Solaris iSER

(iSCSI Extensions for RDMA)

Version 1.0

May 2008

Table of Contents

1 Introduction

1.1 Background

The iSCSI Extensions for RDMA (iSER) project will implement both an iSER initiator and iSER target for Solaris. iSER utilizes an RDMA capable protocol (RCaP) to transfer data directly into SCSI buffers without intermediate copies. Initially, the Solaris iSER implementation will utilize the InfiniBand Reliable Connected (RC) transport via the Solaris InfiniBand Transport Interface (IBTI). In the future, iWARP or other RDMA-capable protocols could be employed.

iSER will provide a storage interconnect that leverages all the capabilities of InfiniBand RC, including

- High bandwidth
- Low CPU utilization due to the use of Remote Direct Memory Access (RDMA)
- Single network connection (multiple protocols can share an IB link)

As a transport for the iSCSI protocol, iSER also provides all the advantages of iSCSI technology including

- iSCSI naming services
- Centralized management via iSNS

Solaris has existing iSCSI initiator and iSCSI target implementations. These implementations are socket-based and unfortunately do not include provisions to accommodate additional modes of transport.

As part of the iSER project, a newly proposed iSCSI Data Mover (IDM) kernel module provides a modular API to the iSCSI layer (both initiator and target) that abstracts the low-level details of the transport layer. A driver that implements these low-level routines and registers them with the IDM is referred to as an “IDM transport”. This document describes a new driver which implements iSER in Solaris as an IDM transport.

1.2 iSER Overview

iSER is defined by IETF RFC 5046 (iSCSI Extensions for RDMA). The abstract of this document states that

“[iSER] provides the RDMA data transfer capability to iSCSI by layering iSCSI on top of an RDMA-Capable Protocol, such as the iWARP protocol suite. An RDMA-Capable Protocol provides RDMA Read and Write services, which enable data to be transferred directly into SCSI I/O Buffers without intermediate data copies.”

This document describes a Datamover Architecture (later refined by IETF RFC 5047 (Datamover Architecture)) and a related Datamover Interface, which provide iSCSI with access to a Datamover Protocol, of which iSER is one. The Solaris IDM kernel module seeks to implement a generic Datamover Interface for the Solaris implementation of iSER, and for any future Datamover protocols. Thus, iSER and IDM development will be tightly coupled.

In iSER, data is transferred between the initiator node and the target node via RDMA services, which allows the software to write into and read from SCSI buffers directly over the RDMA capable protocol. This should vastly improve data throughput while at the same time lessen the overall impact upon system utilization.

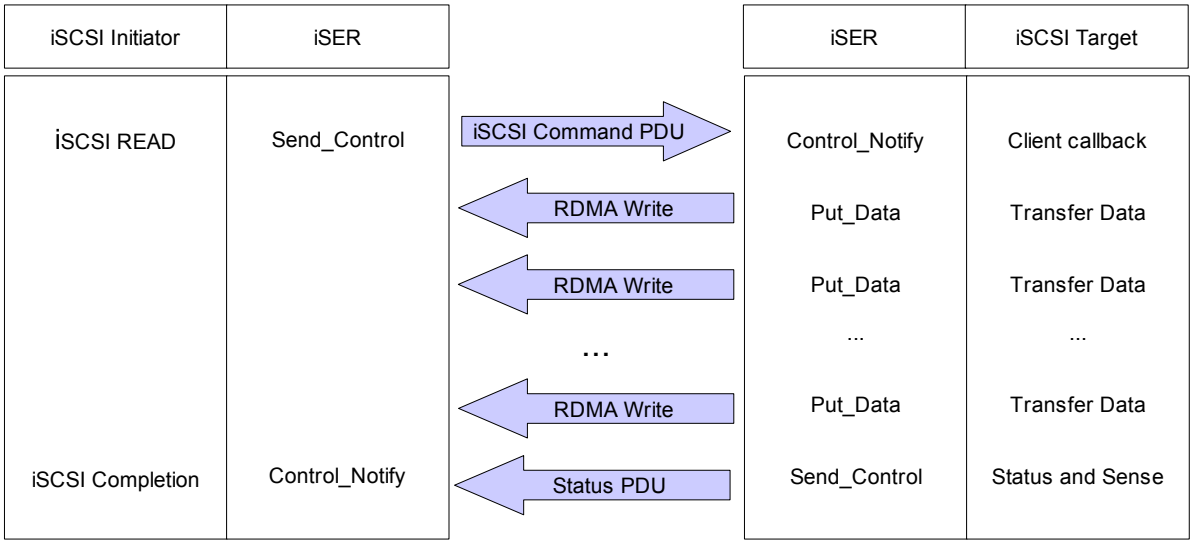
iSER data transfers are all requested by the initiator and are all driven by the target, with the exception of unsolicited (or immediate) data. When an iSCSI read command is sent to the target, the target generates a

series of RDMA writes from its local buffers to the initiator's registered memory to fulfill the read request. Likewise, when an iSCSI write command is sent to the target, the target generates a series of RDMA reads from the initiator's registered memory into local buffers to fulfill the write request.

