time. The SrcAvail Message contains a non-zero length ULP payload. This Mode is used primarily when the Data Sink ULP is not pre-posting receive buffers.

* Pipelined Mode - All Data Transfer Mechanisms are valid, including multiple outstanding transfers at one time (with some limits). The SrcAvail Message contains no ULP payload.

* Buffered Mode - only the Bcopy Data Transfer Mechanism is valid. The main difference between this Mode and Combined Mode is that the Data Source cannot generate SrcAvail Messages.

In all Modes, the Data Sink MAY force the Data Source to transfer data via the Bcopy mechanism. In Buffered Mode, the Data Source always uses the Bcopy mechanism. In Combined or Pipelined Mode, the Data Sink MAY force data transfer using the Bcopy mechanism by issuing a SendSm Message. In this case, however, an extra round trip is required to cause the Bcopy mechanism to be used because of the SrcAvail/SendSm sequence. Buffered Mode eliminates this extra overhead.

Pipelined Mode is the highest performance Mode. It enables multiple outstanding Zero-copy transfers, optimizing for either the Data Sink ULP Buffer being posted first (Write Zcopy) or the Data Source ULP Buffer being posted first (Read Zcopy). Pipelined Mode also enables mixing of Bcopy and Zcopy mechanisms.

The Flow Control Mode between peers MUST be independent in each direction.

For example, data flow from the Local Peer to its Remote Peer could use Buffered Mode in one direction, but the reverse direction could use Combined Mode. Figure 24 summarizes the various characteristics of each Mode. Figure 25 summarizes the possible actions at the Local

Peer A for each combination of Modes between the Local Peer A and Remote Peer B. In the figure, a "1" in the Send or Accept column indicates that the specific SDP Message type is valid, but only one can be outstanding at a time.

Mode	Multiple Outstand Zcopy Requests	Simult. Outstand. SinkAvail & SrcAvail	Data in SrcAvail	Mix ULP Data Msgs w/ Write Zcopy	Mix ULP Data Msgs w/ Read Zcopy	List of Avail Xfer Mech.
Buf.	N/A	N/A	N/A	N/A (Wr. Zcopy not allowed)	N/A (Read Zcopy not allowed)	Bcopy, Trans*
Comb.	No	No	Yes	N/A (Wr. Zcopy not allowed)	Yes, but not at same time	Bcopy, Read Zcopy, Trans*
Pipe.	Yes	Yes	No	Yes	Yes, but not at same time	Bcopy, Read Zcopy, Write Zcopy, Trans*

*if the reverse half-connection is in Pipelined Mode

Figure 24 Mode Characteristics

Half Connection		Host A is allowed to:							
A to B	B to A	Post		Send			Accept		
		RDMA Read Req	RDMA Write	Src Avail	Sink Avail	Trans-action	Src Avail	Sink Avail	Trans-action
Buffered	Buffered	NO	NO	NO	NO	NO	NO	NO	NO
Buffered	Combined	YES	NO	NO	NO	NO	1	NO	NO
Buffered	Pipelined	YES	NO	NO	YES	YES	YES	NO	NO
Combined	Buffered	NO	NO	1	NO	NO	NO	NO	NO
Combined	Combined	YES	NO	1	NO	NO	1	NO	NO
Combined	Pipelined	YES	NO	1	YES	YES	YES	NO	NO
Pipelined	Buffered	NO	YES	YES	NO	NO	NO	YES	YES
Pipelined	Combined	YES	YES	YES	NO	NO	1	YES	YES
Pipelined	Pipelined	YES	YES	YES	YES	YES	YES	YES	YES

Figure 25 Summary of Permitted Actions By Mode Pair

11.1 Buffered Mode

In Buffered Mode, the Data Source MUST either transfer all data using the Bcopy mechanism or optionally, if the opposite half-connection is in Pipelined Mode, the Transaction mechanism.

Thus, only Data or SinkAvail Messages can be used by the Data Source to transfer ULP data.

11.2 Combined Mode

In Combined Mode, if the send ULP Buffer is less than or equal to the Data Source Bcopy Threshold, the Data Source MUST either use the Bcopy mechanism (i.e., by sending Data Messages) or optionally, if the opposite half connection is in Pipelined Mode, the Transaction mechanism. If the ULP Buffer is larger than the Bcopy Threshold, data MUST be transferred using the Read Zcopy mechanism.

In Combined Mode the Data Sink MUST be prepared to receive ULP data through either the Bcopy mechanism, the Read Zcopy mechanism, or the Transaction mechanism.

In Combined Mode, if the Read Zcopy mechanism is used, after the Data Source sends a SrcAvail Message, it MUST NOT send any SDP Messages containing a ULP payload until all data transfer associated with the SrcAvail Message is complete (specifically, a RdmaRdCompl or SendSm Message is received).

This effectively means that only a single SrcAvail Message may be In-Process at any one time, and the Data Source MUST NOT use the Bcopy Data Transfer Mechanism if a Read Zcopy is In-Process.

In Combined Mode, the SrcAvail Message MUST contain greater than zero bytes of ULP payload. The actual amount of ULP data included is implementation dependent.

11.3 Pipelined Mode

In Pipelined Mode, if the ULP Buffer is less than or equal to the Bcopy Threshold, the Data Source MUST use either the Bcopy mechanism (e.g., by sending Data Messages) or optionally, if the opposite half connection is in Pipelined Mode, the Transaction mechanism. If the ULP Buffer is larger than the Bcopy Threshold, data MUST be transferred using either the Read Zcopy mechanism or the Write Zcopy mechanism.

In Pipelined Mode the Data Sink MUST be prepared to receive ULP data through any of the Data Transfer Mechanisms.

