Infiniband Management

- The Infiniband architecture requires a “subnet manager” node to configure and maintain an Infiniband fabric.

- The subnet manager actually consists of several modules. The term “subnet manager” is commonly used to refer to the node that provides both a Subnet Manager (SM) and the Subnet Administrator (SA).

- The SM configures subnet addresses and switch routing tables.

- The SA serves as the subnet’s database, hosting queries from other subnet nodes.

*All brand and product names are trademarks or registered trademarks of their respective companies. 
Copyright© Intel Corporation 2002, All rights reserved.*
OpenSM Features

- Open source
- Object oriented, extensible design implemented in C.
- Embedded documentation with Robodoc.
- Designed for portability to other platforms and Infiniband interfaces. Current implementation runs in Linux user-space on top of the open source AL API.
- Supports major SM features: multipathing (LMC > 0), partitions, multicast groups, SM fail-over.
- Guarantees optimal routing between any two end-points regardless of subnet topology.
- Supports commonly used SA queries.
- Complete logging facilities to the MAD frame level.
Unsupported Features

- The current developers do not plan to implement the following features. These capabilities could be added by other interested developers.
  - Switches with random forwarding tables.
  - Subnet routers.
  - Complete set of SA queries.
  - Virtual lanes.
  - GUI
The OpenSM Object Model
The OpenSM Object Model

- OpenSM models most of its internal data structures on the physical components in an IB fabric, such as nodes, switches, partitions, etc.

- OpenSM objects are linked together to model the physical relationships on the IB subnet. For example, a switch object contains port objects. Port objects point to other port objects.

- Most OpenSM objects provide functions that hide the details of the object’s implementation.

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.*
The Component Library

- The Component Library is a portable, open source library of useful functions and “containers” implemented in C.
- The Component Library provides lists, balanced trees, variable sized arrays, etc.
- OpenSM makes heavy use of the component library. In the source, you will notice use of functions starting with: cl_. These are calls to the Component Library.
- When in doubt, refer to the component library documentation!

*All brand and product names are trademarks or registered trademarks of their respective companies.
Copyright© Intel Corporation 2002, All rights reserved.
OpenSM Objects: A Top Down View

- The “highest order” OpenSM object is the OpenSM object itself.
- The OpenSM object spans exactly one IB subnet.
- The OpenSM object is built from a variety of other objects:
  - One Subnet object
  - One SM object
  - One SA object
  - A Log object & many other supporting objects
- The following slides will examine each of these components in detail.

*All brand and product names are trademarks or registered trademarks of their respective companies.
Copyright© Intel Corporation 2002, All rights reserved.
The Subnet Object

Name: osm_subn_t
Defined in: osm_subnet.h

- The subnet object models the physical composition of the subnet.

- OpenSM builds the subnet object during discovery and maintains it during subsequent sweeps.

- The subnet object is built from many other object types, for example:
  - Node Objects – represent physical nodes
  - Switch Objects – represent physical switches
  - Router Objects – represent physical routers
  - Partition Objects – represent logical partitions
  - Multicast Group Objects – represent multicast groups.
This view shows the internal object-to-object linkage that models the physical subnet.

*All brand and product names are trademarks or registered trademarks of their respective companies.
Copyright© Intel Corporation 2002, All rights reserved.
Subnet Pointer Tables

- The subnet object contains tables of pointers of each type of object in the subnet.
- The pointer tables provide quick access to objects of a particular type, be it a switch, node port, partition, etc.
- Where appropriate, two pointer tables are provided for each type of object:
  - LidTable – pointers sorted by unicast LID
  - GuidTable – pointers sorted by GUID
- The pointer tables sorted by GUID are implemented using cl_map (red-black trees).
- The pointer tables sorted by LID are implemented using cl_ptr_vector (variable size array of pointers).
Pointer Table Example - Nodes

<table>
<thead>
<tr>
<th>Lid</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GUID</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x123</td>
<td></td>
</tr>
<tr>
<td>0x456</td>
<td></td>
</tr>
<tr>
<td>0x789</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.
How does LMC > 0 look?