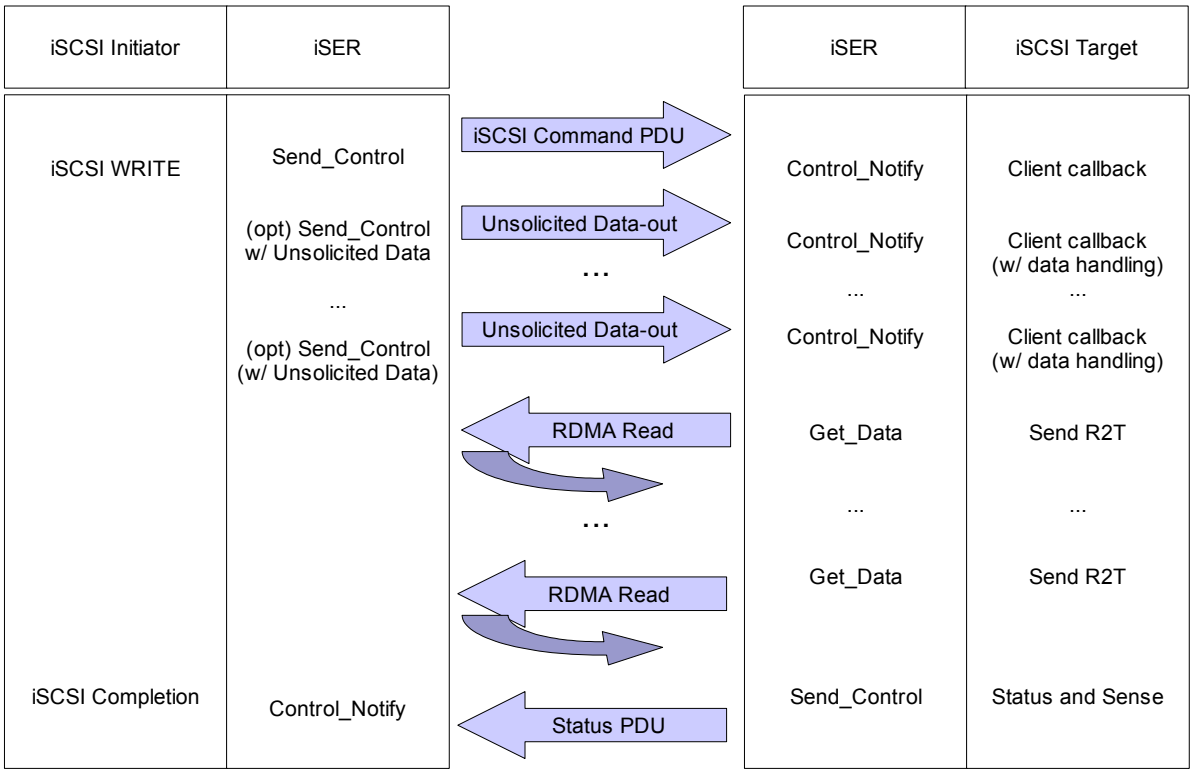
iSCSI moves commands and data between the nodes with Protocol Data Units (PDUs). In IDM, there are two types of PDUs: Control-type PDUs and Data-Type PDUs. Roughly speaking, a Control-type PDU contains a command (or possibly a write command with unsolicited / immediate data), and a Data-type PDU contains data. iSCSI will invoke Control-type PDUs, but Data-type PDUs are the domain of the Datamover.

At a high level, when the Solaris iSCSI initiator needs to execute a read or write operation, it will invoke the IDM layer's Send_Control to issue the read or write command to the iSCSI target. This primitive will provide a mechanism for providing the IDM layer with the required data and parameters for the IO operation. On the iSCSI target node, the iSCSI layer will invoke the IDM layer's Put_Data or Get_Data primitive (depending on whether it is a read or write operation, respectively). It will then issue the status of the iSCSI operation with the Send_Control primitive.

Below is a diagram that describes an iSCSI read operation in iSER-assisted mode, followed by a diagram that shows an iSCSI write operation in iSER-assisted mode.



An iSCSI read operation utilizing iSER-assisted mode.



An iSCSI write operation utilizing iSER-assisted mode.

Note that these diagrams attempt to convey the high-level principles involved in the operations.

1.3 Design Goals

- iSER will abstract from the IDM the allocation and management of all transport-specific (i.e. InfiniBand) resources
- iSER will abstract from the IDM all iSER layer functionality implemented to support the IDM transport routines (DA primitives)
- iSER will implement and register all required IDM transport-layer routines
- iSER will implement and register all required IBTF event and object handling routines
- iSER will not require any iSCSI layer state knowledge, other than opaque object handles from the IDM layer, to operate successfully
- iSER will be utilized as a default transport when available without action from the end user.

1.4 Definitions

iSCSI Initiator	A virtual HBA driver that implements the iSCSI initiator protocol and presents itself as a SCSA HBA driver.

iSCSI Target	STMF port provider implementing the iSCSI target protocol
STMF	SCSI Target Mode Framework provides the interface between SCSI lun providers and SCSI port providers on the target side. (COMSTAR)
iSCSI Datamover Architecture	Defined in IETF RFC 5047 (DA), and IETF RFC 5046 (iSER). Specification for an abstracted Datamover Protocol (e.g. iSER) and a Datamover Interface (e.g. IDM).
iSCSI Datamover (IDM)	Solaris kernel module that implements iSCSI the iSCSI Datamover Interface for iSCSI clients.
IDM Default Transport / IDM Native Transport	IDM native transport functionality for sockets over TCP/IP, implemented as part of the Solaris IDM kernel module.
iSER	“iSCSI Extensions for RDMA “ described in IETF RFC 5046. This document describes an implementation of iSER as a Solaris IDM transport module.
iSER-assisted mode	On the leading iSCSI login, once a session has negotiated iSER and moved the connection to using iSER, the connection is referred to as being in iSER-assisted mode.
RCaP	RDMA Capable Protocol
InfiniBand RC	Reliable Connected transport provided by InfiniBand ; an RCaP
RDMA Write	A method by which the iSCSI Target node can transfer data from local buffers into buffers on the iSCSI Initiator node. This is used to achieve an iSCSI Read operation while in iSER-assisted mode.
RDMA Read	A method by which the iSCSI Target node can transfer data from the iSCSI Initiator node into local buffers This is used to achieve an iSCSI Write operation while in iSER-assisted mode.
RC Send	This refers to a Send IB message type over the RC transport. This is used to send Control-type PDUs while in iSER-assisted mode, and can also be used to send immediate data when appropriate.
IBTF	InfiniBand Transport Framework, the Solaris InfiniBand transport stack
IBTI	InfiniBand Transport Interface, the client driver IBTF programming interface
IB HCA	InfiniBand Host Channel Adapter is the host-side fabric interconnect
IPoIB	IP transport over InfiniBand fabric. iSER connections begin as TCP/IP connections using IPoIB and the IDM Native Transport and are later transitioned to iSER mode.

