1. STL基本知识
2. STL是什么？

Storm Lake（暴风湖）：这套跑在OPA上的Intel给的程序的名称。

1. FM是什么？

Fabric Manager(网络管理器)，它是网络中某台机器上的两个进程：

Fm主进程：由PM/PA和SM/SA功能（函数）以及Fm DB Sync和一些内部工具组成。

Fe进程：提供了一个封装协议，将上层软件的MAD（管理数据，management datagram）包通过TCP/IP发送给FM。

1. SM是什么？

Subnet Manager（子网管理器）：

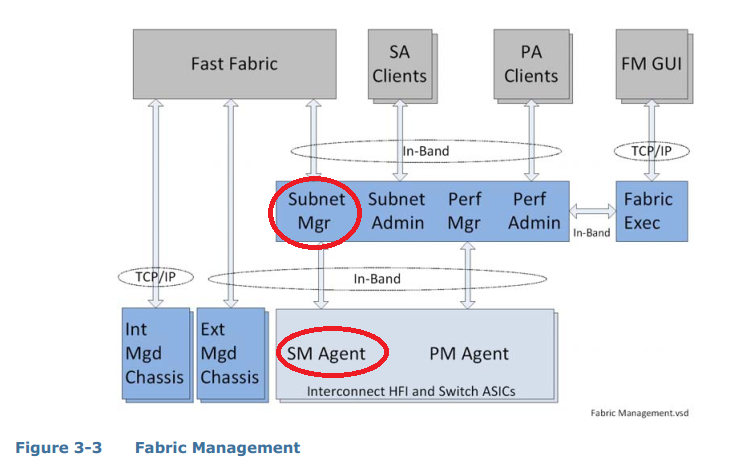
Sm是Fm主进程中的一组线程（其实应该说是由Fm主进程fork出来的一组进程），主要负责拓扑发现、配置和初始化网络的2层设备。

Sm主要负责的事情：  
• Discovering all nodes & switches in fabric  
• Building the fabric topology and routing  
• Assigning local addresses (LIDs) to the devices in the fabric  
• Allocating resources to comply with quality of service policies  
• Establishing the routes between nodes on the subnet  
• Management of multi-cast groups  
• Programming the switch forwarding tables  
• Moving the port states to *active*• Populating the fabric information store, the Subnet Ad  
• Monitoring the subnet for changes and managing new nodes as they appear  
• Updating routes in the subnet in response to changes (such as the addition of new nodes, the removal of nodes, or the failure or degradation of links between nodes)  
• Synchronizing with standby SMs

1. SMA是什么？

SMA（子网管理器代理）：

运行在每一个二层设备上的代理，使SM能够访问2层设备并对其进行配置。



1. PM是什么？

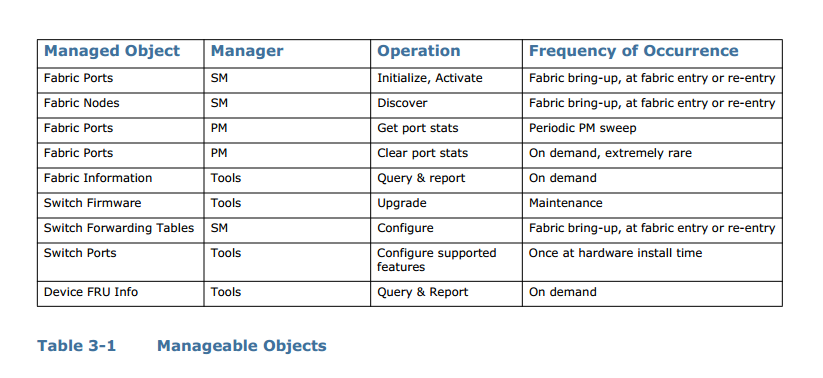
Performance Manager（性能管理器）：

是Fm主进程中的一组线程（其实应该说是由Fm主进程fork出来的一组进程），主要负责统计并存储网络中性能相关的信息（吞吐量啥的），其存储的性能相关的信息可以被上层软件（如如Fm的GUI）访问。

1. PMA是什么？

Performance Manager Agent（性能管理器代理）：与PM的关系如上图figure3-3。

1. 管理者和所管理事务的对应关系：



1. FE 是什么？

Fabric Executive（网络行政部门）：Fm的一个组件,给使用带外（OOB，out of band）局域网络的应用程序的提供一个接口，可以让其使用带外局域网络向Fm发送数据包。