In Pipelined Mode, the Data Source MUST NOT include ULP payload in any SrcAvail Messages.

After sending one or more SrcAvail Messages, the Data Source MUST NOT send any SDP Messages with ULP payload until all data transfers associated with previously sent SrcAvail Message(s) have been Processed. The single exception to this is the case when the Data Sink sends a SendSm Message. In this case, the Data Source MUST send the remaining data associated with the SrcAvail through Data Messages. The remaining Unprocessed or In-Process SrcAvail advertisements remain valid and MUST be Processed by the Data Sink after it consumes these Data Messages.

This restriction is necessary since a crossing SinkAvail Message cancels any advertised SrcAvails, and the Sink would be unable to process the received in-line ULP payload until it received and processed all the data associated with the canceled SrcAvails. See 9.2 Read Zcopy on page 56 for further details.

The Data Sink MAY also advertise receive RDMA Buffers using SinkAvail Messages (i.e., Write Zcopy mechanism, which uses RDMA Writes). If SrcAvail and SinkAvail Messages cross, then Write Zcopy has higher priority than Read Zcopy (i.e., the SrcAvail Messages are canceled).

The implementation MUST obey the following rules for crossing SinkAvail/SrcAvail advertisements. The Data Source MUST give precedence to discarding stale SinkAvail advertisements using the algorithm described in section 9.5.1 Detecting Stale SinkAvail Advertisements on page 65 over the requirements listed below.

1. If the Data Source receives a SinkAvail Message:

    a. the Data Source MUST use the Write Zcopy mechanism to transfer data - even if it has already advertised the ULP send data through a SrcAvail Message.

    b. the Data Source MUST treat all outstanding SrcAvail advertisements as having been discarded by the Data Sink.

        This implies that if the Data Source consumes all SinkAvail advertisements and ULP send data remains that is suitable for RDMA, the Data Source SHOULD advertise the ULP data through a SrcAvail Message, even if a SrcAvail advertisement for that send RDMA Buffer was sent prior to receiving the SinkAvail Message.

2. If the Data Sink receives a SrcAvail Message,

    a. and it has no Unprocessed or In-Process SinkAvail Message(s), the Data Sink MUST NOT send a SinkAvail Message and MUST transfer data using the Read Zcopy mechanism.

    b. and a SinkAvail Message is Unprocessed or In-Process, the Data Sink MUST discard all Unprocessed or In-Process SrcAvail advertisements and MUST ignore the current SrcAvail advertisement (but otherwise process the packet normally, e.g., flow control information, etc.).

In Pipelined Mode, an implementation MAY advertise multiple send/receive RDMA Buffers by sending multiple SrcAvail/SinkAvail Messages without waiting for data transfers associated with prior SrcAvail/SinkAvail Messages to complete.

The Data Sink MUST limit the maximum number of outstanding SrcAvail and SinkAvail advertisements to the HH or HAH MaxAdverts value specified by the Remote Peer during connection setup.

12 SDP Mode Transitions

An SDP implementation MUST only support the transitions between modes defined in Figure 26. Each SDP Mode has a Mode Master, which controls any Mode changes, and a Mode Slave, which passively changes Mode when told by the Mode Master. For each Flow Control Mode, the Mode Master and Mode Slave MUST be as defined in Figure 27.

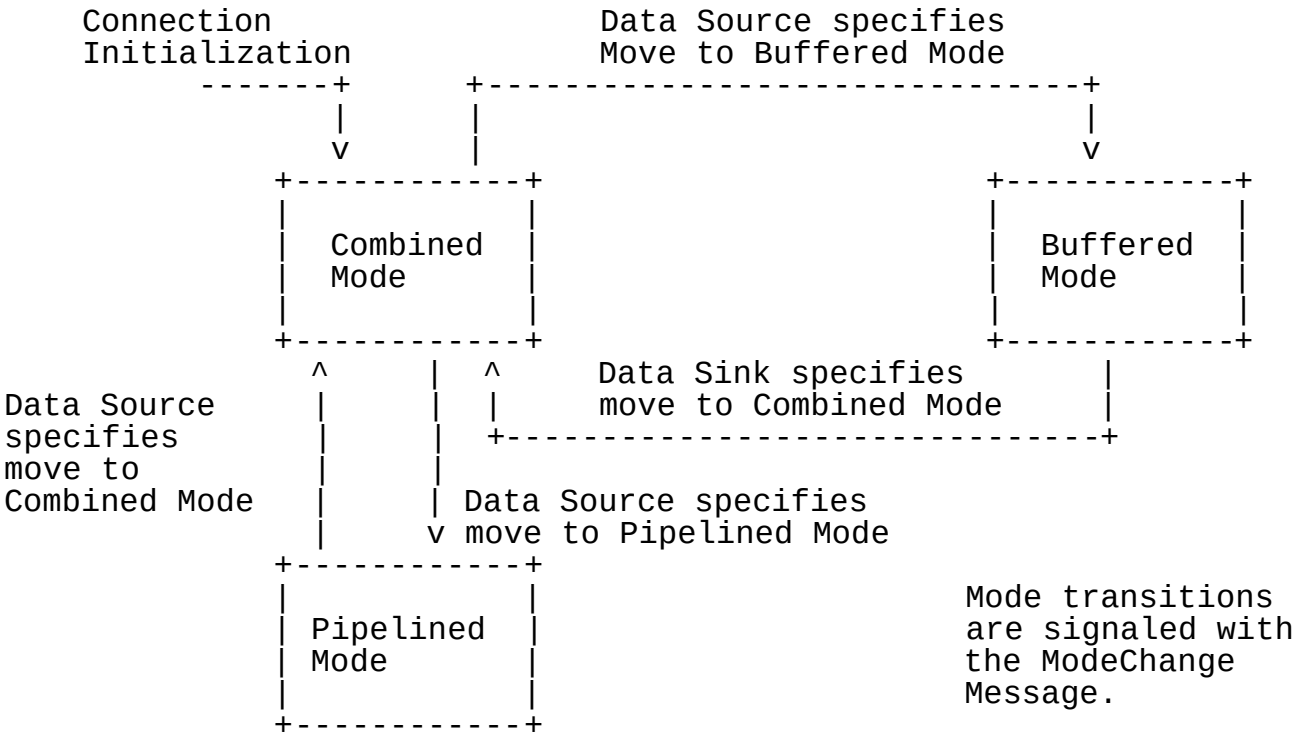

Figure 26 Mode State Machine for a Half-Connection

An SDP implementation MUST NOT send a ModeChange Message that specifies the current Mode.