1.5 Project Goals

It is the overall goal of the iSER project to deliver into the OpenSolaris community and the Solaris Operating Environment a full iSER implementation, including both initiator and target, for use with the InfiniBand RC transport. The driver will be compliant in all possible ways with IETF RFCs 5046 and 5047, unless compliance is found to introduce instability, decrease or remove functionality or otherwise deprecate the quality of the driver.

1.6 Project Non-Goals

There are no specific non-goals of the project at this time.

1.7 Dependencies

The Solaris iSER implementation has dependencies upon the following Solaris software components, along with standard kernel services and the Solaris Device Driver Interface (DDI)

- Solaris InfiniBand Transport Interface (IBTI) and underlying HCA driver support
- Solaris iSCSI Datamover (IDM)

1.8 References

Internet Small Computer Systems Interface (iSCSI)

Internet Small Computer System Interface (iSCSI) Extensions for Remote Direct Memory Access (RDMA)

DA: Datamover Architecture for Internal Small Computer System Interface

InfiniBand Architecture Specification Volume 1, Release 1.2.1

InfiniBand 1.2 Specification Annex 11: RDMA IP CM Service

InfiniBand 1.2 Specification Annex 12: Support for iSCSI Extensions for RDMA

OpenSolaris InfiniBand Developer Documentation

2 Architecture

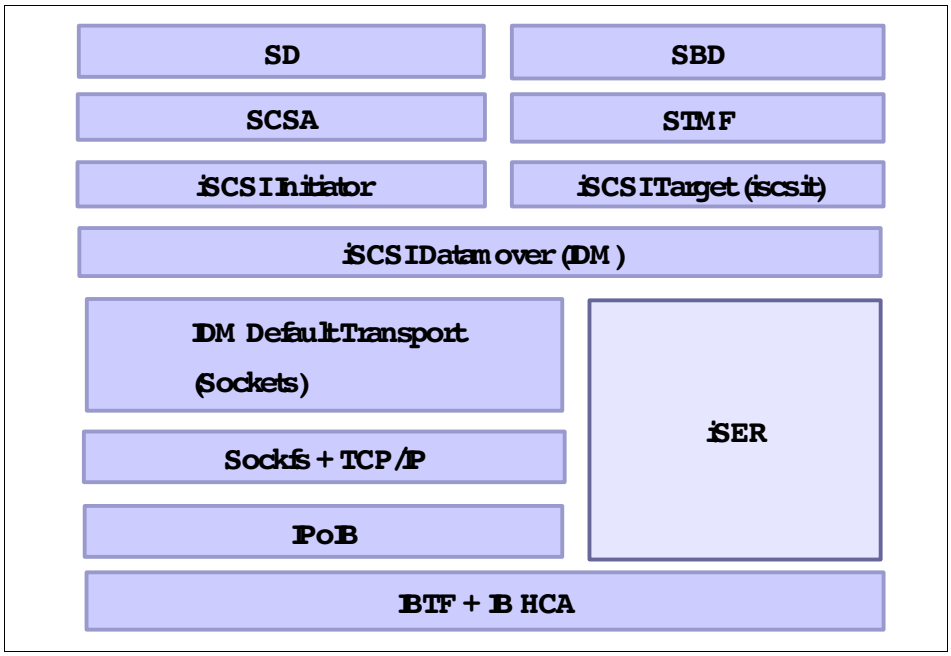
2.1 Overview

iSCSI Extensions for RDMA (iSER) is a Solaris kernel driver that implements an iSCSI Datamover (IDM) transport module, providing IDM with iSER over InfiniBand-specific implementations of each of the transport layer primitives. These primitives provide IDM, and thus iSCSI indirectly, with the following functionality

- RDMA capable protocol (RCaP) channel establishment (specifically, InfiniBand Reliable Connected)
- Allocation and management of RCaP-specific resources
- Transport-related event notification
- Transfer of iSCSI data payload (iSCSI Data-type PDUs) via the RCaP channel (RDMA Read and Write)
- Exchange of iSCSI commands (iSCSI Control-type PDUs) via a secondary channel (IB Send)

In order for iSER to provide IDM with access to the Solaris InfiniBand stack, iSER will be implemented as a Solaris pseudo device driver, parented by the “ib” node. This will facilitate use of the IBTI and give the driver IB event notification and It is also required that iSER be linked during compile time to the IBTL and the IBCM modules, providing transport layer and communications management functionality, respectively. The IBTL provides an IB transport layer abstraction for IB client drivers, namely the IBTI. The IBCM provides an abstraction of the RC channel services for IB client drivers; iSER will need this to establish and utilize RC channels.

The iSER driver layers between the IDM and the IBTF as indicated in the diagram to the right. Note that both IDM native mode (left path) and iSER-assisted mode (right path) are detailed. All iSCSI sessions must establish initially via the IDM native Sockets transport, thus in iSER configuration we utilize the IPoIB protocol.



2.2 Datamover Primitives and the IDM Transport API

The Solaris iSER driver is based upon the IETF RFC 5046 (iSCSI Extensions for RDMA). This RFC refers to the Datamover Architecture detailed in IETF RFC 5047 (Datamover Architecture) and defines a set of Datamover Primitives. The Datamover Architecture seeks to abstract the iSCSI (client) layer from the iSER (transport) layer, and can be implemented in such as way as to provide support for multiple Datamover Protocols (such as iSER).

The Solaris IDM kernel module requires that an IDM transport driver implement transport-specific implementations of the following Datamover primitives. These routines are then referenced by the idm_transport_impl_ops_t passed into IDM via idm_register(). In this way, the IDM layer can indirectly invoke transport-specific DA primitive operations, which facilitates future DA protocol development and improves modularity in the IO stack. See the IDM design document for further information. Each of the following primitives will be discussed in detail in Section 5, Solaris iSCSI Datamover (IDM) Functionality.