Fe主要是将网络应用程序（如Fm的GUI）的数据包封装成TCI/IP包，然后将这些包发送给Fm。

也就是说Fe是Fm中专门负责与上层软件（如Fm的GUI）通信的一个进程。

1. LID：Local Identifiers

一个24位的标志。用来标识单播地址、组播地址等五种地址。

这个就是网络中的虚拟地址，用来在网络中发送包。SLID：源ID。DLID：目的ID。

1. in-band 和out of band是什么意思？
2. P\_Key是应用于管理器与代理之间通信安全的东西

详见手册卷2-----20.2.2.4 Management Security and Management Key

1. SC和SL

SC：Service Channel

SL：Service Level

STL支持粗粒度的流量划分，将应用划分到不同的Traffic Class。

OPA中有：

TCs（traffic classes）：32个

SLs（service levels）：32个

SCs（service channels）：32个

VLs（virtual lanes）：32个

1TC----1个或多个SL

1SL----1个或多个SC

SC被映射到VL（1对1或者多对1）

SC（Service channel）是FP（fabric packet）中唯一与服务质量相关的标志。

经过这个映射过程，不同等级的流量被划分到不同虚通道中。

1. MAI

MAI提供了一系列发送和接收管理数据包（MAD packet）的接口。

（14）VS

指的是Platform Abstraction，这是操作系统抽象层的历史遗留。它的功能大部分已经被IbAccess Public layer代替了，剩余的是：

events

locks

memory pools/heaps

threads

相关的功能。

（15）操作系统抽象层是个挺重要的东西，把程序的业务逻辑部分和操作系统接口调用部分分离，当程序移植到其他操作系统时，不需要更改业务逻辑部分，只需要重新将操作系统接口调用部分重新实现即可。

（16）sweep：

扫描，SM每隔一段时间就会对子网进行扫描，探测拓扑是不是发生了变化。

（17）umad：？

跟端口有关

（18）PGT和PGFT

Port Group Table ：一组提供等价多路径路由的端口组。

Port Group Forwarding Table ：多路径路由的转发表

（19）MFT

MulticastForwardingTable：多播转发表。

（20）LR-DR SMPs是什么？

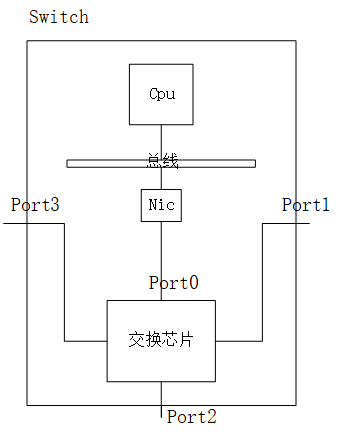
LR指LID routed 基于lid的路由。

DR指directed routed 直接路由。

SMP指SM与SMA之间互相通信的包  
（21）交换机的端口0是不是有什么特殊的地方：

物理不存在的端口？

感觉是这样子的，交换机上面也运行着SMA等进程，相当于交换机上自带一个终端，而交换机的port0与这个终端（HFI、host，网卡）相连。



（22）为什么有一个SC（service channel）到VL（virtual lane）的映射？

Even though all the L2 Link Fabric packet formats support 32 SCs, a device is allowed to implement fewer than 32 sets of resources by utilizing the concept of VLs.

因为在网络的2层（数据链路层）上大家都是有32个SC的，但有些设备没有这么多虚通道，所以就搞了一个映射，可以将多个SC映射到同一个虚通道上，这样保证每个设备都有32个SC。

但是，因为SC15是用来管理网络的，（而VL15具有VL的最高优先级，参见VLArbitrary），所以有且只有SC15能够映射到VL15。

1. STL代码脉络

Opa控制器守护进程运行命令：opafmd -D

参数-D表示开启守护进程

Opafmd.c

-main()

-daemon\_main（）

-loadConfig（）

-parseFmConfig（）

//解析opafm.xml，放在FMXmlCompositeConfig\_t数据结构中。

-updateInstances（）

-spawn（）

//为FM生产模块示例

-fork()

-execle（/usr/lib/opa-fm/runtime/sm）

//此处也可能是-execle（/usr/lib/opa-fm/runtime/fe）

-while(!doStop){ //主进程循环读取处理控制台输入

-parseInput（）；

}

Sm\_linux.c

-main（）

-initSmXmlMemoryPool（）

//初始化了一个内存池，在XML解析的时候使用

-sm\_parse\_xml\_config（）

-parseFmConfig（）

//解析opafm.xml，放在FMXmlCompositeConfig\_t数据结构xml\_config中

//filename =/etc/sysconfig/opafm.xml;