The Mode Slave MAY indicate to the Mode Master that a Flow Control Mode change is recommended, by either setting the REQ_PIPE flag in the BSDH (transition from Combined Mode to Pipelined Mode), or by using SendSm to transfer ULP data (transition from Pipelined Mode to Combined Mode or from Combined Mode to Buffered Mode). The Mode Master MAY choose to ignore the request.

Mode	Master	Mode Change Hint From Slave
Buffered	Data Sink	None
Combined	Data Source	SendSm Messages or REQ_PIPE
Pipelined	Data Source	SendSm Messages

Figure 27 Mode Master

The Mode Master MUST change its Flow Control Mode immediately after sending a ModeChange Message.

The Mode Master MAY change its Flow Control Mode before the completion of the transfer of the ModeChange Message. In a specific Flow Control Mode, the Mode Master MUST only use Data Transfer Mechanisms allowed for that Mode (see section 11 SDP Flow Control Modes on page 78). For example, if the prior Mode was Combined Mode and the current Mode is Buffered Mode, the Data Source must not generate SrcAvail Messages.

When the Mode Slave receives the ModeChange Message, the slave MUST immediately set its current Flow Control Mode to the Mode specified in the ModeChange Message.

In a specific Mode, the Mode Slave MUST only use Data Transfer Mechanisms allowed for that Mode. For example, if the prior Mode was Pipelined Mode and the current Mode is Combined Mode, the Data Sink (slave) MUST NOT generate SinkAvail Messages.

Depending on the Mode transition, the Mode Master and Mode Slave MAY be required to take some further actions, as described later in this chapter.

When a connection is first set up, the Local Peer MUST set the initial Flow Control Mode for the local Data Sink and Data Source to be Combined Mode (see section 8 Connection Setup on page 47).

The following subsections give details of SDP Message exchanges needed to transition from one Flow Control Mode to another. Each transition is caused by sending a ModeChange Message.

12.1 Transition from Combined Mode to Buffered Mode

To transition from Combined to Buffered Mode, the Data Source (Mode Master) MUST send a ModeChange Message with the MCH fields set as follows:

* S=0, (i.e., change the Data Sink Mode)

* Mode = BUFF_MODE (see section 6.3.8.2 Mode - 3 bits on page 26)

The Data Source MUST NOT send the ModeChange Message until all Unprocessed or In-Process Read Zcopy transfers have been moved to the Processed state with either a RdmaRdCompl or SendSm Message from the Data Sink (there will be at most one outstanding in Combined Mode).

This ensures that no SDP Messages specific to Combined Mode (i.e., those related to the Read Zcopy mechanism) can be received by either peer when in Buffered Mode.

12.2 Transition from Buffered Mode to Combined Mode

To transition from Buffered to Combined Mode, the Data Sink (Mode Master) MUST send a ModeChange Message with the MCH fields set as follows:

* S=1 (i.e., change the Data Source Mode)

* Mode = COMB_MODE (see section 6.3.8.2 Mode - 3 bits on page 26)

Because all SDP Message types that are legal for Buffered Mode are also legal for Combined Mode, no special action is needed for this transition.

12.3 Transition From Combined Mode to Pipelined Mode

To transition from Combined to Pipelined Mode, the Data Source (Mode Master) MUST send a ModeChange Message with the MCH fields set as follows:

* S=0 (i.e., change the Data Sink Mode)

* Mode = PIPE_MODE (see section 6.3.8.2 Mode - 3 bits on page 26)

Because all SDP Message types that are legal for Combined Mode are also legal for Pipelined Mode, no special action is needed for this transition. For example, if the Data Source has an outstanding SrcAvail advertisement (and there can be at most one such advertisement outstanding in Combined Mode), then the Data Source need not wait for a RdmaRdCompl or SendSm before sending the ModeChange Message.

12.4 Transition From Pipelined Mode to Combined Mode

To transition from Pipelined to Combined Mode, the Data Source (Mode Master) MUST send a ModeChange Message with the MCH fields set as follows:

* S=0 (i.e., change the Data Sink Mode)

* Mode = COMB_MODE (see section 6.3.8.2 Mode - 3 bits on page 26)

After sending the ModeChange Message, the Data Source MUST immediately transition to Combined Mode, and the Data Source MUST NOT subsequently switch to any other Flow Control Mode until it receives an SDP Message with the following constraint:

MSeqAck >= (MSeq of the ModeChange Message)

In the above calculation, MSeqAck and “MSeq of the ModeChange Message” are treated as signed integers.

This constraint ensures that all stale SinkAvail Messages will be received at the Data Source before a transition out of Combined Mode. The behavior upon receiving a stale SinkAvail Message while in Combined Mode is described later in this section.

If data transfer is occurring, the Data Source is guaranteed to eventually receive the acknowledgement for the ModeChange Message. If no data transfer is occurring, the acknowledgement can take an indeterminate amount of time. However, there is no need to immediately switch out of Combined Mode if no data transfer is occurring.