Datamover Primitive	Description from RFC5047
Send_Control	Request the outbound transfer of an iSCSI control-type PDU
Put_Data	Request the outbound transfer of data for a SCSI Data-in PDU
Get_Data	Request the inbound transfer of solicited data requested by an R2T PDU
Allocate_Connection_Resources	Request the allocation of all transport-specific connection resources
Deallocate_Connection_Resources	Request the deallocation of all transport-specific connection resources
Enable_Datamover	Request that a specific iSCSI connection be transitioned to Datamover-assisted mode
Connection_Terminate	Request that a specific connection be terminated and all associated connection resources be freed
Notice_Key_Values	Request that specific key-value pairs be noted
Deallocate_Task_Resources	Request the deallocation of all task resources

2.3 InfiniBand configuration and RC channel use

The RCaP for the first version of Solaris iSER will be the InfiniBand Reliable Connected transport. As mentioned previously, the iSER driver will be implemented as a pseudo driver, parented by “ib”. The “ib” module is the InfiniBand nexus in Solaris, and provides all access to the IBTF (via the IBTI) to client drivers.

When a new session is established in iSCSI, IDM will attempt to detect whether or not the native connection is over an RCaP link and, if so, able to provide an RCaP functionality (e.g. IPoIB, providing iSER capability). Once established, the session will move to iSER-assisted mode through a set of well-defined operations and related state transitions in IDM. Two of these operations that are specific to IB and RC usage are Allocate_Connection_Resources and Enable_Datamover. Depending upon whether the running iSER instance is on an iSCSI Initiator or iSCSI Target node, iSER will perform the necessary operations required to

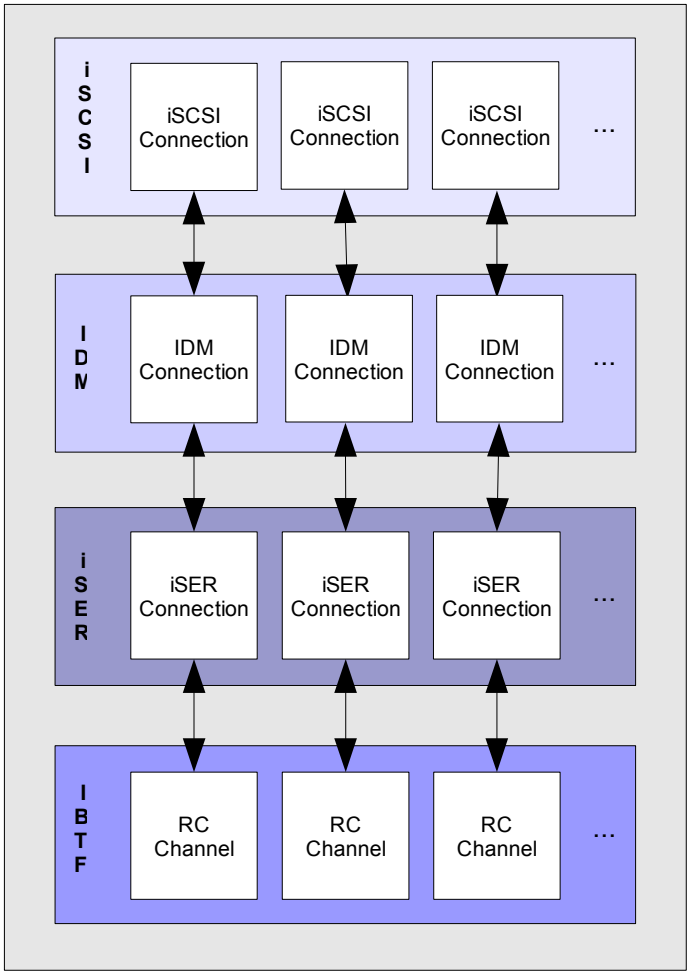
provide the connection allocation and establishment for IDM. Once these operations are complete, the connection is transitioned to iSER-assisted mode. Note that this transition is default behaviora and requires no user intervention or action. For more information on how iSER specifically implements these routines and related functionality, see Section 5.

Once the leading login on a leading connection for a session establishes iSER-assisted mode, all subsequent connections within that session must utilize iSER-assisted mode. The iSER driver will facilitate this, via IDM, for iSCSI by establishing a new RC channel for each IDM connection. Further, each iSER connection will utilize the RC channel for not only its RDMA capabilities, but also to utilize the Send operative for Control-type PDUs. Implicitly, this means that once iSER-assisted mode is selected for a session, the native IDM Sockets transport will no longer be used, and iSER is responsible for all session communication.

The relationship between iSCSI, IDM, iSER and IBTF within a single iSER-assisted iSCSI session is shown in the diagram to the right. Note that each iSCSI connection has its own IDM connection, which provides a dedicated transport layer channel.

For clarity's sake, note that in the current Solaris implementation, iSCSI utilizes its own Sockets code to perform PDU and Data transfers. With the new Datamover in place, iSCSI will utilize the IDM APIs not only for iSER functionality, but for all transport, including the native Sockets transport.

In an iSCSI session that is in iSER-assisted mode, the layering to the right is representative of the call layers on one of the nodes (initiator or target). iSCSI will allocate IDM handles and will invoke IDM API routines. In turn, IDM will invoke routines in the iSER layer via its own IDM transport functionality. iSER will then invoke the necessary routines to transport the iSCSI commands and data via the InfiniBand RC channel.



3 Driver configuration and access

The Solaris iSER driver will be implemented as a pseudo device driver. The Solaris Device Driver Interface (DDI) provides a set of standard auto-configuration entry points for drivers to implement which are called by kernel services for configuration.

3.1 Initialization

3.1.1 _init()

Since iSER will be implemented as a child node of the InfiniBand nexus, it is required that it only load and configure if there is InfiniBand HCA hardware present. During module initialization, iSER will invoke the IBTI routine ibt_hw_is_present() to determine whether or not IB hardware is available for use.

The IDM layer will utilize the Solaris Layered Driver Interface (LDI) to open and access the iSER driver (see section 3.3.1). Note that this provides the iSCSI stack the opportunity to utilize iSER if InfiniBand hardware is not present at boot, but is hot-plugged later.