- xmlInitConfig（）

- XmlParseFmConfig（）

- IXmlParseInputFile（input\_file，，TopLevelFields）（一、1 配置）

- IXmlParseFile（）

//使用expat库解析xml

- IXmlParserInit（）

-XML\_SetElementHandler(state->parser, IXmlParserRawStart, IXmlParserRawEnd);

//真正的xml参数解析在上面设置的两个回调函数中，特别//是IXmlParserRawEnd

- IXmlParserReadFile（）

smCopyConfig(&sm\_config,&xml\_config->fm\_instance[sm\_instance]->sm\_config);

pm\_config = xml\_config->fm\_instance[sm\_instance]->pm\_config;

fe\_config = xml\_config->fm\_instance[sm\_instance]->fe\_config;

sm\_dpl\_config = xml\_config->fm\_instance[sm\_instance]->sm\_dpl\_config;

sm\_mc\_config = xml\_config->fm\_instance[sm\_instance]->sm\_mc\_config;

sm\_mls\_config = xml\_config->fm\_instance[sm\_instance]->sm\_mls\_config;

sm\_mdg\_config = xml\_config->fm\_instance[sm\_instance]->sm\_mdg\_config;

- sm\_initialize\_sm\_pool（）

//初始化一个内存池，在创建sm\_vfdg\_config并根据xml\_config进行初始化时使用

//vf---virtual fabric ； dg---device group

-handleVfDgMemory（）

//分配两块内存，分别由dg\_config、vf\_config指针指向，根据xml\_config中的相关数//据结构对以上两块内存进行初始化。

- sm\_main（）

- sm\_spanning\_tree\_resetGlobals（）

//spanningTrees数据结构清0

- read\_info\_file（） //新版本（10.04.196）已经去掉了

//读/etc/sysconfig/opa/opafm.info文件，

//更新全局变量hfm\_install\_dir（fm安装地址）

-pm\_initialize\_config（）

//获取pm相关的xml配置参数，确定是否应该开启pm

-pm\_get\_xml\_config（）

//利用xml读取的配置参数（存储在pm\_config变量中的），初始化pm相关//的全局变量：g\_pmThresholds，g\_pmThresholdsExceededMsgLimit，//g\_pmIntegrityWeights，g\_pmCongestionWeights。

-fe\_initialize\_config（）

//获取fe相关的xml配置参数，确定是否应该开启fe

-sm\_init\_plt\_table（）

//初始化动态包生命周期表（dynamic packet lifetime table），暂时不知道干什么用。

-releaseXmlConfig（）

//既然已经将xml相关配置转换成了全局变量存储了下来，把配置文件存储结构占//用的内存方就可以了（主要释放了xml\_config）

-//进行了SA和SM、SM.lid的配置工作

- vs\_pool\_alloc（，，lidmap）

//vs\_pool\_alloc（Pool\_t \*poolp, uint32\_t reqSize, void \*\*address）

//可以这么理解这个函数：申请一块reqSize大小的内存，将其在poolp中注册，//最后address指向这块内存的起始位置。Poolp是一个特殊的链表，通过这个函//数分配出的内存都记录在这个链表之中，方便内存的管理。

