**Scalable SA – Access Layer**

High Level Design Document

Modification History

| Rev | Date | Originator | Comment |
| --- | --- | --- | --- |
| 0.1 | 02-June-12 | Sasha Kotchubievsky | Initial draft |

Copyright © 2013 Mellanox Inc. All rights reserved. No part of this document may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express prior written consent of Mellanox Inc.

Table of contents

1. Overview 3

1.1 Purpose and Scope 3

1.2 Project summary 3

1.3 Input data 3

1.3.1 Guid to lid table 4

1.3.2 Node table 4

1.3.3 Link table 4

1.3.4 PKey table 4

1.3.5 SLVL table 5

1.3.6 Ports table 5

1.3.7 LFT 5

2. Path record computation 5

1.4 “Half-World” path record computation 5

1.4.1 Pseudo code 5

1.4.2 Comments 6

1.4.3 Performance 7

1.5 LID to LID Path Record Computation 7

1.5.1 Pseudo code 7

1.6 Data structures 9

1.6.1 Size 9

1.6.2 Usage 9

Introduction

# Overview

## Purpose and Scope

The design document describes Access Layer implementation. The Access Layer is part of SSA project (Scalable SA). The layer is responsible for converting row SA data to actual path records.

## Project summary

Access Layer is data aware. It receives a row data from Distribution Layer and computes an actual path records data. The computed data is pushed to client’s (ACM) cache. Also, Access Layer handles an “epoch” value – data versioning. It notifies clients about data changes and prevents a data distribution if, it’s possible.

Following aspects are relevant for Access Layer design:

1. Path Record computation algorithm
2. Data structures
3. Interface with Distribution Layer
4. Interface with a client
5. Performance
6. Memory consumption
7. Fault tolerance and error handling
8. Runtime components
9. Source code components.

At the first stage of the project we focus on computation algorithm and relevant data structures. In future we will take into account other aspects and specify implementation details.

Now, SA is part of OpenSM. Path Records algorithm is implemented in osm\_sa\_path\_record.c file. We are going to use the implementation as reference.

The current implementation supports:

1. Point-to-point path record computation
2. “half-world” computation (from a specific node to other nodes)
3. “whole-world” computation (all paths in the subnet)
4. Both LID and GID base path records
5. Different selectors
6. Virtual aliases
7. QoS
8. Tavor

For Access Layer’s implementation, only “half-world” computation is relevant.

At the first stage of the project we don’t support: Virtual aliases, Qos, Tavor

## Input data

The input data for path record computation is extracted from OpenSM application using a special plugin. The data is distributed through Distribution Layer in SMDB format. The format supports both fixed length tables and variable length table. Fixed length tables are used for initial data distribution. A variable length table is used for transaction log distribution, for partial data update.

### Guid to lid table

struct ep\_guid\_to\_lid\_tbl\_rec {

uint64\_t guid;

uint16\_t lid;

uint8\_t lmc;

uint8\_t is\_switch;

};

In the table, lid field is a base lid.

### Node table

struct ep\_node\_tbl\_rec {

uint64\_t node\_guid;

uint8\_t is\_enhanced\_sp0;

uint8\_t node\_type;

uint8\_t description[IB\_NODE\_DESCRIPTION\_SIZE];

};

### Link table

struct ep\_link\_tbl\_rec {

uint16\_t from\_lid;

uint16\_t to\_lid;

uint8\_t from\_port\_num;

uint8\_t to\_port\_num;

uint8\_t pad[2];

};

### PKey table

struct ep\_pkey\_tbl\_rec {

uint16\_t port\_lid;

uint16\_t block\_num;

uint16\_t pkey\_tbl[IB\_NUM\_PKEY\_ELEMENTS\_IN\_BLOCK];

};

In the table port\_lid is a base lid.

### SLVL table

struct ep\_slvl\_tbl\_rec {

uint16\_t port\_lid;

uint8\_t port\_in;

uint8\_t port\_out;

uint8\_t slvl\_tbl[IB\_MAX\_NUM\_VLS / 2];

};

### Ports table

struct ep\_port\_tbl\_rec {

uint16\_t max\_pkeys;

uint16\_t port\_lid;

uint8\_t port\_num;

uint8\_t neighbor\_mtu;

uint8\_t link\_speed\_ext;

uint8\_t link\_speed;

uint8\_t link\_width\_active;

uint8\_t vl\_enforce;

uint8\_t is\_fdr10\_active;

uint8\_t pad[3];

};

### LFT

The table is not implemented yet.

