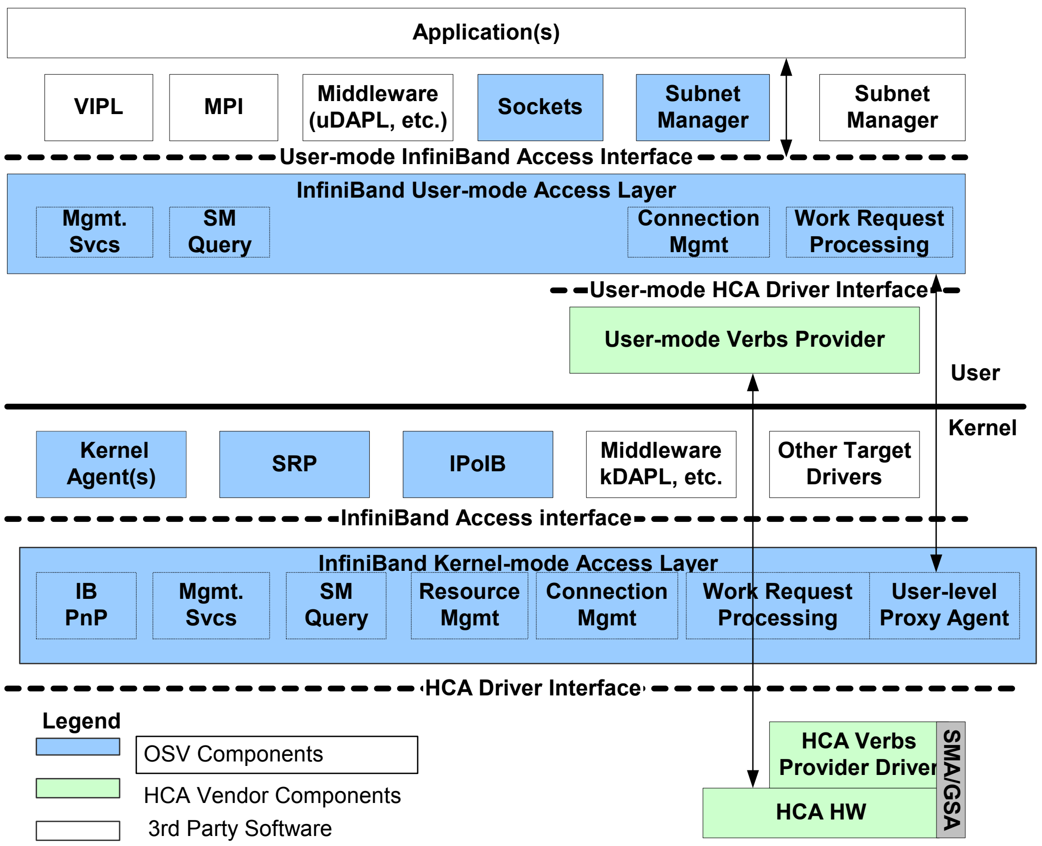
Software Architecture Specification (SAS)

Revision – Draft 2
Last Print Date: 4/19/2002 - 9:04 AM

InfiniBand™ Linux Operating System Software Access Layer

1.1 Introduction

The access layer provides transport level access to an InfiniBand™ fabric. It supplies a foundation upon which a channel driver may be built. The access layer exposes the capabilities of the InfiniBand™ architecture, and adds support for higher-level functionality required by most users of an InfiniBand™ fabric. Users define the protocols and policies used by the access layer, and the access layer implements them under the direction of a user. The diagram below presents a high level architectural overview of the InfiniBand™ Access Layer.



HCA – Host Channel Adapter provided by a HW vendor.

HW Verbs Provider Driver – This is a vendor-specific piece of software that together with the vendor's HCA forms a unit capable of implementing the verbs as specified in the InfiniBand™ specification.

HCA Driver Interface – This interface separates vendor-specific HCA code from code that is independent of any particular HCA vendor. The code that sits on top of this interface utilizes the functionality of the verbs without being dependent upon any particular vendor's implementation of the verbs.

InfiniBand Kernel-mode Access Layer– This software exports the full capabilities of the underlying HCA implementations to higher-level software. It exists to provide useful services that would otherwise be duplicated in the independent upper-level components, and to coordinate access to shared InfiniBand™ resources (such as the SMI and GSI) that are needed by multiple upper-level components. The InfiniBand Access component is envisioned to provide several services:

- **Resource Management** – This tracks HCA resource usage by upper level software components. This is useful for cleaning up those resources when/if the upper level components go away, and for having a means of tracking which upper level components should have access to which HCA resources.
- **Work Request Processing –** Process incoming work requests and dispatch the work request to the appropriate HCA drivers and handles the dispatching of work request completion notification to upper-level components. It also notifies interested upper level components of asynchronous events and errors that are detected by the vendor's HCA during its processing.
- **Connection Management** – Encapsulates the connection management protocol as defined in Chapter 12 of Volume 1 of the InfiniBand™ specification. This significantly simplifies the process of forming connections for those upper-level components that require them, and coordinates access to the GSI amongst those components.
- **SM Query** – Provides an interface to a standard set of subnet administration queries regarding the subnet configuration.
- **Management Services (SMI/GSI QP Access**)– Coordinates access amongst multiple upper-level components to the management queue pairs provided on each port of the vendor's HCA. This also routes incoming SMPs and GMPs to the appropriate upper-level components.
- **User-level Support Services (Proxy Agent)** – Interfaces to the user-mode InfiniBand™ Access Layer component to support all the services appropriate for user mode that need a kernel transition. Utilizes resources provided by both the host operating system and by the HW specific HCA verbs provider driver to accomplish this. The user-mode InfiniBand Access Layer communicates with the proxy agent through a set of IO control calls.
- **IB PnP** – The plug-and-play implementation for InfiniBand™ provided by the host operating system.

InfiniBand Access Interface – This exports the interface that all upper-level components within the kernel use to access the functionality provided by an HCA. The Proxy Agent also uses this interface to support some of the services provided to the user mode access layer.

Subnet Manager – Provides the basic Subnet Manager functionality as defined in Volume 1 of the InfiniBand™ specification.

Kernel Agents – Various kernel components provided by the OS vendor that utilize InfiniBand™.

Middleware – Transport and OS independent software layers that support InfiniBand™ and other RDMA capable transports like iWarp. Also provide an API that is portable across operating systems.