Once it has been established that InfiniBand hardware is present, configuration can proceed.

3.1.2 attach()

The iSER attach() routine will be invoked once init() has completed successfully. It will continue the process of initializing the iSER driver for service, and will allocate and initialize the driver's soft state structure, shown below:

```
typedef struct iser_state_s {
        dev_info_t      *is_dip;
        int             is_instance;
        kmutex_t        is_refcnt_lock;   /* IDM open reference count lock */
        int             is_open_refcnt;   /* IDM open reference count */
        ibt_clnt_hdl_t  is_ibhdl;         /* IBT handle */
        ibt_srv_hdl_t   is_srvhdl;        /* iSER service handle */
        iser_hca_t      *is_hcalist;      /* list of HCAs */
        uint_t          is_num_hcas;
        iser_conn_t     is_connlist;
        iser_config_t   is_cfg;           /* configuration profile */
} iser_state_t;
```

iSER must connect to the IBTF via ibt_attach(). This IBTI routine initializes an ibt_clnt_hdl_t, which is then stored in is_ibhdl in the iSER state. Once attached, iSER must then retrieve the current list of HCAs and store a handle for each in the is_hcalist. Part of the initialization of the HCA list will be the allocation of a PDU per HCA, and the creation of data and message kmem_cache's for each HCA. See section 4.2 for information related to the use of these caches.

Additionally, iser_attach() will invoke the idm_register() routine, passing iSER's operations vector for its implementation of the IDM transport operations (idm_transport_ops_t) up to the IDM kernel module. Access to the iSER driver from IDM will be provided via LDI. IDM retains a list of well-known transport drivers and accesses them through LDI handles returned by ldi_open_by_name(9F).When IDM wishes to utilize iSER as a transport, it will do so via the idm_transport_ops_t structure that is registered by iSER via this invocation of the idm_register() routine. See Section 5 for details on the implementation of the routines referenced in this structure.

3.2 Teardown

3.2.1 detach()

For its part, detach() undoes what has been done in attach(). idm_unregister() will be invoked to inform the IDM layer that this transport is no longer in place. The iSER service needs to be unbound and unregistered, via the ibt_unbind_service() and ibt_deregister_service() routines.

3.2.2 _fini()

The standard DDI fini operation will be invoked here (mod_remove, etc).

4 Solaris InfiniBand Transport Framework (IBTF) Usage

As stated previously, the initial iSER implementation for Solaris described in this document will utilize the InfiniBand Reliable Connected service for transmission of both Control-type and Data-type PDUs. The RC Send operation will be used to transmit Control-type PDUs from the Initiator and Target, and the RDMA operations are used from the Target to transmit data to and from the Initiator.

4.1 Provisioning for an iSER connection

In the IDM-enabled iSCSI implementation, the state transitions act upon iSCSI connections. Multiple connections may make up a session, but sessions are only referenced in the upper layers. Therefore, we use the IDM connection as the single handle for all related work between two nodes.

A configuration profile will be used within iSER which will contain, among other things, queue depths, work request size, and the various settings for the RC channel and related queues. This will be very useful during bring-up, but will also provide for a patchable configuration mechanism which can be used for performance or other tuning. The structure and exact contents of this profile will be defined during the development cycle.

Each IDM connection that utilizes iSER will be allocated its own RC channel with its own set of work queues: Send queue (SQ), Receive queue (RQ) and related Completion Queues (CQs).

4.2 Message and data caches

To alleviate some of the latency induced by memory registration operations, iSER will make use of kmem caches for both message and data buffer software structures. Embedded in these structures will be the memory required for the message, plus the IBT memory registration handle. Upon initialization of these objects in the kmem_cache_constructor routine, the memory will be allocated and registered with the IBTF. Once used, the implementation will free it back to the cache, but will not unregister it. Thus, after a period of cache warming, we will be taking advantage of pre-registered memory for IB messages.

Note we can utilize this design on the initiator for Send messages only, as buffers are sent into the iSER layer from the upper layers. However, on the target node, we are required to provide the upper layers with buffers. Therefore, we can take advantage of this caching approach for data buffers.

4.2.1 Message objects

On both Initiator and Target nodes, message objects will be allocated from an HCA's message cache. These message objects will contain two memory segments, registered during cache object construction. One will be used for the message, and the second can be used for payload (text messages, etc).

4.2.2 Data buffer objects

On Target nodes, data buffers will be allocated from an HCA's data buffer cache. These data buffers objects will contain a buffer which will be sized to the default COMSTAR buffer size (currently 128k).

4.3 Channel allocation and RC connection establishment

Once basic configuration options are set, iSER then must allocate the handle for the connection, via ibt_allloc_rc_channel(), and open the channel, via ibt_open_rc_channel(). These routines will occur at different points during the RC channel connection establishment, based upon whether or not the code is executing on an Initiator node or on a Target node. Note that this is unrelated to the iSER Hello exchange or the Datamover login processes.

4.4 IBT handlers

The iSER connection state will be driven from two event generation points, as well as via upper layer direction. The two main event sourcese are the IBT Asynchronous Events and the IBT CM Events.

4.4.1 IBT Asynchronous Events

As an IBTF client driver, iSER will implement an IBT Asynchronous Event handler and register it with the framework as part of its IBT client module info (ibt_clnt_modinfo_s), which is passed via the ibt_attach() routine.

The following asynchronous events will be handled by iSER.

IBT_EVENT_PORT_UP

This event indicates that a port on an HCA already known to the client has come up from a down state. The HCA port will be queried for any changes, and then the iSER service will be bound on the port.

IBT_ERROR_PORT_DOWN

This event indicates that a port on a known HCA has gone down. How this is ultimately handled will depend upon how IDM and iSCSI handle connection loss or how they have configured the transport to handle such an event. The initial implementation will quiesce all activity bound for the port for a given amount of time, then will tear down the connection.