<table>
<thead>
<tr>
<th>Lid</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GUID</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x123</td>
<td></td>
</tr>
<tr>
<td>0x456</td>
<td></td>
</tr>
<tr>
<td>0x789</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

LMC > 0 is not a special case!

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.
Node Object

Name: osm_node_t
Defined in: osm_node.h

- A node object represents a node on the subnet.
- A node may be a Host Channel Adapter (HCA), a Target Channel Adapter (TCA), a Switch or a Router.
- A node object is built from other object types:
  - An array of Physical Port objects
  - NodeInfo attribute
  - NodeDescription attribute
- Nodes “own” their corresponding Physical Port objects. All other references to a Physical Port point to a Physical Port inside a Node.
Node Objects are linked to their neighbors through their Physical Port Objects.

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.*
OpenSM Port Object Model

- The object model for ports is somewhat complicated.

- In switches, all ports share a common port GUID value, and are thus logically grouped together. In all other node types, ports have a unique GUID value.

- The OpenSM object model handles this by creating two types of port objects: Physical Ports and Logical Ports.

- Physical Port objects are “owned” by their parent Node object, be it a switch, HCA, etc. and are one-for-one with physical ports on the subnet.

- Logical Port objects contain 1 or more Physical Port objects that all share the same GUID value. Logical Port objects are one-to-many with physical ports on a switch, and one-to-one for physical ports on any other type of node.
Physical Port Objects

Name: osm_physp_t
Defined in: osm_port.h

- Physical Port objects represent physical ports on the subnet.
- A Physical Port object points to its neighbor Physical Port object on the other end of the wire.
- A Physical Port object also contains a pointer to its parent node object.
- These two pointers allow functions to traverse the subnet object in a manner that mirrors the topology of the physical subnet.
Logical Port Objects
Name: osm_port_t
Defined in: osm_port.h

- A Port object represents one or more physical ports that share the same GUID.
- Port objects are “owned” by the Subnet object.
- The Port object contains:
  - An array of pointers to the corresponding Physical Port objects.
  - The GUID value of this port.
  - A pointer to the Node object that owns the ports.
Port Objects are logical collections of Physical Port objects that share the same port GUID. Switches have many Physical Ports per GUID, while other nodes have only 1 Physical Port per GUID.
Switch Object
Name: osm_switch_t
Defined in: osm_switch.h

- A switch object represents a switch on the subnet.

- A switch object is built from other object types:
  - SwitchInfo attribute
  - Pointer to its parent node object
  - Switch forwarding table object
  - Lid Matrix object
LID Matrix Object
Name: osm_lid_matrix_t
Defined in: osm_matrix.h

- The LID Matrix object contains the number of hops to any LID value through any port in the parent switch.
- The LID Matrix is structured like a two dimensional array, with LID values on one axis, and port numbers on the other.
- The LID Matrix expands as new LIDs are added to the subnet.
- The routing algorithms populate the hop count values for each switch’s matrix.
- Constructing a switch’s LID Matrix is a precursor step to building its forwarding table.
Partition Objects

- Partition objects model the logical grouping of physical ports in an IB partition.
- Partition objects contain general information about that partition and a pointer table to port objects in the partition.

<table>
<thead>
<tr>
<th>Port GUID</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x203</td>
<td></td>
</tr>
<tr>
<td>0x789</td>
<td></td>
</tr>
<tr>
<td>0x922</td>
<td></td>
</tr>
</tbody>
</table>

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.*
Pointers to partition objects are stored in a the partition pointer table indexed by P_KEY value.
Multicast Groups

- A multicast group is identified by its LID in the range 0xC000-0xFFFFE, which is assigned to all ports in that multicast group.
- Multicast group objects look a lot like partition objects: They have a pointer table to ports in the multicast group.

<table>
<thead>
<tr>
<th>Port GUID</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x123</td>
<td></td>
</tr>
<tr>
<td>0x456</td>
<td></td>
</tr>
<tr>
<td>0x789</td>
<td></td>
</tr>
</tbody>
</table>

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright © Intel Corporation 2002, All rights reserved.*
Multicast Groups (continued)

- Pointers to multicast group objects are stored in a the multicast group object pointer table indexed by MLID.