IB Target Drivers – Provides access to TCAs of various types. One example would be a driver provided by the host operating system that utilizes the SCSI RDMA Protocol (SRP) running on top of InfiniBand™ hardware to access InfiniBand™-attached storage. Another example would be a network driver that implements Internet Protocol (IP) over InfiniBand (IPoIB).

User-mode HCA Verbs Provider - Vendor-specific software in user-mode to assist with direct user mode IO (OS bypass) for data transfer operations (DTO).

User-mode HCA Driver Interface - This interface separates user-mode vendor-specific HCA code from code that is independent of any particular HCA vendor.

User-mode Access Layer – This modules exports capabilities of all the underlying HCAs that assist in developing upper layer protocols, like VIPL, uDAPL, SM, HCA Diagnostics etc., providing access to InfiniBand™ pimitives in user-mode. The user-mode InfiniBand Access Layer is a vendor independent shared library that can be dynamically loaded by the higher-level software components.

User-mode InfiniBand Access Interface – This is the interface that all applications running in user-mode use to access the underlying HCA. It attempts to minimize the level of abstraction of the underlying hardware while simultaneously being independent of the implementation of any particular vendor's HCA. It also provides the facilities to allow multiple applications to share access the HCAs.

Sockets, VIPL, MPI, and other messaging interfaces – user-mode implementations of all of these messaging interfaces can be built using the User-mode InfiniBand Access Interface.

> **Applications** – Applications that wish to live "close to the metal" in order to fully exploit the capabilities of the underlying InfiniBand™ hardware and are willing to use a lower-level interface for optimum performance can access the User-mode InfiniBand Access Interface directly

1.2 Design Methodology

The access layer architecture is constructed using bottom-up, object-oriented design techniques with top-down functional requirements. Based on user requirements, specific functionality exposed by the access layer is captured. Commonality between the different functional areas is then identified, and generic components are constructed to support the common areas. Generic components are created using standard library modules where possible. To support user-specific API requirements, detailed modules are layered over the generic components. This is shown in the Figure 1-1 below.

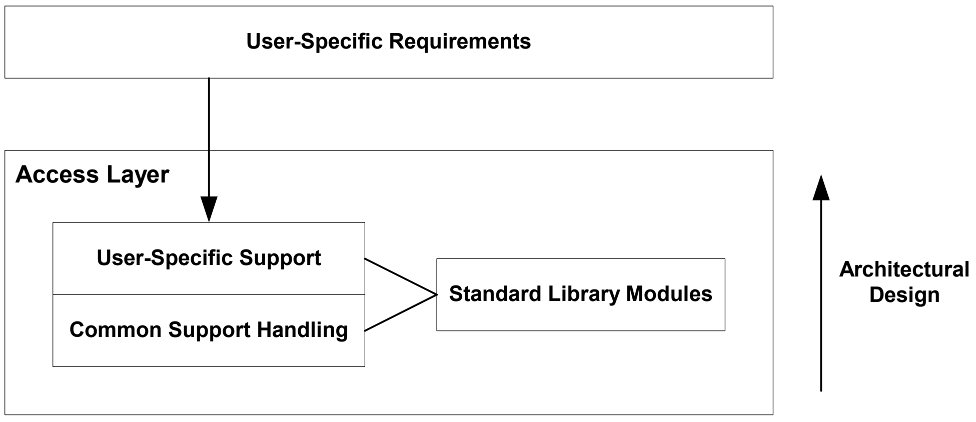


Figure 1-1: Access Layer Architectural Design

1.2.1 Functional Services

The Figure 1-2 highlights the services provided by the access layer.

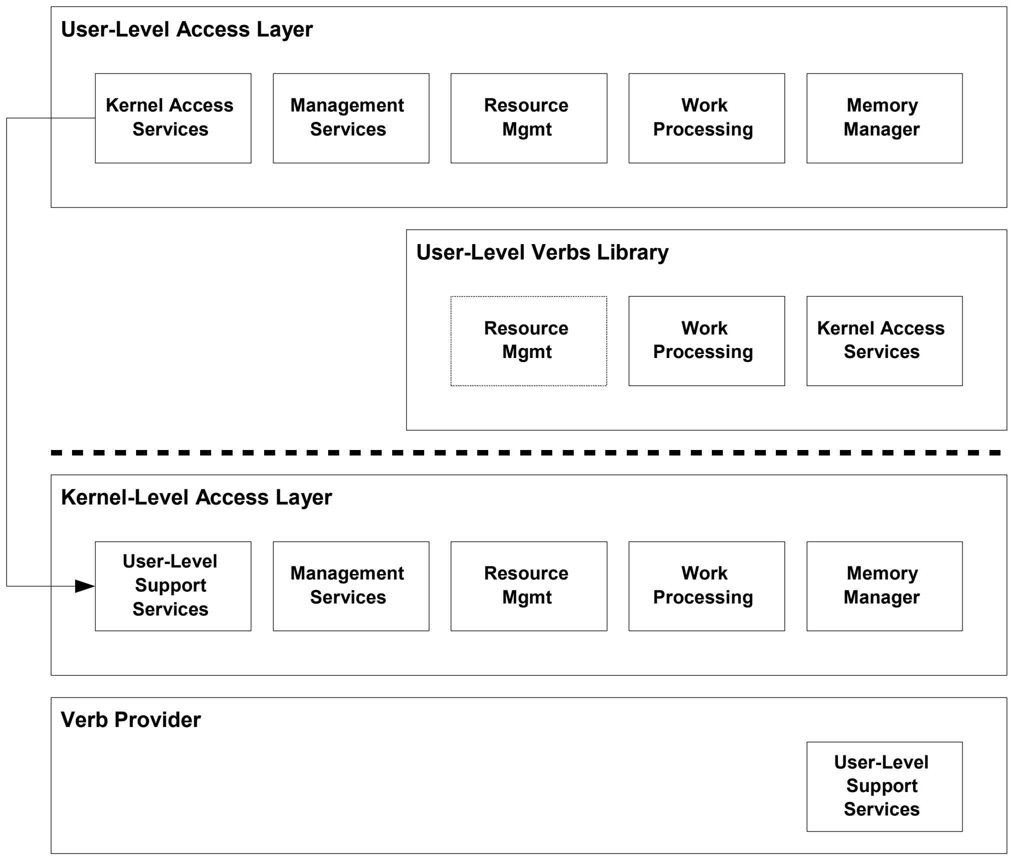


Figure 1-2: Access Layer Functional Services

As seen in the figure, the access layer is located above the verbs provider drivers and operates in both kernel and user-level environments. Kernel access and user-level support services provide proxy agents and implement IO control codes to communicate between the user-level and kernel-level modules. The user-level verbs library is an optional component that provides a speed path for some operations. If a user-level verbs library is not provided, the access layer proxy agent will be used for all user-level operations. The main features of the access layer are grouped into the following areas: management services, resource management, work processing, and memory management. These areas are discussed in more detail in the following sections.