The transition from Pipelined to Combined Mode imposes additional constraints on the Data Source and Data Sink.

The transition from Pipelined Mode to Combined Mode MUST be governed by Figure 28 Data Source Transition from Pipelined to Combined Mode on page 88 and Figure 29 Data Sink Transition from Pipelined to Combined Mode on page 89.

Before the Data Source transitions from Pipelined to Combined Mode, the Data Source MUST complete any RDMA Writes that are In-Process, and issue all RdmaWrCompl Messages before sending a ModeChange Message.

This ensures that RDMA Writes will not be used in Combined Mode.

Before the Data Source transitions from Pipelined to Combined Mode, if the Data Source has incomplete SinkAvail advertisements, the Data

Source MUST discard those advertisements and MUST re-issue all Unprocessed or In-Process SrcAvail Messages, if there are any.

After the transition to Combined Mode, normal Combined Mode rules apply. For example, only one SrcAvail Message may be outstanding at any one time and the SrcAvail Message must contain ULP payload. If the Data Source has no incomplete SinkAvail advertisements, but a SinkAvail advertisement is received before the acknowledgement for the ModeChange Message, the Data Source ignores the SinkAvail Message (but process flow control information, etc.) and re-issue all outstanding SrcAvail Messages according to Combined Mode rules. In either case, if any more SinkAvail Messages arrive after the initial discard of SinkAvail Message(s), the Data Source must ignore these SinkAvail Messages (but must process flow control information, etc., normally).

Normal Data Sink behavior in Pipelined Mode requires it to drop any SrcAvail Messages (but process flow control information, etc.) if there is a SinkAvail outstanding.

Thus, the only behavior the Data Sink MUST follow when transitioning to Combined Mode after receiving the ModeChange Message is to invalidate local state associated with any outstanding SinkAvail Messages.

12.5 State Mode Transition Summary

Figure 30 Data Source Mode Transition Events on page 90 and Figure 31 Data Sink Mode Transition Events on page 91 summarize the events and consequent actions for the Data Source and Data Sink, respectively. Advisory input to the Mode Master to change modes is not meant to be exhaustive. The Mode Master MAY change modes at any time, possibly for reasons beyond the scope of this specification. The figures only list events that may cause a Mode transition, or may be an event that is handled uniquely in a specific Mode.