IBT_HCA_ATTACH_EVENT

This event indicates a new HCA has been added to the node, probably via hot-plug. iSER will query this new HCA and bind service on all available ports.

IBT_HCA_DETACH_EVENT

This event indicates that an HCA known to iSER has requested that clients cease using it, possibly for a hot-plug operation. iSER will quiesce all traffic, unbind service on all the HCA ports, and remove it from its active list.

4.4.2 IBT CM Events

The following Communication Manager events will be handled by routines that are invoked from the registered CM Event handler (ibt_cm_event_handler_t), which is passed via ibt_register_service().

The IBTF CM event model provides clients with a private data field which can be passed in the ibt_open_rc_channel() 'args' argument, or can be set and retrieved via ibt_set_chan_private() and ibt_get_chan_private(), respectively. iSER channel-private data will utilize the following structure.

The following CM events will be handled by iSER.

IBT_CM_EVENT_REQ_RCV

This event indicates an inbound request for a new connection. This event will be generated on the Target node when the Initiator node invokes the Allocate_Connection_Resources Datamover primitive. This will not be generated in the context of an Initiator node.

The Target node will read in the private data from the event, which will provide the IP information used to lookup the IB path via ibt_get_ip_paths(). iSER will then allocate an RC channel handle, and can bind the channel private data via ibt_set_chan_private().

A set of Receive work requests must be posted on the RQ for this channel in anticipation of inbound traffic. iSER will then return IBT_CM_ACCEPT.

iSER will post an event to the IDM connection state machine on the Target Node indicating that the iSER-enabled connection has been initialized.

IBT_CM_EVENT_REP_RCV

This event indicates a reply from a previous request. This event will be generated on the Initiator node after it has invoked ibt_open_rc_channel() during Allocate_Connection_Resources. This will not be generated in the context of a Target node.

The Initiator will read in the private data from the event, which will contain a copy of the channel handle. The channel handle in turn can be used to look up the channel private date via ibt_get_chan_private().

A set of Receive work requests must be posted on the RQ for this channel in anticipation of inbound traffic. iSER will then return IBT_CM_ACCEPT.

iSER will post an event to the IDM connection state machine on the Initiator Node indicating that the iSER-enabled connection has been initialized.

IBT_CM_EVENT_MRA_RCV

This event provides an acknowledgement that the peer node received a message, and may need additional

time to process the request. The channel handle and private data will be retrieved from the event, providing the iser_conn_t handle. If timing issues are discovered during development bring-up, this event can be used to set flags or otherwise indicate that the message has been received, so as to avoid a timeout condition.

IBT_CM_EVENT_CONN_EST

This indicates that the connection is established between the two endpoints, and that the client can now utilize the channel.

iSER will post an event to the IDM connection state machine on the indicating that the iSER-enabled connection is complete.

IBT_CM_EVENT_CONN_CLOSED

This event indicates that the connection has been closed. iSER should read in the channel handle and private data and establish whether or not this is a protocol error or if we are in logout.

iSER will post an event to the IDM connection state machine on the indicating that the iSER-enabled

connection is closed.

IBT_CM_EVENT_FAILURE

This event is delivered when the CM connection establishment fails. The channel handle and private data will be read in and the connection state verified.

iSER will post an event to the IDM connection state machine on the indicating that the iSER-enabled connection is failed.

4.5 IO operations

As previously discussed, iSCSI traffic can be partitioned into two types of PDU (protocol data unit): Control-type PDUs and Data-type PDUs. Commands and status messages are carried in Control-type PDUs and will be transmitted in iSER using the RC Send operation, with mesage objects allocated from a given Message cache. These may also include PDUs with inline data. All other data transmission will be done via RDMA operations, via the RDMA Read and RDMA Write operations. Below, these three basic IO operations are described. The operations described below are the basic IO building blocks of the iSER driver. For detailed information regarding iSER messaging and related topics, see Section 5 and the iSER RFC 5046.

4.5.1 Send message

Any Control-type PDU transmissions will result in iSER generating an RC Send work request, with the content being the PDU and related information. A PDU that does not invoke data transmission will be passed through without intervention. When handling a PDU that will result in data transmission, iSER must identify the appropriate memory key and write it into the iSER header portion of the message. On the Target node, Send will only be used to transmit Status PDUs back to the Initiator node.

4.5.2 RDMA Write

The RDMA Write operation is invoked on the Target node to transfer data from a Target node buffer to a pre-registered buffer (tagged buffer) on the Initiator node. Memory key, offset, data length and other information regarding the data transfer will have been received in a previous Control-type PDU. One or more RDMA Write operations implement the SCSI Read command's data transfer, transferring data from the Target buffers into the Initiator buffers. The Initiator node does not invoke the RDMA Write operation.

4.5.3 RDMA Read

The RDMA Read operation is invoked on the Target node to transfer data from a pre-registered buffer (tagged buffer) on the Initiator node to a Target node buffer. Memory key, offset, data length and other information regarding the data transfer will have been received in a previous Control-type PDU. One or more RDMA Read operations implement the SCSI Write command's data transfer, transferring data from the Initiator buffers into the Target buffers. The Initiator node does not invoke the RDMA Read operation.

5 Solaris iSCSI Datamover (IDM) Functionality

5.1 IDM registration

IDM registration in iSER is handled during iser_attach(). See Section 3.1.2 of this document for further information.

5.2 IDM Transport implementation

iSER implements the following IDM transport operations.

5.2.1 transport_conn_is_capable

```
idm_status_t (*transport_conn_is_capable)(idm_conn_t *ic,
      idm_transport_caps_t *caps);
```

This routine is invoked by IDM to determine if the connection passed in is provided for by this transport. iSER will extract the IP address from the idm_conn_t and will invoke ibt_get_ip_paths() to determine if the IP address passed in is reachable over any of the HCAs currently known by iSER. If not, a status of IDM_STATUS_FAIL is returned, indicating that this connection is not iSER-capable. In the case that the connection is found to be provided via this transport, this routine will return IDM_STATUS_SUCCESS.