1.3 Theory of Operation

1.3.1 Management Services

The access layer provides extensive support for sending, receiving, and processing management datagrams (MADs). All management classes are supported. Figure 1-3 illustrates the MAD architecture exposed by the access layer.

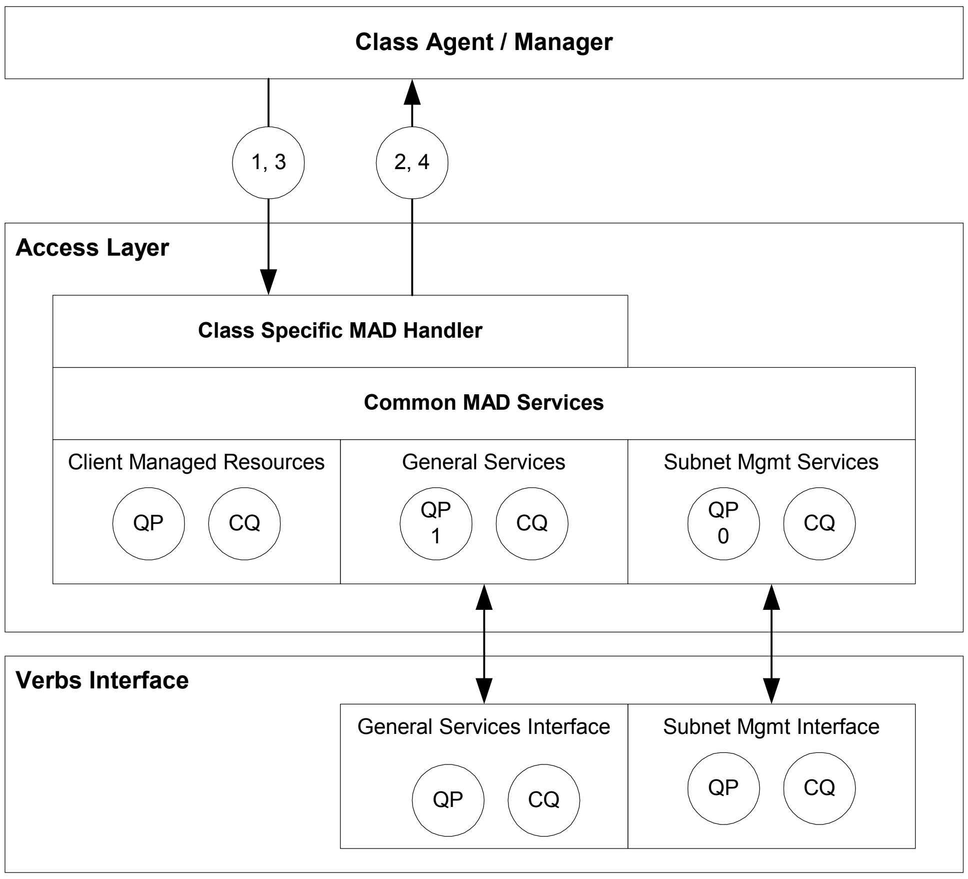


Figure 1-3: Access Layer MAD Architecture

Operation:

1. Clients register as an agent or manager of a specific class. All management classes are supported, but only one class manager may be registered for a specific class.
2. The access layer returns an alias to a QP that may be used to send management datagrams.
3. The class agent or manager posts MADs to the QP alias or, in the case of QP redirection, their own QP.
4. Clients receive incoming MADs through a callback. Unsolicited MADs are routed to the registered class manager. Responses are directed to the agent who initiated the request.

As shown by Figure 1-3, the access layer supports both subnet management (QP0) and general service (QP1) datagrams. Work requests posted to QP0 or QP1 are always processed by the access layer; direct access to these QP's is not allowed. MAD services are divided between common and class specific services. Class specific services deal directly with MADs for a given management class, such as communication management. For convenience, class specific handlers support high-level helper functions that allow users to avoid dealing directly with MADs as shown in Figure 1-4. Class specific helper functions are described in section 1.3.1.1.

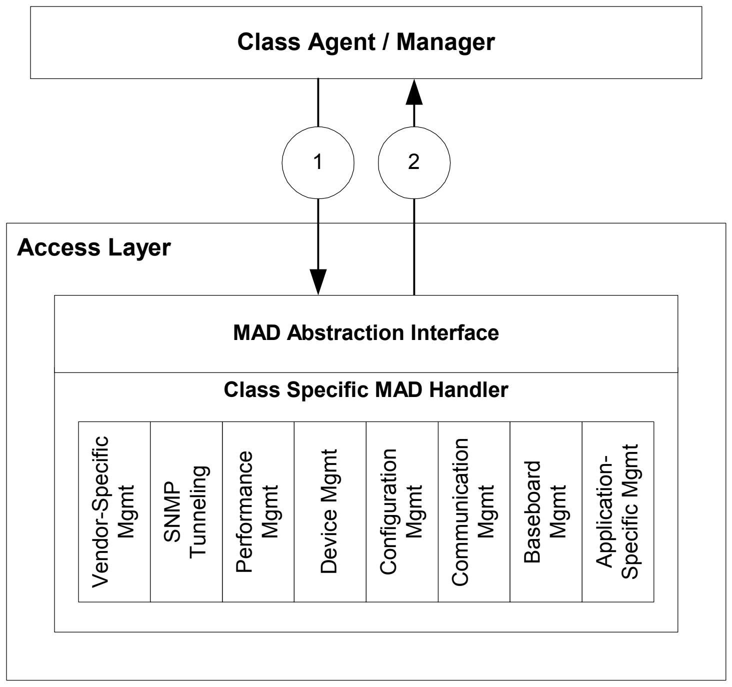


Figure 1-4: MAD Class Specific Handlers

Operation:

1. A user invokes a helper function supported by one of the class specific handlers. MAD helper functions format and send MADs on behalf of the user.
2. The response to a user's request is returned asynchronously to the user.