//此处这个函数主要功能：为lidmap分配内存

- vs\_pool\_alloc（，，sm\_GuidToLidMap）

//为sm\_GuidToLidMap分配内存

- cl\_qmap\_init（）

//初始化sm\_GuidToLidMap，但是不知道这个数据结构是干什么用的

- vs\_pool\_alloc（，，sm\_threads）

//给sm\_threads分配内存

//sm\_threads用于记录下面将要开启的线程（进程）。

- vs\_pool\_alloc（，，uniqueSpanningTrees）

//给uniqueSpanningTrees分配内存，但是不知道它是做啥用的

- sm\_routing\_init（）//创建一个map，存储moduleFacMap各种路由算法工厂类

- sm\_routing\_makeModule（）//根据配置文件，创建相应路由算法，存储于

//old\_topology.routingModule

- sm\_process\_vf\_info（）

//处理网络相关信息

- sm\_init\_vf\_info（）

//初始化网络，主要是vfInfo数据结构

- sm\_process\_vf\_qos\_params（）

//处理服务质量相关的参数，没细看，涉及两个数据结构vfInfo和VirtualFabrics

-sm\_assign\_sls（）

//给各个虚网络分配SL，包括配置文件中未定义base\_sl的虚网络

//base\_sl已由配置文件定义，按照其配置设置usedSLs

//base\_sl未由配置文件定义，寻找usedSLs中最小的未使用SL，设置base\_sl

//若虚网络有QOS，则每个虚网络都有一个单独SL，否则，所有虚网络共享

//一个

//多播如果设置了mcast\_isolate

//也是如上，寻找最小的未使用SL，设置mcast\_sl。

//若有QOS。。。。。。。

-sm\_assign\_scs\_to\_sls\_FixedMap（）

//等函数分配服务通道（SC）---服务等级（SL）映射表

-sm\_setup\_SC2VLFixedMap（）

-AllocateSCsForFixedMap（）

//在torus中

//dor->routingSCs = dor->escapeVLs ? 2 : 1;

//dor->routingSCs = dor->escapeVLs ? 4 : 2;

//总共的虚通道数有配置文件中的

//<sm>

//<MinSupportedVLs>8</MinSupportedVLs>（10.4.06版本中去掉了）

//</sm>

//配置。

//其中每个virtual fabric会占用 routingSC个虚通道，而有一些拓

//扑广播的时候需要占用单独的虚通道，所以将MinSupportedVLs%

//routingSCs个虚通道分配给mcastVLs作为广播虚通道，如果余数

//为0，则分配routingSCs多个，如果虚通道不够用，则打印错误

//信息。

-slscFixedAssign（）

//SLtoSC分配sm\_SLtoSC、sm\_SCtoSL，还是不太懂

- sm\_resolve\_pkeys\_for\_vfs（）

//为网络分配pkey？没细看

-//初始化信号量

- vs\_lock\_init（）

//初始化锁

-//初始化MAI子系统：MAI子系统好像是一个通讯库，各个模块用它来发送和接-//收管理数据包（MAD packet）

- sm\_threads[SM\_THREAD\_SA\_READER].function = sa\_main\_reader;（零、1线程）

- sm\_start\_thread(&sm\_threads[SM\_THREAD\_SA\_READER])

- vs\_thread\_create（）

- vs\_real\_thread\_create（）

- impl\_vs\_thread\_create（）

-pthread\_create（）

-sm\_threads[SM\_THREAD\_SA\_WRITER].function = sa\_main\_writer;（零、2线程）

- sm\_start\_thread(&sm\_threads[SM\_THREAD\_SA\_WRITER])

- sm\_threads[SM\_THREAD\_TOPOLOGY].function = topology\_main; （零、3线程）

- sm\_start\_thread(&sm\_threads[SM\_THREAD\_TOPOLOGY]);

- sm\_threads[SM\_THREAD\_ASYNC].function = async\_main; （零、4线程）

- sm\_start\_thread(&sm\_threads[SM\_THREAD\_ASYNC]);

- sm\_threads[SM\_THREAD\_TOP\_RCV].function = topology\_rcv; （零、5线程） - sm\_start\_thread(&sm\_threads[SM\_THREAD\_TOP\_RCV]);

- sm\_threads[SM\_THREAD\_DBSYNC].function = sm\_dbsync; （零、6线程）

- sm\_start\_thread(&sm\_threads[SM\_THREAD\_DBSYNC]);

- sm\_threads[SM\_THREAD\_PM].function = unified\_sm\_pm; （零、7线程）

- sm\_start\_thread(&sm\_threads[SM\_THREAD\_PM]);

（零、3线程）

-topology\_main （）

-vs\_lock（）

//很奇怪，这里是用互斥锁+条件变量实现了一个大互斥锁的功能。这个大互斥锁跟普//通互斥锁相比有什么好处？

-vs\_wrlock（）

-topology\_load\_predef（）

-sm\_routing\_init（）

- cl\_qmap\_init（）

// qmap：Quick Map，cl：组件库。初始化了一个map：moduleFacMap

- sm\_routing\_init（）

//初始化了多种路由模块工厂，包括dor的

-vs\_rwunlock（）

-vs\_unlock（）

- sm\_removedEntities\_init()

//初始化sm\_removedEntities，包含锁和map，存储网络中remove掉的点

- bitset\_init（）

//初始化了三个位图new\_switchesInUse、old\_switchesInUse、new\_endnodesInUse

- sm\_init\_counters（）

//初始化计数器

-。。。。。。

-while（1）{ ////包含sm状态机的转换：sm\_transition（）函数

-if (sm\_peer\_quarantined && sm\_state == SM\_STATE\_NOTACTIVE)