5.2.2 transport_configure_service

```
idm_status_t (*transport_configure_service)(idm_conn_t *ic);
```

This routine is invoked by IDM to establish the required transport services on a Target node. In this routine, iSER will register its Communications Manager (CM) service and bind it on all available ports. These two steps are explained below.

First, iSER will register itself as a service provider via ibt_register_service(). This establishes iSER as a service provider and returns a service identification handle (ibt_srv_hdl_t), which is also stored in the state (is_srvhdl).. Among the parameters passed in are an ibt_cm_handler_t, which encodes a callback routine to handle RC CM events (see Section 4.1 below), and a Service ID (SID), which is used to identify the service. The RDMA-aware SID format is defined by IB Specification Annex 11, and the specific iSER SID is defined by IB Specification Annex 12, and is as follows:

Byte Location	Description	Value
0	IBTA AGN	0x0
1 to 3	Prefix of RDMA-aware SID	each byte 0x0
4	RDMA-aware ULP SID	0x1
5	Protocol	0x6 (TCP)

6 o 7	Destination Port Number	Set by iSCSI / iSER (usually the iSCSI well-known TCP port number)

Given the above values, and the iSCSI well-known port number of 3260, we derive a default SID of 0x00000000011060CBC. However, since the port number can be altered depending upon system or socket connection configuration, iSER will utilize a simple macro to establish this value based upon a port number (stored in the idm_conn_t, along with the IP address).

Once iSER has registered its service with the IBTF, it can now move to the second phase of binding the service to each of the available ports via ibt_bind_service(). This routine is invoked upon each port found on each HCA that has been opened by iSER. This routine requests that the CM accept service requests for the service ID on the port. Also, it allows the SA (Subnet Administration) to advertise the service.

The core functionality in this routine will be in a secondary routine which can also be invoked upon detection of an IBT_HCA_ATTACH_EVENT.

5.2.3 transport_notice_key_values

```
idm_status_t (*transport_notice_key_values)(idm_conn_t *ic,
      nv_list_t *request_nvl, nv_list_t *response_nvl,
      nv_list_t *negotiated_nvl);
```

IDM invokes this routine to request that the transport layer review and make determinations upon the current set of key-value pairs for use during iSCSI login negotiation. iSER will review each of the key value pairs and accept the current value by placing it on the negotiated_nvl, reject it by leaving it off of the negotiated_nvl, or alter its value and place it on the negotiated_nvl and response_nvl. The following is an initial set of key value pairs that are of interest to iSER.

RDMAExtensions

This should already be set to “Yes”.

HeaderDigest and **DataDigest**

Both of these should be set to “None”.

MaxRecvDataSegmentLength

On the Initiator node, the value of InitiatorRecvDataSegmentLength is used to set this value.

On the Target node, the value of TargetRecvDataSegmentLength is used to set this value.

TargetRecvDataSegmentLength

Set to maximum data segment size on a Target-bound Control-type PDU. Value is TBD.

InitiatorRecvDataSegmentLength

Set to maximum data segment size on a Target-bound Control-type PDU. Value is TBD.

OFMarker and **IFMarker**

Both of these should be set to “No”.

MaxOutstandingUnexpectedPDUs

Set to the number of outstanding unexpected Control-type PDUs. Value is TBD.

5.2.4 transport_alloc_conn_rsrcs

```
idm_status_t (*transport_alloc_conn_rsrc)(idm_conn_t *ic);
```

This routine is invoked to allocate all required transport-specific connection resources for the provided connection.

On the initiator, this involves allocating and opening an RC channel handle, and also allocating related resource such as CQ and other IBTF resources. The RC channel open on the Initiator node should cause a CM connection request event on the Target node, which will result in an RC connection establishment between the two nodes.

On the Target node, since this routine is called after the RC connection is allocated and established in response to the inbound CM request from the initiator, this routine will just validate that the connection is indeed in place and in good standing before returning success.

5.2.5 transport_free_conn_rsrcs

```
idm_status_t (*transport_free_conn_rsrc)(idm_conn_t *ic);
```

This routine is invoked to free the transport-specific connection resources for the provided connection. This routine will invoke ibt_close_rc_channel() and ibt_free_channel(), as well as free up other related resources if necessary.

5.2.6 transport_enable_datamover

```
idm_status_t (*transport_enable_datamover)(idm_conn_t *ic,
      idm_pdu_t final_login_rsp);
```

This is the routine that shifts the active transport operations for the connection from the previous transport (presumably the native sockets transport) to the transport it is invoked upon. In iSER's case, it will move the active transport handle to the ops vector passed in iSER's invocation of idm_register(), and will deprecate the native sockets code.

The final step in Enable_Datamover is dependent upon which side of the iSCSI connection it is executing on. On both nodes, the successful invocation of this routine indicates that all further communication will be iSER-assisted, and the Sockets transport is no longer needed.

On the Target node, the final Login Response PDU of the iSCSI Login negotiation phase is sent on the Sockets transport. This is passed in by the iSCSI layer in the `final_login_rsp` argument. The Sockets transport may then be torn down, and the iSER transport handle put in its place.

On the Initiator node, Enable_Datamover is invoked upon the receipt of the final Login Response PDU from the Target node (sent in its Enable_Datamover routine). The routine will tear down the Sockets transport and put the iSER transport handle in its place. Then, prior to returning success, the Initiator node will initiate the iSER Hello Exchange, which must be the first message exchange in the newly established iSER-assisted session.

5.2.7 transport_conn_terminate

```
idm_status_t (*transport_conn_terminate)(idm_conn_t *ic);
```

This routine directs iSER to quiesce all traffic, free up all task and connection resources, and to close the RC channel connection and free it. It is a single access point for a clean shutdown of the iSER-assisted connection. All channel resources, including related queues, will be freed and all memory resources will be invalidated and unmapped prior to returning success from this routine.