Regardless of a MAD's class, the access layer provides the following features where appropriate.

- Transaction ID management
- Segmentation and reassembly (SAR) for MAD classes that require SAR
- Support for queue pair redirection
- Queuing to ensure that multiple clients do not overrun an underlying queue pair
- Retransmission of MADs for which a response is expected
- Disabling of MAD support on a per-port basis

1.3.1.1 Management Helper Functions

The access layer provides a set of helper functions that allow users to perform MAD transactions without dealing directly with MADs themselves. These functions include Service Record Registration, Service ID Registration and Resolution, Information Queries, Registration For Local Events, and Subscription For Reports.

1.3.1.1.1 Service Record Registration

Service records allow clients to advertise the availability of basic services to the subnet. The access layer provides support for clients to register and de-register service records with subnet administration. The client provides the service record with the registration request. The access layer returns a handle to the client and performs the MAD transactions with subnet administration. Using the returned handle, the client can unregister the service. The result of the service registration or de-registration is returned to the client through a callback function.

1.3.1.1.2 Service ID Registration and Resolution

Registration of a service ID provides a method for remote users of unreliable datagram services to determine the queue pair number and queue key of a port that supports a given service ID. The access layer provides an interface to allow clients to register and deregister a service ID with the communication manager and to resolve service IDs using the service ID resolution protocol.

Registration:

A client registers a service ID by providing the service ID, the channel adapter and port GUIDs, a callback function, and an optional buffer and length for comparing the private data exchanged during the service ID resolution process. The access layer returns a handle that the client may use to deregister the service.

Resolution:

A client requesting service ID resolution provides the desired service ID, a path record to the remote port, a callback function, and an optional buffer containing private data. A high-level view of the service ID resolution process is shown in Figure 1-5.

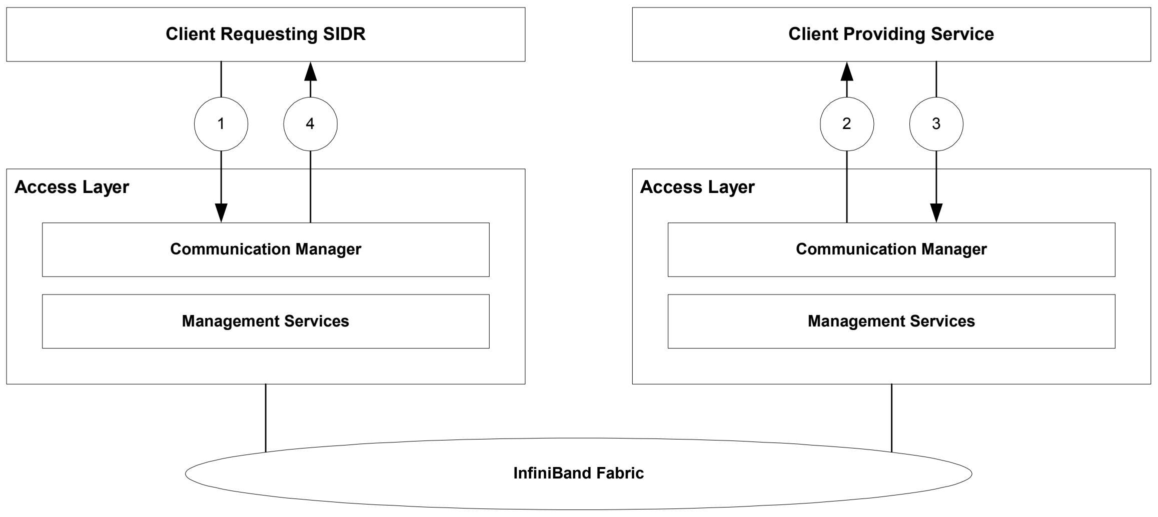


Figure 1-5: Service ID Resolution

Operation:

1. A client makes a service ID resolution request. The communication manager sends a service ID resolution request MAD to a remote node.
2. The communication manager invokes a callback after receiving a service ID resolution request. The client is given information about the request including the private data.
3. The client accepts or rejects the service ID resolution request. A client accepts by providing the QP and QKey. A rejection may optionally indicate redirection to another port.
4. The communication manager returns the result of the service ID resolution process.

1.3.1.1.3 Information Queries

The access layer provides support for client to query for commonly used data. Based on the query type and input data, the access layer performs required the MAD transactions with the appropriate class management entities to obtain the requested data. The result of the query operation is provided to the client through a callback function.

The access layer provides the following queries:

- Query for service records by service name
- Query for service records by well-known service ID
- Query for IOC information by local port GUID
- Query for IO controller service entries by IO controller GUID Note that this query expects the third reserved field in the IOControllerProfile structure is set to the assigned slot ID of the IO controller. Use of this reserved field is a deviation from the InfiniBand™ Architecture specification.
- Query for node record by node GUID
- Query for port record by port GUID
- Query for path records by a port GUID pair
- Query for path records by a GID pair

The access layer supports other queries by sending and receiving MADs through the standard management services.

1.3.1.1.4 Registration For Local Events

Clients of the access layer can register for notification of local events. Local events include the insertion and removal of channel adapters, ports becoming active and inactive, LID changes, partition changes, and IO controllers being assigned or unassigned to a port. Figure 1-6 illustrates the operation of local events.

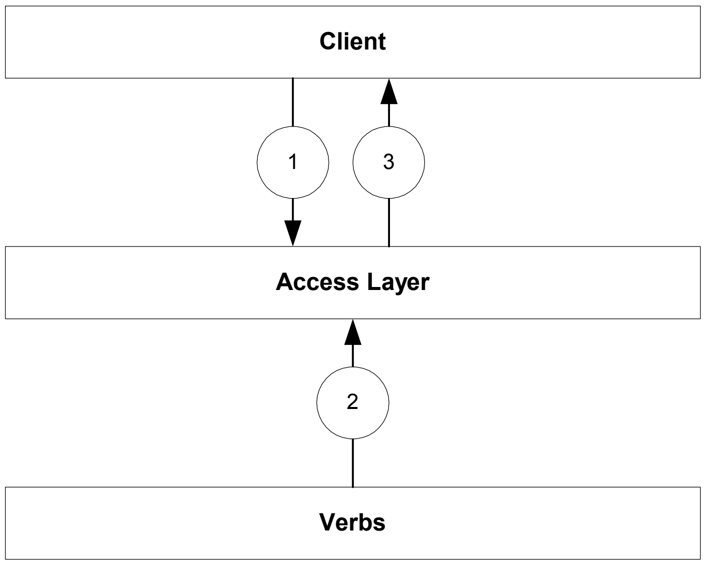


Figure 1-6: Local Event Operation

Operation:

1. The client registers for notification of a local event.
2. The access layer receives notification of a locally occurring event.
3. The access layer notifies clients registered for the local event via a callback.