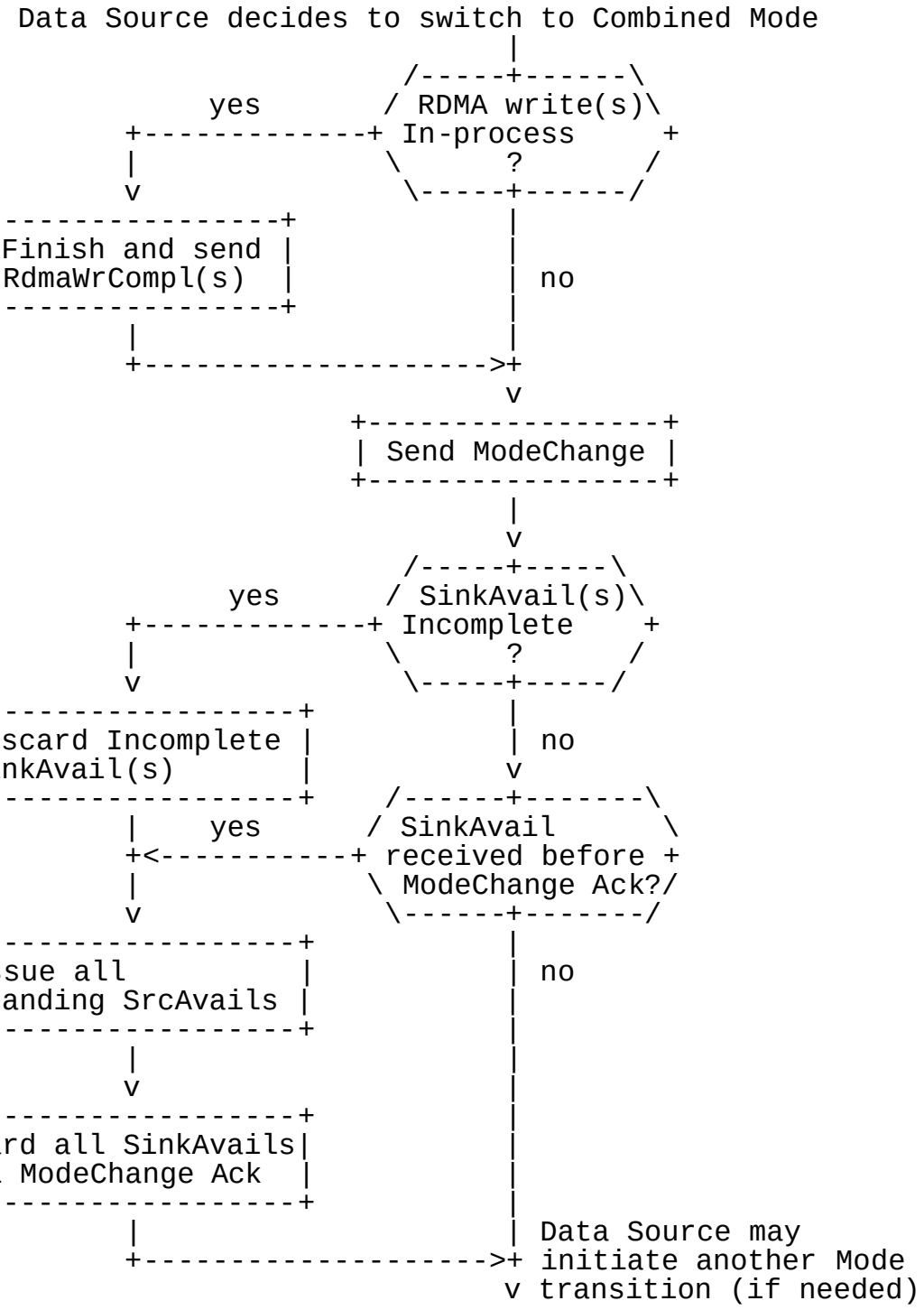

Figure 28 Data Source Transition from Pipelined to Combined Mode

ModeChange Message
Received by Data Sink

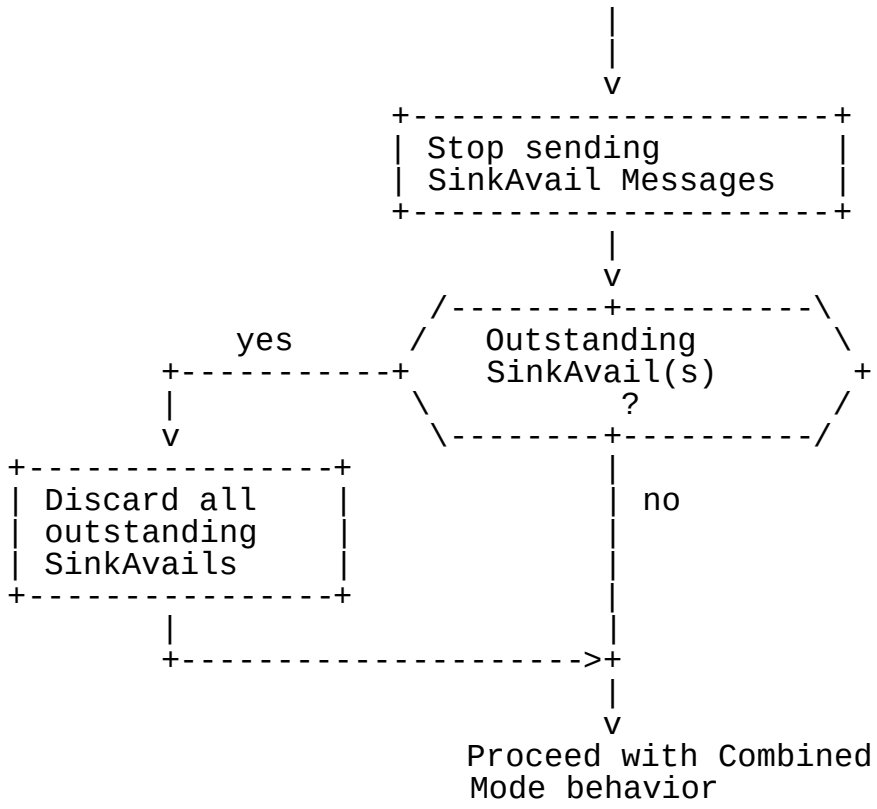

Figure 29 Data Sink Transition from Pipelined to Combined Mode

Data Source Mode	Event	Action/Transition
Combined (master	Receive REQ_PIPE=1 in RdmaRdCompl msg	Advisory: May decide to transition to Pipelined Mode
	Data Source decides to change Modes to Buffered	If outstanding SrcAvail, wait for RdmaRdCompl or SendSm from Data Sink, then send ModeChange Message
	Data Source decides to change to Pipelined Mode	Change to Pipelined Mode and send a ModeChange Message
	Receive SendSm	Advisory: May decide to transition to Buffered Mode
	Receive SinkAvail	Can happen if just transitioned from Pipelined Mode. See Figure 28 Data Source Transition from Pipelined to Combined Mode for processing details.
Buffered (slave)	Receive ModeChange MSG with S=1 and Mode=COMB_MODE	Immediately transition to Combined Mode.
Pipelined (master)	Receive SendSm	Advisory: May decide to transition to Combined Mode
	Data Source decides to change to Comb. Mode	Change to Combined Mode and send a ModeChange Message. See Figure 28 Data Source Transition from Pipelined to Combined Mode for processing details.

Figure 30 Data Source Mode Transition Events

Data Sink Mode	Event	Action/Transition
Combined (slave)	Receive ModeChange Message with S=0 and Mode=BUFF_MODE	Immediately transition to Buffered Mode
	Receive ModeChange Message with S=0 and Mode=PIPE_MODE	Immediately transition to Pipelined Mode
Buffered (master)	ULP receive buffer is posted	Advisory: If the receive ULP buffer is suitable for RDMA, the Data Sink may choose to transition to Combined Mode
	Data Sink decides to change to Combined Mode	Send ModeChange Message
Pipelined (slave)	Receive ModeChange Message with S=0 and Mode=COMB_MODE	Immediately transition to Combined Mode. See Figure 29 Data Sink Transition from Pipelined to Combined Mode for processing details.

Figure 31 Data Sink Mode Transition Events

13 Socket Duplication

When a socket exists in one address space and is then accessed in a different address space (on the same peer), the socket needs to be duplicated into the second address space. Note that if two threads are accessing the socket in the same address space, socket duplication is not required.

Performing socket duplication in user-mode imposes certain restrictions because socket state cannot be shared between the address spaces. In fact, in the context of iWARP networks available today, the socket can only exist in one address space at a time (since RNICs are not required to support sharing Connection Context memory between multiple address spaces). Because of these restrictions, SDP allows only one address space at a time to execute operations that either transfer data or change state for an underlying shared socket. Address spaces dynamically swap control of the underlying socket, as needed, to execute requested operations. The SDP socket duplication procedure serializes operations that different address spaces request on a shared socket. The procedure waits for all In-Process operations to complete before swapping control of an underlying socket to another address space. Logically, the procedure takes control of the underlying socket away from the Controlling Address Space as soon as a non-Controlling Address Space requests an operation on that socket. After control is taken away, the procedure treats the original Controlling Address Space like a non-Controlling Address Space if the original Controlling Address Space requests operations on that socket. In this way a socket may transition back and forth between Controlling Address Spaces based on ULP behavior.