-SM\_Get\_PortInfo（）

//获取端口信息，填充到STL\_PORT\_INFO数据结构中

-sm\_get\_stl\_attribute（）

//利用MAI发送MAD包，等待回复，获取端口信息

sm\_transition(SM\_STATE\_DISCOVERING);

- if ((sm\_state == SM\_STATE\_DISCOVERING) || (sm\_state == SM\_STATE\_MASTER))

- for (i = 0; topology\_functions[i] != NULL; i++)

-（topology\_functions[i])(); //其中就包括topology\_discovery（二，2）

}

（二，1）

-topology\_initialize（）

//主要是获取了本端口（由MAI通信的fd\_topology句柄代表）的端口信息

//和节点信息，如果本节点是交换节点获取交换机信息。

//判断本端口的邻居是sw还是hfi，设置连接模式（直连-hfi，非直连sw）

//初始化sm\_newTopology（sm\_topop），包括其中的几个map：nodeIdMap，nodeMap，portMap，quarantinedNodeMap。从old\_topology中获取路由方法。

-sm\_setup\_node（）

//在这里是将本节点添加到sm\_topop，作为node\_head。重要，应该还会用到

（二，2）

-topology\_discovery（）

-//判断sm\_topop路由算法（routingModule）是否为空，是则根据

-//sm\_config.routing\_algorithm用路由工厂生产一个

- sm\_topop->routingModule->funcs.pre\_process\_discovery（）（三，1）

-vs\_pool\_alloc(&sm\_pool, sizeof(DorTopology\_t), &topop->routingModule->data)

//在&topop->routingModule->data中放置一个DorTopology\_t数据结构

-//根据smDorRouting（来自于sm\_config.smDorRouting），初始化

-//DorTopology\_t变量

-//获取SM所在节点，和端口

-//以下为网络上直接路由探索（直接路由—path，因为现在还没路由转发表）

-for\_all\_nodes(sm\_topop, nodep){

-sm\_topop->routingModule->funcs.discover\_node（） （三，2）

-for (i = start\_port; i <= end\_port; i++){//对所有端口

path[path[0]] = portp->index; //设置每个端口连接节点所对应的path

-sm\_setup\_node（）； （四）

-sm\_topop->routingModule->funcs.discover\_node\_port（） （三，3）

}

}

-post\_process\_discovery（）

（二，3）

-topology\_resolve（）//给拓扑中的节点分配lid

//主要是针对LID进行新旧拓扑的调节，将旧拓扑（记录在lidmap

//数组中）中不符合新拓朴规则（lmc）的lid删除并重新设置，

//最后找出拓扑中最大的lid号

-//先根据配置设置topo->lmc,sm\_lid

-//检查当前节点的端口portp->portData->lid是否符合要求，不符合要求//sm\_clear\_lid（）清除lidmap[]和sm\_GuidToLidMap中的项（lid与guid映射），

//并设置标记

-//将设置标记的端口重新分配lid,函数sm\_set\_lid在lidmap数组范围内（1-//UNICAST\_LID\_MAX）搜寻空闲的连续2^lmc个项的空间，分配给当前端口，并设置//lidmap相应的项，添加sm\_GuidToLidMap相应的项。

（二，4）

- topology\_assignments（）

-//检查是否有环，设置loopPathLidEnd，设置sm\_topop->maxLid

- for\_all\_switch\_nodes（）{

-//检查LinearFDBCap是否够大，设置LinearFDBTop

-//检查MulticastFDBCap是否够大，设置MulticastFDBTop

- pauseState //设置pauseState，好多种情况，没太看明白

- sm\_VerifyAdaptiveRoutingConfig （）//确保

//switchp->switchInfo.AdaptiveRouting信息与配置相同

-//按配置设置nodep->switchInfo.u1.s.LifeTimeValue

- nodep->switchInfo.AdaptiveRouting.s.Pause = pauseState

- SM\_Set\_SwitchInfo（）//将节点信息发回交换节点

}

- topology\_setup\_switches\_LR\_DR（）//

- topology\_setup\_routing\_cost\_matrix（）//计算路径和损耗矩阵记录在

//topop->cost中（此处用的是floyd算法）

- sm\_topop->routingModule->funcs.allocate\_cost\_matrix(sm\_topop)