5.2.8 transport_tx

```
void (*transport_tx)(idm_conn_t *ic, idm_pdu_t *pdu);
```

This routine will direct iSER to place the passed PDU in an RC Send message and transmit it to the remote node indicated in the connection. iSER

5.2.9 transport_buf_tx_to_ini

```
idm_status_t (*transport_buf_tx_to_ini)(idm_task_t *idt,
    idm_buf_t *idb);
```

IDM will invoke this routine on the Target Node to initiate data transfer via one or more RDMA Write operations after receiving a command PDU requesting a SCSI Read. A task handle `idt` is passed into the transport layer via this routine, which houses all of the required contexts and handles for requesting a data transfer start, including the connection context and affiliated buffers.

5.2.10 transport_buf_rx_from_ini

```
idm_status_t (*transport_buf_rx_from_ini)(idm_task_t *idt,
    idm_buf_t *idb);
```

IDM will invoke this routine on the Target Node to initiate data transfer via one or more RDMA Read operations after receiving a command PDU requesting a SCSI Write. A task handle `idt` is passed into the transport layer via this routine, which houses all of the required contexts and handles for requesting a data transfer start, including the connection context and affiliated buffers.

5.2.11 transport_rx_datain

```
void (*transport_rx_datain)(idm_conn_t *ic, idm_pdu_t *pdu);
```

This routine is invoked on the Initiator node to handle receipt of inbound Data-type PDUs.

5.2.12 transport_rx_rtt

```
void (*transport_rx_rtt)(idm_conn_t *ic, idm_pdu_t *pdu);
```

This routine is invoked on the Initiator node to handle receipt of inbound R2T PDUs.

5.2.13 transport_rx_dataout

```
void (*transport_rx_dataout)(idm_conn_t *ic, idm_pdu_t *pdu);
```

The IDM layer calls this function on the target-side to handle receipt of inbound Data-out PDUs.

5.2.14 transport_free_task_rsrcs

```
idm_status_t (*transport_free_task_rsrcs)(idm_task_t *it,
      idm_conn_t *ic);
```

This routine is invoked to free the transport-specific task resources for the provided task on the connection. This routine is used when a task needs to be aborted during transmission. iSER will release all task-related resources when invoked.

5.2.15 transport_buf_alloc

```
idm_status_t (*transport_buf_alloc)(idm_buf_t *idb, int buflen);
```

This routine allocates a buffer from the associated channel's HCA data buffer cache. This routine is only invoked on target nodes.

5.2.16 transport_buf_free

```
idm_status_t (*transport_buf_free)(idm_buf_t *idb);
```

This routine frees a buffer back to the associated channel's HCA data buffer cache. Note that the buffer memory is not deregistered with the HCA, but it is invalidated for use.

5.2.17 transport_buf_setup

```
idm_status_t (*transport_buf_setup)(idm_buf_t *idb);
```

This routine provides the transport layer an opportunity to take any necessary action on a buffer prior to its use. In the case of iSER, a memory registration operation will be invoked upon the memory in the buffer so that it may be used with the InfiniBand framework. A set of flags may be required to keep track of different types of memory registration operations (ibt_register_mr(), ibt_register_buf(), ibt_register_physical_fmr()) if multiple operations are supported.

5.2.18 transport_buf_teardown

```
void (*transport_buf_teardown)(idm_buf_t *idb);
```

This routine is the opposite member of the setup routine above, allowing the transport layer to tear down any transport-specific operations on the buffer that were performed. iSER will check any flags that may be set and undo the previous memory registration operation on the buffer.

6 Design considerations

The following are design considerations which will require some research during driver bring-up and test. They are primarily related to performance, and may represent work beyond the first release of the software, or indeed ongoing work.

6.1 Channel and Memory attributes

At least initially, the iSER driver should utilize a patchable interface for internal resource sizing configuration. In this way, values can be set via mdb or the /etc/system file and it can be deteremined where the appropriate levels for the production software lie.

Attributes that should be explored include QP depth (queue size), WR size, memory registration options and memory allocation sizes. Others may be added as development progresses.

6.2 Signaled versus Unsignaled operations

When the RC channel is allocated in ibt_alloc_rc_channel(), the caller indicates whether or not every work request will generate a work request completion (IBT_ALL_SIGNALED) or if completions are generated based upon settings in each individual work request (IBT_WR_SIGNALED). In the latter case, if a work request is posted without requesting signaling, it will only generate a work request completion if it ends in failure.

If a work request completion indicates success, it is known that all previous outstanding work requests have also completed in success. Therefore, IB client drivers can deploy a method by which periodic signaled work requests can indicate the status of all previously posted work requests.

Provided the vast majority of work requests will end in success, some latency improvement may be realized by not generating a completion event for every work request, but rather only requesting a completion event at a set interval, every 32 work requests, for example.

6.3 Fast Memory Registration (Mellanox FMR)

Currently, IBTF provides support for “Mellanox FMR”, which is a Mellanox HCA-specific memory registration mechanism which provides faster memory registration operations for IBTF clients through the use remap operations on cached resources. Use of this mechanism should be explored and performance measurements taken for comparison.

6.4 RDMA write versus inline data

Keeping in mind that new RDMA operations may require the buffer to be registered prior to invoking the RDMA operation, in regards to latency impact there exists a threshold where it is more effective to copy data onto an existing (preallocated) RC Send message than to register the memory on demand. This treshold should be determined, and logic used to employ it as a cutoff for RDMA operations versus inlining data on a Send message. Based upon previous client development efforts, this research should start with 2KB, 4KB and 8KB thresholds and performance data gathered.

6.5 Additional IBTF enhancements

The two following InfiniBand software features are described in the InfiniBand Specification, version 1.2, and are not yet supported in Solaris IBTF. They are both employed in the iSER RFC, but can both be negotiated to not in use during session establishment.

Send With Invalidate

This operation piggybacks an Invalidate operation on a Send operation. Every iSER task's advertised STag is to be invalidated upon completion of the task. This operation would save iSER from posting a separate Invalidate operation per task.

Zero Based Virtual Address (ZBVA)

This feature would make the starting virtual address of all InfiniBand memory regions 0, which would simplify buffer management in iSER.