### Sa\_db

struct ssa\_db {

/\* Fabric/SM related \*/

uint64\_t subnet\_prefix; /\* even if full PortInfo used \*/

uint8\_t sm\_state;

uint8\_t lmc;

uint8\_t subnet\_timeout;

uint8\_t fabric\_mtu;

uint8\_t fabric\_rate;

boolean\_t enable\_quirks;

boolean\_t allow\_both\_pkeys;

/\* boolean\_t qos; \*/

/\* prefix\_routes \*/

uint8\_t initialized;

};

# Path record computation

## “Half-World” path record computation

### Pseudo code

You can find a pseudo code for a “half-world” path record computation below.

|  |  |
| --- | --- |
| **COMPUTE\_HALF\_WORLD\_PATHS**(source\_guid) | The input parameter is guid of port |
| FIND source\_base\_lid, source\_lmc,source\_is\_sw **IN** ep\_guid\_to\_lid\_tbl | O(N) |
| **FOREACH** dest\_guid **IN** ep\_guid\_lid\_tbl | Iterates all ports in the subnet. O(N) |
| **SET** dest\_guid, dest\_base\_lid, dest\_lmc,dest\_is\_sw | We know these params for all ports |
| shared\_pkeys = **FIND\_SHARED\_PKEYS**(source\_base\_lid, dest\_base\_lid) | The function searches PKeys for source end destination ports in ep\_pkey\_tbl\_rec and compares them |
| **IF** shared\_pkeys **IS NULL** |  |
| **CONTINUE** | Go to next port |
| **FOREACH** source\_lid **IN** [source\_base\_lid… source\_base\_lid 2^source\_lmc-1] | Iterates LIDs on the source node |
| **FOREACH** dest\_lid **IN** [dest\_base\_lid... dest\_base\_lid +2^dest\_lmc-1] | Iterates LIDs on the destination node |
| path = **COMPUTE\_PATH(** source\_lid,dest\_lid,source\_is\_sw,dest\_is\_sw) | See below ‎1.5 |
|  |  |
| **IF** path **IS NOT NULL** |  |
| reversible path = **COMPUTE\_PATH(**dest\_lid, source\_lid , dest\_is\_sw ,source\_is\_sw,) | See below ‎1.5 |
| IF reversible path IS NOT NULL |  |
| SET path\_is\_reversible = TRUE |  |
| **OUTPUT(** path, shared\_pkeys, path\_is\_reversible) | The function outputs the found path |

### Comments

Basically, the computation process receives a source guid. For the guid it finds a base\_lid. At the next step it iterates list of guids. If there are shared PKs between source guid and destination guid, it computes paths between every LID on the source node and destination node. If a path is found, it outputs the path.

For each path we compute following parameters:

1. mtu
2. rate
3. is\_reversible
4. shared PKeys
5. Packet Life Time
6. SL

## LID to LID Path Record Computation

### Pseudo code

|  |  |
| --- | --- |
| **COMPUTE\_PATH (**source\_lid, dest\_lid,source\_is\_sw,dest\_is\_sw) | Input parameter: source and destination LIDs |
|  |  |
|  |  |
|  |  |
| IF source\_is\_sw | If the source is switch |
| FIND source\_port\_num IN ep\_lft\_tbl FOR dest\_lid | We start from outgoing port |
| ELSE |  |
| SET source\_port\_num = -1 | If not, a port is not relevant |
|  |  |
| IF dest\_is\_sw | If the destination is switch |
| SET dest\_port\_num = 0 | We want to point to port 0 |
| ELSE |  |
| SET dest\_port\_num = -1 | If not, a port is not relevant |
|  |  |
|  |  |
| FIND source\_port IN ep\_port\_tbl FOR source\_lid , source\_port\_num |  |
| FIND dest\_port IN ep\_port\_tblFOR dest\_lid , dest\_port\_num |  |
|  |  |
| IF source\_port = dest\_port |  |
| pkt\_life = 0 |  |
| ELSE |  |
| pkt\_life = sa\_db.subnet\_timeout |  |
|  | Initialize path parameters |
| SET mtu = source\_port.mtu |  |
| SET rate = source\_port.rate |  |
| SET hops=0 |  |
|  |  |
| SET port = source\_port |  |
|  |  |
| WHILE port IS NOTdest\_port | Now go through the path step by step |
| FIND link FROM ep\_link\_table FOR port.lid , port.num |  |
|  |  |
| IF IS NULL link |  |
| RETUN NULL | A path is not found |
|  |  |
| SET port\_num=link.dest\_port\_num |  |
| SET lid = link.\_dest\_port\_link |  |
|  |  |
| FIND portINep\_port\_tbl FOR remote\_lid,port\_num |  |
|  |  |
| IF port IS dest\_port |  |
| BREAK; |  |
|  |  |
| FIND port\_guid AND is\_switch IN ep\_guid\_to\_lid FOR lid |  |
| IF NOT is\_switch |  |
| RETURN ERROR |  |
|  |  |
| SET mtu =MIN(mtu,port.neighbor\_mtu) |  |
| SET rate =MIN(rate,port.rate) |  |
|  |  |
| FIND port\_num IN ep\_lft\_tbl FOR dest\_lid | Find outgoing port on the switch |
| FIND port IN ep\_port\_tbl OR lid,port\_num |  |
|  |  |
|  |  |
| SET rate =MIN(rate,port.rate) | Update rate |
|  |  |
| SET hops=hops+1 |  |
|  |  |
| IF hops**>**MAX\_hops | MAX\_HOPS=64 |
| RETURN ERROR |  |
|  |  |
| RETURN mtu,rate,hops,packet\_life\_time |  |

## Data structures

### Size

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table name | Record Size in bytes | Estimated number of records for 40K fabric(4K SW) | Estimated size in MB | Key |
| ep\_guid\_to\_lid\_tlb | 8 | 40,000 | 0.3 | port guid |
| ep\_node\_tlb | 80 | 40,000 | 3 | node guid |
| ep\_link\_tlb | 8 | 185,000 | 0.6 | LID+port |
| ep\_pkey\_tlb | 68 | 40,000 | 2.7 | LID+block |
| ep\_slvl\_tlb | 20 | ? | -- | LID+port in+portout |
| ep\_port\_tlb | 16 | 185,000 | 3 | LID+port |
| ep\_lft\_tlb | --- | 160,000,000 | -- | --- |
|  |  |  |  |  |

### Usage

|  |  |  |  |
| --- | --- | --- | --- |
| Function name | Table name | Usage |  |
| COMPUTE\_HALF\_WORLD\_PATHS | ep\_guid\_to\_lid\_tbl | search by guid |  |
| COMPUTE\_HALF\_WORLD\_PATHS | ep\_guid\_to\_lid\_tbl | iteration |  |
| FIND\_SHARED\_PKEYS | ep\_pkey\_tbl | search by LID | twice |
| COMPUTE\_PATH | ep\_port\_tbl | search by LID and port | twice |
| COMPUTE\_PATH | ep\_link\_tbl | search by LID and port num | twice |
| COMPUTE\_PATH | ep\_guid\_to\_lid\_tbl | search by LID |  |
| COMPUTE\_PATH | ep\_lft\_tbl | search by LID and switch |  |

From the table we can see that we access ep\_guid\_to\_lid\_tbl, ep\_pkey\_tbl, ep\_link\_tbl, ep\_port\_tbl by LID. We can build an index for these tables. It’s a lookup table for LIDs. An index entry corresponds to LID and includes indexes for four tables mentioned above.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LID** | **Index ep\_guid\_to\_lid\_tbl** | **ep\_pkey\_tbl** | **ep\_link\_tbl** | **ep\_port\_tbl** |
| 0x0000 | 0 | 0 | 0 | 0 |
| ….. |  |  |  |  |
| 0xBB80 | 0x123 | 0x123 | 0xBB | 0xBB |

A key in ep\_port\_tbl and ep\_link\_tbl is a pair: LID and port num. In this case, we assume that all records for specific LID are placed in sequence. The index points to a fist record for specific LID and other can be found by iteration. For example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Index** | **LID** | **PORT NUM** | **Other data** |
|  |  |  |  |
|  |  |  |  |