// sm\_topop->cost（路径损耗表）分配空间（交换机个数^2\*size(uint16)）

//设置sm\_topop->bytes（交换机个数^2\*size(uint16)）

-initialize\_cost\_matrix（）

//初始化损耗矩阵（用最大损耗，这里是32767）

//计算相邻节点间的路径损耗（用nodeInfo），

//填充到cost[i][j]和cost[j][i]，本节点到本节点的损耗清零

- calculate\_cost\_matrix（）

//利用floyd算法，依次把每个节点作为中转节点，计算损耗矩阵

//进行了一些跟多播树路径损耗有关的计算

- post\_process\_routing（）

- \_calc\_dor\_closure（）

- \_is\_path\_realizable（）

//验证从源节点到目的节点是否是通的，

//若通，且向右走（第一给不同的维度上），则在位图//\_DorTopology-> dorRight相应位置置1设置，若通，且向左走（

//第一个不同的维度上），则在位图\_DorTopology-> dorLeft相应位//置置1.

-sm\_initialize\_port\_LR\_DR()

-//设置下一步通信的原lid和目的lid，源lid是sm所在节点的本地端口

//lid，目的lid是参数指定交换节点的0号端口lid（若参数指定的是host节

//点，则找到与其相连的交换节点）。

//根据交换节点是否支持lr\_dr模式选择合适方式初始化端口。

- init\_switch\_lfts（）//就是sm\_topop-> routingModule->funcs.init\_switch\_routing

- for\_all\_switch\_nodes

- calculate\_lft（）

- for\_all\_switch\_nodes(topop, toSwitchp){//把所有交换机点作为目的

-get\_port\_group（）

//里面除了给出一组dor的port还有自适应的port。

//先利用\_DorTopology-> dorRight和left判断是否能到达目的

//地，比较坐标，找出第一个不同的维度，利用//nodep->routingData- >left [dim->dimension]或right信息找//到下一跳的节点及本节点出

//端口号。

- \_get\_dor\_port\_group（）

-\_get\_alternate\_path\_port\_group（）（五，1）

-//设置switch to end point routing

-//设置switch to switch routing（端口0的lid代表了该

//switch）

- for\_all\_physical\_ports（toSwitchp, toSwitchPortp）{

//遍历所有目的交换节点端口，找到对应host节点的端口

- for\_all\_end\_ports(nodep, portp)

//对当前host所有端口的所有lid生成lft

}

- for\_all\_end\_ports(toSwitchp, toSwitchPortp)

//对当前交换机的0号端口所有lid生成lft

}

-\_setup\_pgfdb（）//跟等价多路径路由有关，待看（五，2）

- sm\_initialize\_switch\_LR\_DR（）//将相应lft发送给第一个交换机head\_switch

- sm\_routing\_route\_switch\_LR()

- sm\_routing\_route\_new\_switch\_LR()//给所有（新的？）交换机发送lft

- sm\_routing\_route\_old\_switch（）//给所有（旧的？）交换节点发送lft

- for\_all\_nodes(sm\_topop, nodep){

- for\_all\_ports(nodep, portp){

- sm\_initialize\_port\_LR\_DR（）//开启每个端口，给他们发送portinfo（包含

//新的lid信息）

}

}

-for\_all\_nodes（）{

//SLVL映射相关操作

If（交换机）{

-sm\_initialize\_Switch\_SLMaps（）

-sm\_initialize\_Switch\_SLSCMap（）

-sm\_initialize\_Switch\_SCSLMap（）

-topop->routingModule->funcs.select\_scsl\_map（）

-sm\_initialize\_Switch\_SCSCMap（）

-topop->routingModule->funcs.select\_scsc\_map（）（五，3）·

//（10.4.196）移到sm\_initialize\_Switch\_SCVLMaps（）中了

-for\_all\_ports（）{

-sm\_initialize\_Switch\_SCVLMaps（）

//（10.4.196）移到sm\_initialize\_Switch\_SLMaps（）中了

-sm\_initialize\_VLArbitration（）

}

}else（HFI）{

-sm\_initialize\_Node\_SLMaps（）

- sm\_initialize\_Node\_Port\_SLSCMap（）

- sm\_initialize\_Node\_Port\_SCSLMap（）

- sm\_initialize\_Node\_Port\_SCVLMaps（）

}

}

（二，5）

-topology\_adaptiverouting(void)