1.3.1.1.5 Subscription For Reports

Clients of the access layer can register for reports delivered through the event forwarding mechanism.

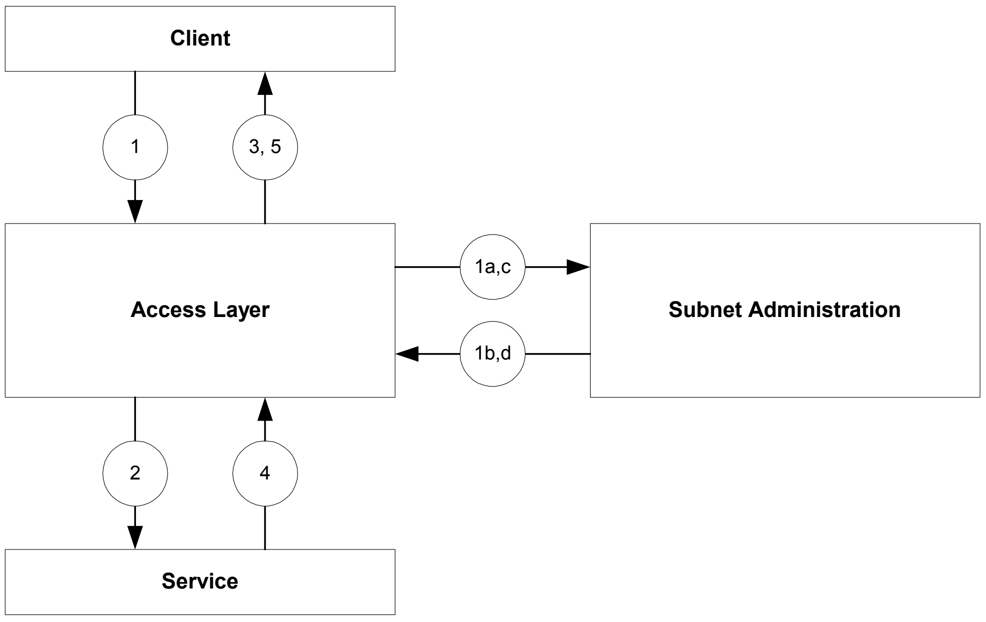


Figure 1-7: Report Subscription

Operation:

1. The client subscribes for a report providing the InformInfo record and the name of the service to subscribe with. The access layer performs the following tasks:
   a. The access layer issues a query to the subnet administration for the named service record. Refer to section 1.3.1.1.3.
   b. The access layer receives the requested service record. The service record contains the GID where the specified service may be found.
   c. The access layer issues a query to the subnet administration to obtain path records associated with the specified service using the source and destination GIDs. Refer to section 1.3.1.1.3.
   d. The access layer receives the requested path records.
2. The access layer initiates a MAD transaction to subscribe for the report.
3. The access layer returns the result of the subscription process to the client through a callback.
4. At some point in the future, the access layer receives a report from the service for the subscribed event.
5. The access layer notifies the client of the subscribed event through a report callback function.

1.3.1.1.6 Rejecting An IO Controller Assignment

The InfiniBand™ Annex E: Configuration Management specification allows a host to reject the assignment of an IO controller. The access layer can perform this operation on behalf of a client. The client requests the IO controller rejection providing the reported IOC information. The access layer performs the MAD transactions with the.

1.3.1.2 Communication Manager

In addition to providing support for class managers and agents to send and receive MADs, the access layer provides a class manager for communication. The communication manager abstracts connecting a queue pair with a remote queue pair and transitions the queue pair through the proper states. A high-level view of the communication manager is shown in Figure 1-8.

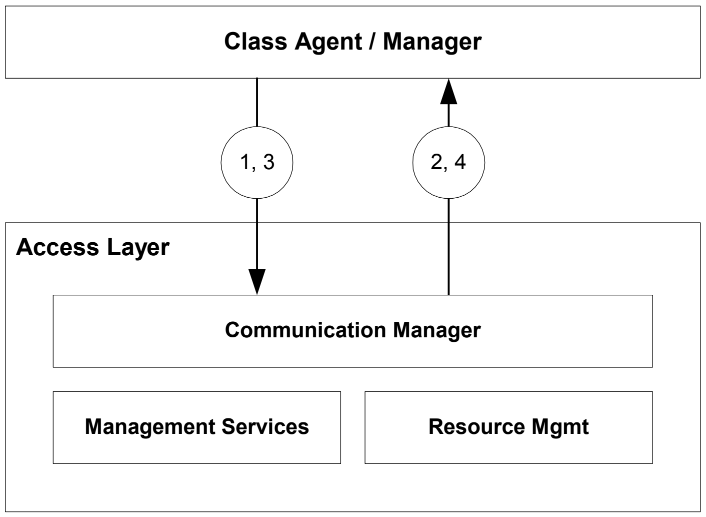


Figure 1-8: Communication Manager

Operation:

1. Clients make a request to connect a QP. For active connection requests, this results in the communication manager sending a connection request MAD to a remote node.
2. The communication manager invokes a callback after receiving a reply to the connection request. The user is given information about the connection attempt, along with the queue pair that will be used.
3. The client accepts or rejects the connection.
4. The communication manager returns the result of the connection process. If the connection was successful, the client may perform data transfers on the given queue pair.

The communication manager performs the following functions.

- Transitions the queue pair state throughout the connection process
- Manages the state of the connection throughout its lifetime
- Supports automatic path migration on the connection

1.3.1.3 Device Management Agent

Some clients of the access layer may require the ability to export an IO controller to the InfiniBand™ fabric as an IO unit. To support this requirement, the access layer provides a device management agent. The device management agent provides a central point of operation that allows multiple clients to register IO controllers. The device management agent maintains the data necessary to respond to device management MADs related to IO unit operations. A high-level view of the device management agent is shown in Figure 1-9.