SDP enables socket duplication by bringing the connection to a consistent state, closing the LLP connection, handing the state to the new Controlling Address Space, and then creating a new connection in the new address space. Note that after the connection is suspended and then restarted on a new LLP connection, the connection, by definition, does not have any outstanding SinkAvail or SrcAvail advertisements. Any incomplete SinkAvail or SrcAvail advertisements were effectively canceled during the transition to a new connection.

13.1 Implementing Socket Duplication

An SDP implementation is REQUIRED to support responding to a socket duplication request using the procedure defined for the Remote Peer in section 13.1.1.

If SDP socket duplication initiation is supported, an SDP Local Peer MUST employ the procedure defined in section 13.1.1 when initiating socket duplication. Initiation of socket duplication is OPTIONAL.

13.1.1 Socket Duplication Procedure

1. The SDP implementation in the non-Controlling Address Space MUST enable an accept for incoming connection requests at a TCP destination port.

   This may or may not be the original TCP port number specified during the initial connection setup. The connection request MUST NOT be completed unless the source IP address, destination IP address, and source TCP port number contained in the Hello Message match the expected value.

2. The SDP implementation in the Controlling Address Space MUST wait for all In-Process data transfer operations to complete, then it MUST send a SuspComm Message to the Remote Peer to request a suspension of the session. This SDP Message contains the destination TCP port number received from the non-Controlling Address Space. The Remote Peer MUST connect to this TCP port number when resuming communication (step 4). The Local Peer MUST NOT send additional SDP Messages or perform any RDMA operations from the Controlling Address Space, after sending the SuspComm Message.

3. Upon receiving the SuspComm Message, the Remote Peer MUST wait for all In-Process data transfer operations to complete, then MUST send a SuspCommAck Message indicating that the session is suspended. After sending the SuspCommAck Message, this peer MUST NOT send any more SDP Messages or perform any RDMA operations until a new connection is set up (step 8).

4. The Remote Peer MUST wait for completion of the send of the SuspCommAck Message, then close the LLP connection. The Remote Peer MUST then initiate the new connection to the destination TCP port number received through the SuspComm Message, utilizing the same IP address specified in the prior connection setup sequence. Posting of receive Private Buffers and the contents of the HH MUST follow the same rules as connection setup (see section 8 Connection Setup on page 47).

5. Once the SuspCommAck Message is received, the Controlling Address Space on the Local Peer MUST send a signal to the non-Controlling Address Space through some private means outside the scope of this specification. The Controlling Address Space MUST also send to the non-Controlling Address Space through some private means outside the scope of this specification:

   * Any buffered receive ULP data.

* The Remote Peer's TCP port number (to ensure the parameter does not change when the socket is re-connected).

* The size of the local receive Private Buffers.

* The current values for IRD and ORD.

6. The non-Controlling Address Space MUST accept the connection request from the Remote Peer and initialize its state variables for the new connection. The Hello Message initializes SDP connection state.

7. When both steps 5 and 6 have completed, the (previously) non-Controlling Address Space:

   a. MUST send a HelloAck Message to the Remote Peer (see section 8 Connection Setup on page 47). The receive Private Buffer size parameter in the HelloAck Message MUST be the values received from the Controlling Address Space. The IRD and ORD values MAY be the values received from the Controlling Address Space.

   b. MUST make buffered received ULP data from the Controlling Address Space available to the ULP.

8. When connection setup is complete, the Local Peer MUST resume normal data transfer. See section 8.1.1 iWARP Connection Setup on page 47.

13.1.2 Conflict Resolution

If both peers concurrently send each other SuspComm Messages, then the Accepting Peer MUST disregard the SuspComm Message, while the Connecting Peer MUST respond to the SuspComm Message. The Connecting Peer MUST re-send its own SuspComm Message once communication is re-established.

13.2 SDP Managed Failover

SDP supports Managed Failover by leveraging the Socket Duplication procedure (see section 13 Socket Duplication on page 92). Note that LLP managed failover between RNICs may be done by other mechanisms. Such mechanisms are beyond the scope of this specification.

During socket duplication, a change of address space is occurring. In managed failover, the SDP connection MAY in fact be reestablished using different paths, ports, RNICs or hosts. The original connection in a managed failover scenario is analogous to the Controlling Address Space in socket duplication. The new failed over connection is analogous to the non-Controlling Address Space.

Managed failover changes where one end of the connection is situated. Failing over both ends requires two managed failover operations.

The decision to attempt a managed failover must occur before step 1 of the Socket Duplication procedure. How such a decision is made is dependent on policy and outside the scope of the SDP specification.

If an SDP implementation supports SDP Managed Failover, it MUST do so using the socket duplication procedure with the following change to Step 1:

An implementation MUST choose a new endpoint for the failover connection. How the new endpoint is chosen is a matter of policy. However, the endpoint must be chosen in such a way that the address (see section 7 on page 31) will resolve to the failover port. This address resolution uses the IP addresses from the original connection. As before, the TCP destination port chosen for the SuspComm Message (see section 6.5.1 SuspComm Message on page 29) should resolve to the failover endpoint.

14 SDP Usage of iWARP and LLP Features

14.1 iWARP Message Requirements

A conforming implementation of SDP MUST enable RDMA Reads and RDMA Writes on each connection.