-for\_all\_switch\_nodes(sm\_topop, nodep) {

-sm\_AdaptiveRoutingSwitchUpdate（sm\_topop, nodep）；

//主要执行了几个发送操作，先向交换节点发送pgt，然后向交换节点发送pgft。

//最后又发送switchInfo

}

Dor：routingModule. funcs

sm\_shortestpath\_make\_routing\_module(rm);

rm->name = "dor";

rm->funcs.pre\_process\_discovery = \_pre\_process\_discovery;

rm->funcs.discover\_node = \_discover\_node;

rm->funcs.discover\_node\_port = \_discover\_node\_port;

rm->funcs.post\_process\_discovery = \_post\_process\_discovery;

rm->funcs.post\_process\_routing = \_post\_process\_routing;

rm->funcs.post\_process\_routing\_copy = \_post\_process\_routing\_copy;

rm->funcs.setup\_pgs = \_setup\_pgs;

rm->funcs.calculate\_lft = \_calculate\_lft;

rm->funcs.init\_switch\_lfts = \_init\_switch\_lfts;

rm->funcs.get\_port\_group = \_get\_port\_group;

rm->funcs.setup\_xft = \_setup\_xft;

rm->funcs.select\_scsc\_map = \_generate\_scsc\_map;

rm->funcs.process\_swIdx\_change = \_process\_swIdx\_change;

rm->funcs.build\_spanning\_trees = \_build\_spanning\_trees;

rm->funcs.destroy = \_destroy;

rm->copy = \_copy;

（三，2）

-\_discover\_node（）////主要检查节点与邻居的连接是否符合配置（维度，端口，环）

//判断当前节点不是交换节点，返回

//给detected\_dim\_t数组分配空间，存储发现的维度信息

For\_all\_ports(){

-neighbor = sm\_find\_node()

//在topop->nodeArray，和topop->nodeIdMap中找neighbor

-//判断neighbor与当前节点连接的端口是否down，是则把down掉前节点端口

-//若邻居节点是交换节点

-//判断当前端口所在维度是否被detected\_dim\_t数组记录，没有则添加。

-//若邻局节点是hfi

-//判断该维度是否有环。这是一种错误情况。

}

Invalid==1超出维度范围

Invalid==3 （维度不同的）两个port连接到同一个邻居节点

Invalid==2（维度相同的）两个port链接到不同的邻居节点

incorrect\_ca==1 连接hfi的维度上配置有环路

（因为配置文件未对这个维度进行配置，所以不会出现这种情况）

（四）

- sm\_setup\_node（）

-//获取path对应节点信息

-//若path对应节点不在已发现拓扑中

- SM\_Get\_NodeDesc（）//获得path对应节点描述nodeDesc

- SM\_Get\_PortInfo（）//获得path对应节点端口信息conPortInfo

- Node\_Create（）//在拓扑中创建一个节点

-//判断如果是hfi节点，加入ca\_head,ca\_tail链表中

-//如果是交换节点，加入sw\_head,sw\_tail链表中

-//不论那种节点，都加入node\_head,node\_tail链表中

-memcpy((void \*) nodep->path, (void \*) path, 64);//为创建的节点存储路径

- nodep->index = topop->num\_nodes++;//新建节点的序号

- cl\_qmap\_insert（）//将节点插入sm\_newTopology.nodeIdMap中，key是index

//将节点插入sm\_newTopology.nodeMap中，key是NodeGUID

-//一点善后工作如：

cpp->nodeno = nodep->index;

cpp->portno = portp->index;

portp->nodeno = cnp->index;

portp->portno = cpp->index;

//涉及到了给节点和端口结构体中的path[]赋值，节点的path代表sm到该节点的路

//径，端口的path代表sm通过该端口到达的节点的信息。

（三，3）

-\_discover\_node\_port（）

-//nodep->routingData初始化（这个数据结构很重要）

- \_propagate\_coord\_through\_port（）//根据本节点坐标生成邻居节点坐标

-\_lookup\_dimension（）//通过state->dimensionMap查找当前节点，当前端口

//所在维度。

-\_create\_dimension （）//若不存在，根据配置smDorRouting.dimension

//新建维度

-//根据端口记载的维度与方向，生成相邻节点的坐标

-//根据坐标将相邻节点填到nodep->routingData-right[dim->dimension]