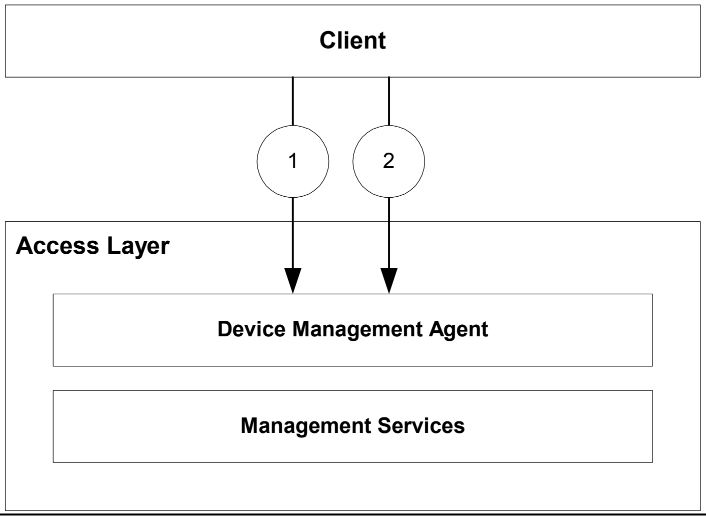


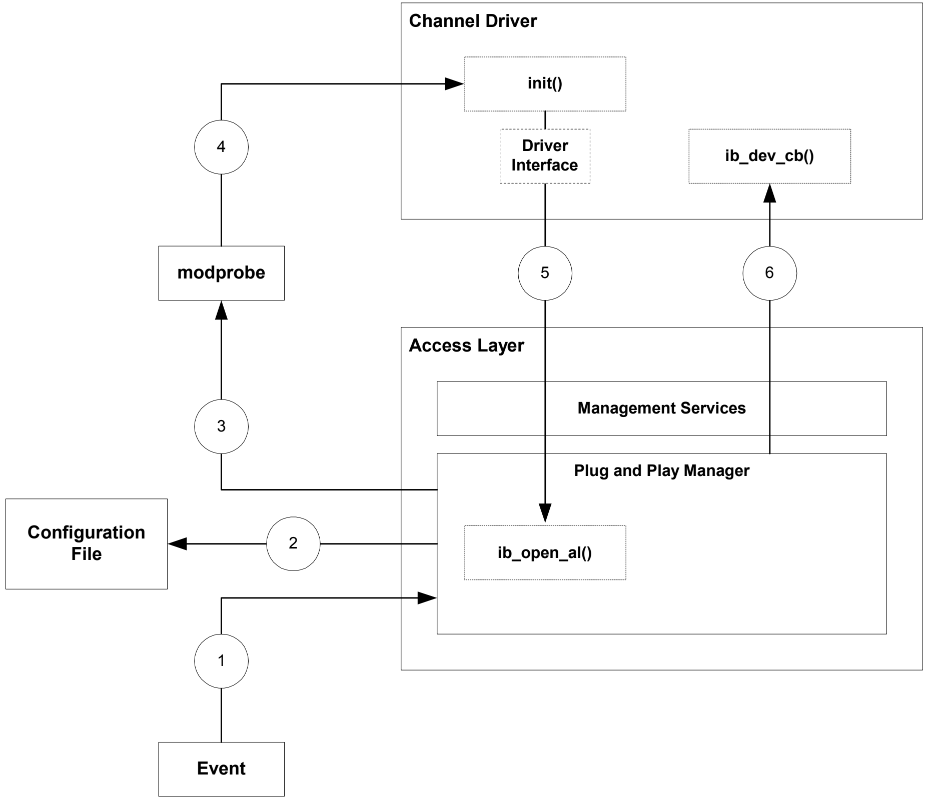
Figure 1-9: Device Management Agent

Operation:

1. The client registers an IO controller profile with the device management agent for a channel adapter. The first IO controller profile registered causes the access layer to set the PortInfo CapabilityMask IsDeviceManagementSupported flag for all ports of the given channel adapter. The access layer returns a handle to the registered IO controller.
2. The client adds service entries to the registered IO controller. The access layer creates a new service entry for the IO controller and returns a service entry handle.

The device management agent performs the following functions.

- Maintains the IO unit information, IO controller profiles, and service entries on behalf of all registered clients
- Automatically assigns IO controllers to slots.
  Note that the access layer sets the third reserved field in the IOControllerProfile structure to the assigned slot ID of the IO controller. Use of this reserved field is a deviation from the InfiniBand™ Architecture specification.
- Indicates the presence of device management support in the PortInfo CapabilityMask flags
- Responds to the device management class MADs related to IO unit operations
- Provides a pass-through interface to a client registered for device management class MADs related to diagnostics



1.3.1.4 Plug and Play Manager

The purpose of the plug and play manager is to notify kernel-mode clients of changes in device status. For example, a channel driver may request notification when an IO controller becomes available or unavailable, or perhaps when a channel adapter port becomes active or goes offline.

The plug and play manager reads a configuration file that contains a mapping of notification events to channel drivers. If a channel driver is not present at the time the plug and play manager needs to notify it, the plug and play manager will load the driver. The plug and play manager delivers notifications to channel drivers asynchronously through a callback registered with the plug and play manager when the driver is loaded. Figure 1-10 illustrates the channel driver load and notification process.

Figure 1-10: Channel Driver Load Sequence

1. The plug and play manager receives notification of an event that has occurred, such as the addition of a new IO controller.
2. The plug and play manager reads a configuration file to determine the correct channel driver to load.
3. The plug and play manager invokes a user-level application to load the appropriate channel driver.

4. The channel driver module is loaded and the driver initialization function is called.
5. The channel driver opens the access layer and registers an interface with the plug and play manager.
6. The plug and play manager notifies the channel driver of a new IO controller instance.

The plug and play manager is a built upon the local event and report subscription mechanisms described in 1.3.1.1.4 and 1.3.1.1.5. The notification callback uses the same parameter format as the local event and report subscription callbacks. Preserving this interface allows the configuration file to be thought of as a "manual" method for a channel driver to subscribe for notifications before the channel driver is actually loaded. The configuration file format supports the same notification capabilities as the programmatic interface.

For IO controller channel drivers, the plug and play manager uses the compatibility string matching method described in InfiniBand™ Annex A: IO Infrastructure specification. This technique allows the configuration file to list a set of compatibility strings for each driver. The matching algorithm uses a search order that selects more specific channel drivers over more generic channel drivers.