14.2 Solicited Events

An SDP implementation occasionally needs to stop processing on a half-connection and wait for one of the following SDP Messages to arrive before proceeding further:

1. Flow control credit update SDP Message - flow control credits for the receive Private Buffer pool have been exhausted, thus it cannot send data, control, and/or RDMA advertisement to the peer;

2. Data or RDMA advertisement SDP Message - the ULP has indicated its interest in data via a sockets interface select call or receive;

3. RDMA completion (or cancel) SDP Message - before it can de-register a send or receive RDMA Buffer it must either complete the RDMA transfer or receive a cancel acknowledgement message.

A typical SDP implementation would request completion queue notification and block the ULP process (or thread) until the appropriate SDP Message arrives and the notification is delivered. The goal of using iWARP Send with Solicited Event Message is to minimize completion queue notification events and corresponding process (or thread) wake-ups when the arriving SDP Message does not match the class of SDP Messages that the implementation requires. For example, if the Data Source is waiting for a RdmaRdCompl Message to complete a send ULP Buffer and there is no local receive ULP Buffer posted or local invocation of a sockets interface select on the opposite half-channel (local Data Sink), it should not receive notifications for the opposite half-connection if the peer sends a Data Message or SrcAvail RDMA advertisement.

To accomplish this goal, all SDP Messages are subdivided into solicited or unsolicited SDP Messages.

Solicited SDP Messages are those that most likely require immediate attention regardless of ULP behavior at the Remote Peer and regardless of whether the Remote Peer is waiting for other (unsolicited) SDP Messages. Solicited SDP Messages are defined as:

* AbortConn, SuspComm, SuspCommAck, SendSm, SrcAvailCancel, SinkAvailCancel, HelloAck - because it is essential for the

sender of these SDP Messages that its peer react to them as soon as possible;

* RdmaWrCompl, RdmaRdCompl, SinkCancelAck - because the peer most likely needs to deregister and release RDMA Buffers to the ULP upon reception of these SDP Messages;

* Data with OOB_PRES or OOB_PEND bit set - because the peer SDP implementation most likely needs to notify the ULP as soon as possible that this SDP Message has been received.

An SDP implementation MUST use an iWARP Send with Solicited Event or Send with Solicited Event and Invalidate Message for solicited SDP Messages.

Unsolicited SDP Messages are those that may require immediate attention by the Remote Peer, but only that Peer can decide whether or not a notification is necessary - it depends on the ULP behavior or the implementation of that Remote Peer. Unsolicited SDP Messages are defined as:

* DisConn, SrcAvail, Data without OOB_PEND or OOB_PRES bit set - because the peer only needs to immediately process these SDP Messages when the ULP has issued a sockets interface select or receive request;

* ModeChange, ChRcvBuf, ChRcvBufAck - because after receiving these SDP Messages, the peer only needs to take action for new SDP Messages it generates itself, or for SDP Messages that follow that are solicited SDP Messages (e.g., it will not be blocked specifically waiting for these SDP Messages).

An SDP implementation MUST use an iWARP Send Message or Send with Invalidate Message for Unsolicited SDP Messages.

Solicited events are never applicable for RDMA Read or RDMA Write operations.

14.3 Keepalive Messages

The sockets interface provides the ULP with the capability to periodically transmit messages to the peer (which require an answer) to determine if the peer is still alive (SO_KEEPALIVE). This is referred to as the keepalive feature, and the associated messages are known as keepalive messages. The keepalive feature is OPTIONAL. If an SDP implementation supports the keepalive feature, then it MUST implement this functionality by turning on the TCP keepalive timer.

15 Security Considerations

This section describes security issues related to the implementation and use of the Port Mapper and SDP protocols. Only the security issues specific to those protocols are described here. Security issues for the Direct Data Placement (DDP) and Remote Direct Memory Access Protocol (RDMAP) are covered in a separate document, [RDDP-Security].

Both the Port Mapper service provider (PMSP) and an SDP implementation are required to communicate with Untrusted Remote Peers. Therefore, countermeasures should be put into place to guard against the different types of deliberate attacks that could be launched by Remote Peers.

This section describes many of the attacks that are possible, but may not be comprehensive. More attacks may be possible than have been enumerated here.

15.1 Spoofing

Spoofing is a potential issue for both the Port Mapper and SDP protocols. A man-in-the-middle with the appropriate capabilities (primarily the ability to generate packets that will be accepted by the LLP) could generate attacks that result in denial of service, tampering, information disclosure and repudiation.

The best protection against this form of attack is through the use of end-to-end authentication, such as IPSec.

15.2 Denial of Service (DOS)

15.2.1 Port Flooding

A DOS attack against the Port Mapper service provider is relatively simple to launch. The most straightforward attack is to send a flood of UDP messages to the PMSP port. A similar style of attack could be launched against the SDP listen port, keeping legitimate requests from being able to access the port. Flooding a PMSP is potentially more serious than an SDP listen port. For most configurations, flooding the PMSP port is likely to affect more clients because generally there are fewer PMSPs to handle requests than nodes supplying SDP services, but there is at least one SDP listen port is present per node.

One potential countermeasure is to set an aggressive timeout on the state created due to a PMReq Message and release the associated resources on expiration of the timer. Another countermeasure that could be used alone or in conjunction with a timeout is to track the rate of attack and raise an alarm to management indicating an

arrival rate outside of the expected bounds from either one or multiple nodes. Management could take steps to source quench any node that attempts partial mappings that timeout more than N times within a defined time period. In addition, the Port Mapper service provider (PMSP) could be implemented as a distributed service, making it more difficult for an attacker to flood all of the PMSPs in the system at the same time.