<table>
<thead>
<tr>
<th>MLID</th>
<th>Ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xC001</td>
<td></td>
</tr>
<tr>
<td>0xD0B5</td>
<td></td>
</tr>
<tr>
<td>0xE100</td>
<td></td>
</tr>
</tbody>
</table>

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright © Intel Corporation 2002, All rights reserved.*
Path Objects

- SM/SA must provide the information in the PathRecord attribute:
  - Path MTU
  - Rate
  - Packet lifetime
  - Preference value relative to other paths between the two nodes in question.

- There are too many paths in the fabric to precompute all the path records. In a fabric with N nodes, the number of paths is:
  \[ N(N-1)^2 \]
  which grows large quickly!

*All brand and product names are trademarks or registered trademarks of their respective companies.
Copyright© Intel Corporation 2002, All rights reserved.
Handling Paths

- Path information must be determined on the fly by the SM/SA.

- The SM/SA traverses the subnet object, including the local copies of the switch forwarding tables, to accumulate path parameters in a path object.

- The path object is discarded once SA has delivered the information to the requester.
Consider a multicast group that consists of nodes A, B, C.

The multicast spanning tree for this group is shown in red.

The spanning tree covers 3 nodes, 3 switches and 10 Physical ports.
Multicast LID Routing

- Multicast LID routing through a subnet is configured as a single spanning tree, i.e. loops are not allowed.
- Multicast forwarding tables in switches have limited size, so the spanning tree for each multicast group should include only the minimum number of switches.
- Some multicast members can be “send only”
- Non members cannot send to the group.
Multicast Spanning Trees

- A single subnet contains many spanning trees, one per multicast group.
- The subnet object models the physical interconnections between subnet components, whereas spanning trees are logical structures.
- Normal unicast routes can be considered trivial spanning trees.
- Do spanning trees need first class treatment? Can they just be implicit somehow?
Multicast Group Creation

- Start with the set of ports belonging to the group.
- Build the optimal spanning tree by traversing the subnet object.
- Convert the spanning tree into forwarding table entries for the affected switches.
- Configure the switches.
- The same process works for both unicast and multicast routing.
OpenSM Methods & Algorithms
OpenSM Object Methods

- Since we’re using ‘C’, we don’t have the concept of a class with member functions.

- The common work-around is to have pointers to functions in a ‘C’ struct. Unfortunately, these functions are never made inline by the compiler.

- The OpenSM approach will be to use a ‘C’ struct with separately defined functions (and hence able to inline) that operate on that structure, as done in the component library.

- This method offers higher performance at the expense of wordy function names.
OpenSM Method Style

- Follows the Component Library style.
- OpenSM methods use the following form:
  `osm_<object type>_rest_of_name`
- For example:

```c
ib_net16_t
  osm_port_get_base_lid(
    IN const osm_port_t* const p_port );

ib_net16_t
  osm_switch_get_base_lid(
    IN const osm_switch_t* const p_sw );
```
Paths & Switch Forwarding

- Consider the routes through the subnet from A to B.
- There are two possible paths: A – 1 – 4 – B and A – 1 – 2 – 3 – 4 – B
- Obviously, we’d like to take advantage of the shorter path!

*All brand and product names are trademarks or registered trademarks of their respective companies.
Copyright© Intel Corporation 2002, All rights reserved.
IB LID Routing Basics

- Leaf nodes are trivial.
- Switches with only 1 inter-switch link are also trivial to handle.
- Switches with multiple inter-switch links to the same switch can have statically load balanced paths.
- All the interesting action pertains to switches with inter-switch links to more than one other switch.
- Switches with a random forwarding table have a “default route” like in Fibre Channel switching.
- Switches with a linear forwarding table do not have a default route. Thus, a route for every LID in the subnet must be programmed into the switch’s routing table.
OpenSM Subnet Path Strategy

- OpenSM hop-optimizes subnet routing.
  - Routes to from any two nodes traverse the minimum number of switches.
  - Given a choice of equal hop-count paths, OpenSM performs static load balancing using a simple weighted round-robin scheme. Different weights are selected for 1x, 4x and 12x links.