1.3.2 Resource Management

The resource management support provided by the access layer is responsible for the allocation and management of exposed channel adapter resources, such as queue pairs, completion queues, address vectors, and so forth. The resource manager is responsible for managing resource domains for resources that are allocated within a hierarchy. For example a CA resource contains CQ and PD resources, and PD's contain QP's. Destruction of a resource domain will automatically result in the release of all associated resources. Figure 1-11 shows the general operation of the resource manager.

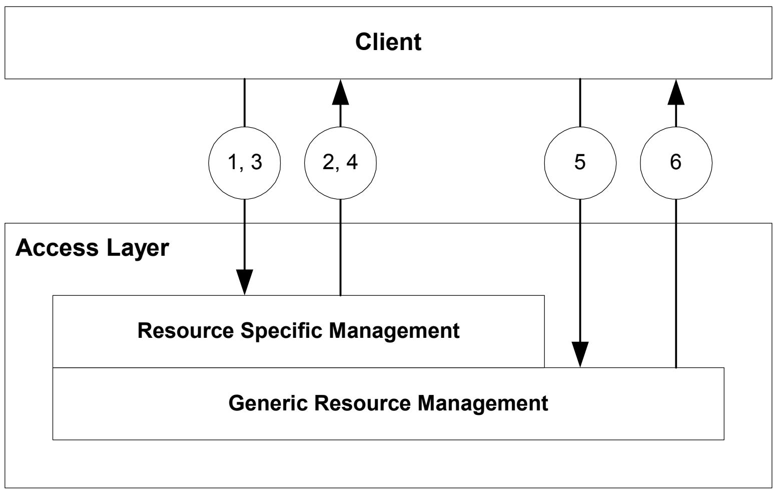


Figure 1-11: Resource Manager

Operation:

1. Clients allocate a new a resource by invoking a resource specific function.
2. The access layer synchronously returns a handle to the allocated resource.
3. Clients can query the access layer for additional properties of the resource. The type of information returned is dependent on the resource being accessed.
4. The access layer synchronously returns the properties of the resource.
5. Once a resource is no longer needed, it may be destroyed through a destruction call.
6. Once the resource has been destroyed, the user is notified. To avoid race conditions, the destruction process operates asynchronously.

Resource management performs the following functions.

- Expose all resources accessible through verbs.
- Manage resource domains to permit destruction of an entire domain. For example, destroying an instance of a CA will automatically release all related resources, such as CQ's, QP's, etc.
- Resource management is not considered a critical speed-path operation.

1.3.3 Work Processing

The work processing module handles issuing work requests and completion processing. In general, work processing deals with operations that occur on queue pairs and completion queues.

1.3.3.1 General Work Processing

A general overview of work processing operation is shown in Figure 1-12.

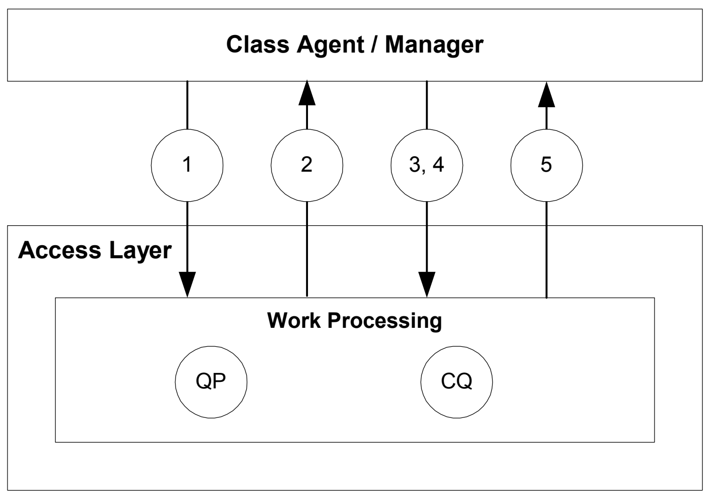


Figure 1-12: Work Processing Overview

Operation:

1. Clients post a work request to a specified queue pair.
2. Upon completion of a work request, the completion queue signals an event to notify the user of the completion. Clients may either wait on a completion event or may be notified through a callback mechanism.
3. Clients re-arm the completion queue to indicate that the completion event should be signaled on the next completed request.
4. Clients poll the completion queue to retrieve information on the completed work request.
5. The completion information is returned to the user synchronously as a result of polling.

Specific requirements of the work-processing module are listed below:

- Completion signaling is controlled by the user
- Users specify whether completion notification is provided via callbacks or event signaling

1.3.3.2 Unreliable Datagram Services

A more detailed operation of a general unreliable datagram QP is outlined below.

Operation:

- The client creates a QP.
- The access layer returns a handle to the newly created QP.
- The client initializes the QP for datagram transfers.
- The client creates an address vector that references the remote destination.
- The access layer returns a handle to the requested address vector.

- The client submits a work request, indicating the QP, QKey, and address vector of the destination.
- The access layer posts the work request for the client.
- The access layer notifies the client of the send completion via the CQ specified when the QP was created.
- The client retrieves the send completion information from the associated CQ.

1.3.3.3 MAD Work Processing

Work processing is handled differently than that shown in Figure 1-12 for clients using the MAD services provided by the access layer. With the use of MAD services, the access layer processes work requests before passing them on to the queue pair. Likewise, the access layer performs post-processing on completion requests. Figure 1-13 highlights the differences.

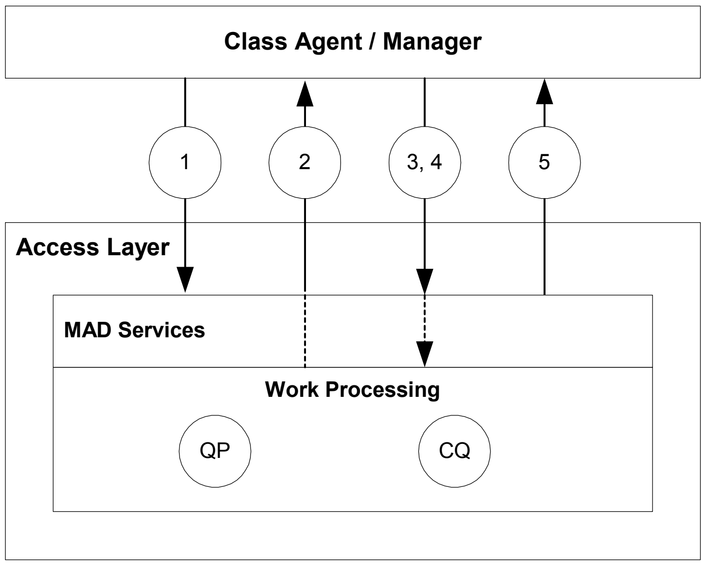


Figure 1-13: Work Processing with MAD Services

Operation:

1. Users indicate that a service should perform a specified task. The indication may come from the posting of a work request, the issuing of a MAD, or the invocation of a function call.