15.2.2 Resource Consumption by an Idle Process

15.2.2.1 Port Mapper

When the attacker builds a valid PMReq message, in addition to potentially keeping other legitimate clients from being able to get their requests through to the PSMP, it is possible for the attacker to consume all of the resources associated with the SDP listen port(s) managed by the PSMP.

The countermeasures taken to thwart the flood of incoming requests should help to mitigate the potential damage. The same countermeasures described in 15.2.1 Port Flooding apply to this scenario.

15.2.2.2 SDP Protocol

An attacker could consume even more resources by sending valid SDP Hello Messages in an attempt to establish multiple SDP connections. This requires more sophistication by the attacker because the Stream already must have been ESTABLISHED, and the attacker must be able to send valid SDP Hello Messages. Additional resources are allocated for the SDP connection context, if the connection request is accepted.

This type of attack is less likely because it requires more sophistication and effort on the part of an attacker.

One countermeasure is to install a filter against the attacker, since it is possible to tell for sure who it is given the Stream is already established, and the attacker must be able to send and receive data.

16 IANA Considerations

Issue: The Port Mapper requires an IANA port - need to get one allocated by IANA and document it here.

The Port Mapper Service by default MUST use UDP Port [TBD by IANA].

17 References

17.1 Normative References

[RFC1982] Elz, R., Bush, R., "Serial Number Arithmetic", RFC1982, August 1996.

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

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

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

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[MPA] P. Culley et al., "Markers with PDU Alignment", IETF Internet Draft draft-cully-iwarp-mpa-01, October 2002 .

[DDP] H. Shah et al., "Direct Data Placement over Reliable Transports", IETF Internet Draft draft-shah-iwarp-ddp-01, October 2002 .

[RDMAP] R. Recio et al., "RDMA Protocol Specification", IETF Internet Draft draft-recio-iwarp-01, October 2002 .

17.2 Informative References

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

[VERBS] J. Hilland et al., "RDMA Protocol Verbs Specification", IETF Internet Draft draft-hilland-iwarp-00, April 2003 .

[IBTA-SDP] InfiniBand Architecture Specification, Volume 1, Release 1.1 (http://www.infinibandta.org/specs)

[RDMAP-SECURITY] J. Pinkerton et al., “DDP/RDMAP Security”, IETF Internet Draft draft-pinkerton-rdddp-security-00, June 2003.

18 Author’s Addresses

James Pinkerton
Microsoft Corporation
One Microsoft Way
Redmond, WA. 98052 USA
Phone: +1 (425)705-5442
Email: jpink@windows.microsoft.com

Ellen Deleganes
Intel Corporation
MS JF3-206
2111 NE 25th Ave.
Hillsboro, OR 97124 USA
Phone: +1 (503) 712-4173
Email: ellen.m.deleganes@intel.com

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

19 Acknowledgments

John Carrier
Adaptec, Inc.
691 S. Milpitas Blvd.
Milpitas, CA 95035 USA
Phone: +1 (360) 378-8526
Email: john_carrier@adaptec.com

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

Mike Penna
Broadcom Corporation
16215 Alton Parkway
Irvine, CA 92619-7013 USA
Phone: +1 (949) 926-7149
Email: MPenna@Broadcom.com

Uri Elzur
Broadcom Corporation
16215 Alton Parkway
Irvine, CA 92619-7013 USA
Phone: +1 (949) 585-6432
Email: Uri@Broadcom.com

Ted Compton
EMC Corporation
Research Triangle Park, NC 27709 USA
Phone: +1 (919) 248-6075
Email: compton_ted@emc.com

Frank Berry
Intel Corporation
2111 NE 25th Ave.
Hillsboro, OR 97124 USA
Phone: +1 (503) 712-3897
Email: frank.berry@intel.com

Tom Talpey
Network Appliance
375 Totten Pond Road
Waltham, MA 02451 USA
Phone: +1 (781) 768-5329
Email: thomas.talpey@netapp.com

Dwight Barron
Hewlett-Packard Company
20555 SH 249
Houston, TX 77070-2698 USA
Phone: +1 (281) 514-2769
Email: Dwight.Barron@Hp.com

Mallikarjun Chadalapaka
Hewlett-Packard Company
8000 Foothills Blvd.
Roseville, CA 95747-5668, USA
Phone: +1 (916) 785-5621
Email: cbm@rose.hp.com

Bill Edwards
Hewlett-Packard Company
20555 SH 249
Houston, TX 77070-2698 USA
Phone: +1 281 518 9034
Email: w.Edwards@hp.com

Dave Garcia
Hewlett-Packard Company
19333 Vallco Parkway
Cupertino, CA 95014 USA
Phone: +1 (408) 285-6116
Email: dave.garcia@hp.com

Jeff Hilland
Hewlett-Packard Company
20555 SH 249
Houston, TX 77070-2698 USA
Phone: +1 (281) 514-9489
Email: jeff.hilland@hp.com

Renato Recio
IBM Corporation
11501 Burnett Road
Austin, TX 78758 USA
Phone: +1 (512) 838-1365
Email: recio@us.ibm.com

John L. Hufferd
IBM Corp.
650 Harry Rd.
San Jose CA
Phone: +1 (408) 256-0403
Email: hufferd@us.ibm.com

Mike Ko
IBM Corp.
650 Harry Rd.
San Jose, CA 95120, USA
Phone: +1 (408) 927-2085
Email: mako@us.ibm.com

James Livingston
NEC Solutions (America), Inc.
7525 166th Ave. N.E., Suite D210
Redmond, WA 98052-7811
Phone: +1 (425) 897-2033
Email: james.livingston@necsam.com

Hemal Shah
Intel Corporation
MS PTL1
1501 South MoPac Expressway, #400
Austin, TX 78746, USA
Phone: +1 (512) 732-3963
Email: hemal.shah@intel.com

20 Full Copyright Statement