- LMC > 0 paths are also distributed using a weighted round-robin scheme.
Handling Path Records

- Path information must be determined on the fly by the SM/SA.
- The SM/SA traverses the subnet object, including the local copies of the switch forwarding tables, to accumulate path parameters.
OpenSM Subnet Configuration Overview

- The process is asynchronous. As response MADs arrive, OpenSM processes them and may initiate further probing.

- OpenSM throttles DR-MAD traffic so as to not overload the VL15 buffering capability of the switches.

- The various parts of the discovery algorithm are divided into relatively simple Dispatcher “controllers”.

- Initial discovery and subsequent sweeps share common code where possible.

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.*
OpenSM Subnet Configuration

- OpenSM performs a multi-stage subnet configuration:
  1. Discovery & subnet inventory
  2. LID assignment
  3. Configure each switch’s LID Matrix for local leaf nodes.
  4. Configure each switch’s LID Matrix for neighbor’s leaf nodes.
  5. Configure each switch’s LID Matrix for successively more remote nodes.
  6. Build and download switch forwarding tables.
  7. Bring ports to ARMED state as necessary.
  8. Bring ports to ACTIVE state as necessary.

*All brand and product names are trademarks or registered trademarks of their respective companies.
Copyright© Intel Corporation 2002, All rights reserved.
Subnet Discovery & Inventory

- Objects representing subnet components are created on the fly as OpenSM discovers each component.

- The OpenSM architecture is based on a central asynchronous Dispatcher model.

- ‘Controllers’ around the Dispatcher perform the actual processing. The processing performed by any one Controller is narrow in scope.

- The Dispatcher uses its worker threads to call the processing function of each Controller.

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.
Subnet Discovery & Inventory

Serialized Access

Attribute Rcv Controller

Attribute Rcv Controller

Attribute Request Controller

Assignment Controller

Assignment Controller

Inbound SMP Controller

VL15 Outbound Queue

Serialized Access

*All brand and product names are trademarks or registered trademarks of their respective companies.

Copyright© Intel Corporation 2002, All rights reserved.
LID Assignment

- OpenSM attempts to preserve existing LID assignments when possible.
- Base LID assignments on the subnet may follow predictable patterns in practice, but OpenSM treats every base LID as an arbitrary value.
- For example:
  - Switches are not assigned dedicated LID ranges.
  - The base LIDs of leaf nodes on a switch may be discontiguous
- OpenSM assigns consecutive base LIDs when possible.
  - Helps minimizes the number of switch forwarding table blocks needed to hold all possible routes.

*All brand and product names are trademarks or registered trademarks of their respective companies.*

Copyright © Intel Corporation 2002, All rights reserved.
Multicast Group Creation

- Start with the set of ports belonging to the group.
- Build the optimal spanning tree by traversing the subnet object.
- Convert the spanning tree into forwarding table entries for the affected switches.
- Configure the switches.
- The same process works for both unicast and multicast routing.

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.
The Discovery Process

- Discovery starts by requesting the NodeInfo attribute of the node at “hop 0” which is the SM’s own HCA.
- The OpenSM object’s “sweep” method initiates this NodeInfo Request.
- Once started at hop 0, the discovery process is a self-sustaining “chain reaction” the proceeds until all nodes and ports have been discovered.
- The GUID Tables are populated during the discovery process.
Discovery Completion

- The discovery process is multi-threaded, thus nodes may be discovered in different orders from one sweep to the next.

- Discovery is complete when the OpenSM statistics block indicates there are no outstanding QP0 MADs. The SM Mad Controller checks for this condition each time it retires a QP0 MAD to the MAD pool.

- Once all QP0 MADs are off the subnet, the SM Mad Controller posts a message to the Dispatcher. This message causes the State Manager to begin processing the discovered nodes.

*All brand and product names are trademarks or registered trademarks of their respective companies. Copyright© Intel Corporation 2002, All rights reserved.*
LID Configuration

- The LID Manager walks the container of discovered Port objects and compares their reported LID values with the Subnet Object’s LID table.

- The possible cases are:
  - The Port reports the same LID range that OpenSM believes it owns.
  - The Port reports a LID range that overlaps the LID range owned by another Port.
  - The port reports a LID range that OpenSM thought was unoccupied.
  - The port does not yet have a LID range.