2-4. Steps 2-4 are similar to steps 2-4 in Figure 1-12. These steps only occur for users who request MAD services, but are using queue pair redirection. In such cases, when a user polls a completed work requests from a completion queue, the request is first given to the access layer for additional processing. The completed request is only returned to the user after all post-processing has completed.

5. The completion information is returned to the user asynchronously through a callback.

In general, most users of client specified services would only interface to steps 1 and 5 listed above.

1.3.4 Memory Manager

The memory manager provides users a way to register and deregister memory with a specified channel adapter. Memory must be registered before being used in a data transfer operation. Memory registration is considered separately from resource management for two main reasons. It is considered a speed path operation for kernel-mode users and uses synchronous deregistration. The memory registration process is shown in Figure 1-14.

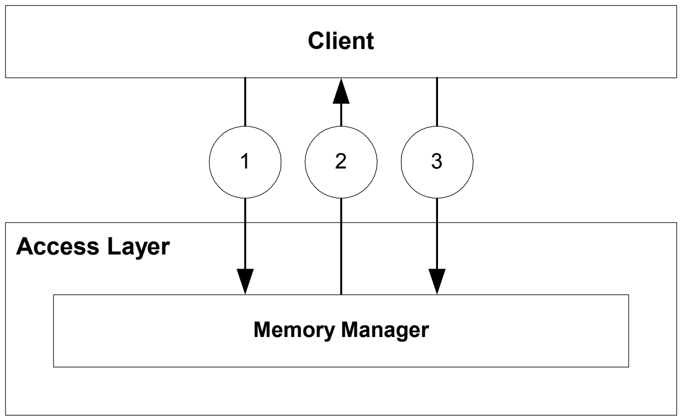


Figure 1-14: Memory Registration

Operation:

1. Clients register memory with the memory manager. Kernel-level users may register either physical or virtual memory. User-level clients cannot access to physical memory and may therefore only register virtual memory.
2. The memory manager returns a handle to the registered memory region, along with a key usable by a remote system to access the memory through an RDMA operation.
3. Once the memory is no longer being used to support data transfer operations, the client de-registers the memory region.

Requirements:

- Virtual and physical memory registration support

- Memory registration in the kernel is considered a speed-path operation

1.3.5 Operation Examples

1.3.5.1 Establishing a Connection to an IOC

The steps below provide a detailed example a client may follow to establish a connection to an IO controller.

Operation:

1. A configuration file is created to register a client for notification of an IO controller with the PnP manager. Refer to section 1.3.1.4 for information on the PnP manager.
2. The access layer invokes a callback function to notify the client of the IO controller.
3. The client creates a protection domain.
4. The access layer returns a handle to the protection domain the client.
5. The client creates a CQ.
6. The access layer returns a handle to the CQ the client.
7. The client creates a QP.
8. The access layer returns a handle to the QP the client.
9. The client obtains path records to the IO unit by querying the subnet administrator using information provided to the notification callback function. Refer to section 1.3.1.1.3.
10. The access layer returns the requested path records to the client.
11. The client formats a connection request structure providing the QP handle and selected path record and initiates the connection request process through the connection manager. Refer to section 1.3.1.2.
12. The access layer performs the connection request and invokes the client callback with the connection reply.
13. The client accepts the connection through the connection manager.
14. The access layer transfers a ready to use datagram to the remote node and returns to the client.
15. The client QP is now connected and ready for use.

1.3.5.2 Client Generated MAD Transaction

The steps below provide a detailed example a client may follow to perform a MAD transaction with a class management entity.

Operation:

1. The client obtains the service record information of the remote node by querying the subnet administrator via the access layer. Refer to section 1.3.1.1.3.
2. The access layer returns the requested service record to the user.
3. The client obtains path records to the remote node by querying the subnet administrator using the information returned in the service record. Refer to section 1.3.1.1.3.

4. The access layer returns the requested path records to the client.
5. The client creates a QP. They may create their own QP or create an alias to one of the special QPs.
6. The access layer returns a handle to the newly created QP or the QP alias to the client.
7. The client initializes the QP for datagram transfers, specifying callback information for handling completed MAD work requests.
8. The client registers the QP with a management class to request MAD support from the access layer. A client can register a single QP with multiple management classes.
9. The client creates an address vector that references the remote destination.
10. The access layer returns a handle to the requested address vector.
11. The client formats a work request, indicating the QP, QKey, and address vector of the destination.
12. The access layer posts the work request for the user. If a response is expected, the access layer continues to resend the request until a response is received or the request times out.
13. The client is notified of the send completion through a send MAD callback.

1.3.5.3 Joining a Multicast Group

The steps below provide a detailed example a client may follow to join and transfer datagrams with a multicast group.

Operation:

1. The client creates a QP.
2. The access layer returns a handle to the newly created QP.
3. The client initializes the QP for datagram transfers.
4. The client may post receive work requests to the QP.
5. The client obtains the PKey of the multicast group. The method used to obtain this PKey is implementation specific.
6. The client obtains the MGID of the multicast group. The method used to obtain the MGID is implementation specific.
7. The client joins the multicast group.
8. The access layer adds the QP as a group member, attaches the QP to multicast group, and returns the member record to the client. The client may receive datagrams from the multicast group before this call returns.
9. The client obtains path records to the group MGID by querying the subnet administrator using the information returned in the service record. Refer to section 1.3.1.1.3.
10. The access layer returns the requested path records to the client.
11. The client creates an address vector that references the MLID.
12. The access layer returns a handle to the requested address vector.
13. The client formats a work request, indicating the well-known QP number, and the QKey and address vector of the multicast group.
14. The access layer posts the work request for the client.
15. The client is notified of the send completion through a send callback.