//或nodep->routingData- left [dim->dimension]中，表示该节点在某维度上右边

//（加1）是谁，左边是谁（减1）

（五，1）

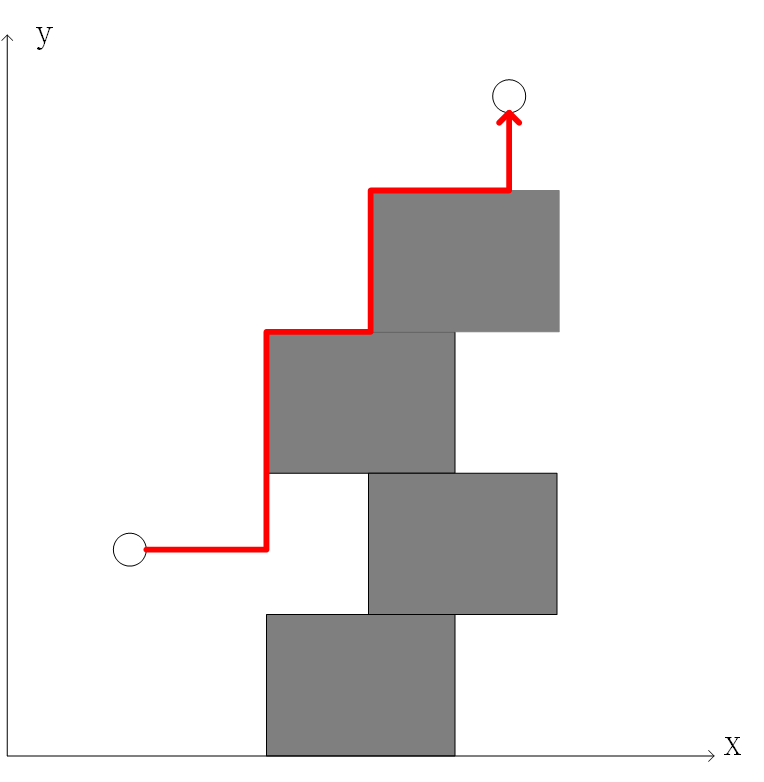
-\_get\_alternate\_path\_port\_group（）//自适应路由，借助了消耗矩阵，防止进入凹形区域

-//遍历当前节点所有端口，找到对应的下一节点

-//如果下一节点在最短路径上

-//当下一节点是唯一在最短路径上的邻居节点时，将该端口返回。

-//否则，按照dor维度顺序挑选下一个端口。



（五，2）

-\_setup\_pgfdb（）

//这个函数是用来生成节点的pgt和pgft的，所使用的信息是（二，4）中的//get\_port\_group（）提供的portgroup。

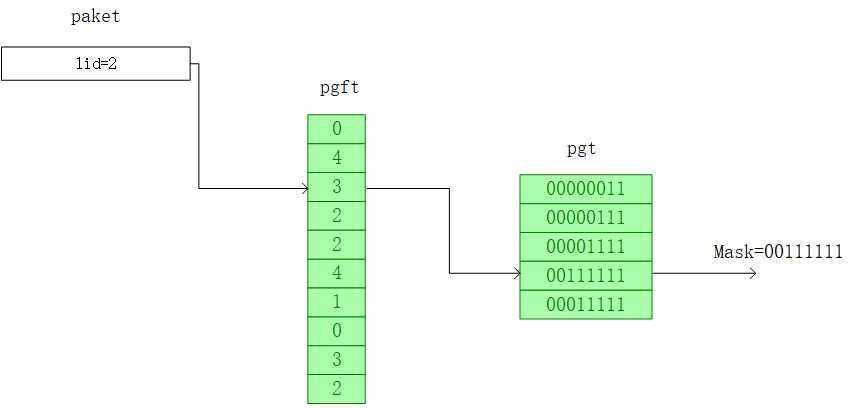
//这里生成了两个数组pgt[]和pgft[]。

//pgt中每一项8个字节，用一个位图mask存储了对应某个目的节点的可选输出端口组，//端口1可用，则mask的0位值1，端口2可用则mask的1位置1。。。。。。

//pgt最多有128项

//pgft是一个长度较长的数组，数组的索引是目的节点lid，数组存储的内容是该目的节点//对应的mask在pgt中的索引。

//因为有好多lid会对应形同的mask，所以这样做能够大大节省空间。



（五，3）

在torus中

dor->routingSCs = dor->escapeVLs ? 2 : 1;

dor->routingSCs = dor->escapeVLs ? 4 : 2;

总共的虚通道数有配置文件中的

<sm>

<MinSupportedVLs>8</MinSupportedVLs>

</sm>

配置。

其中每个virtual fabric会占用 routingSC多个虚通道，而有一些拓扑广播的时候需要占用单独的虚通道，所以将MinSupportedVLs% routingSCs个虚通道分配给mcastVLs作为广播虚通道，如果余数为0，则分配routingSCs多个，如果虚通道不够用，则打印错误信息。

（AllocateSCsForFixedMap（））

-\_generate\_scsc\_map（）

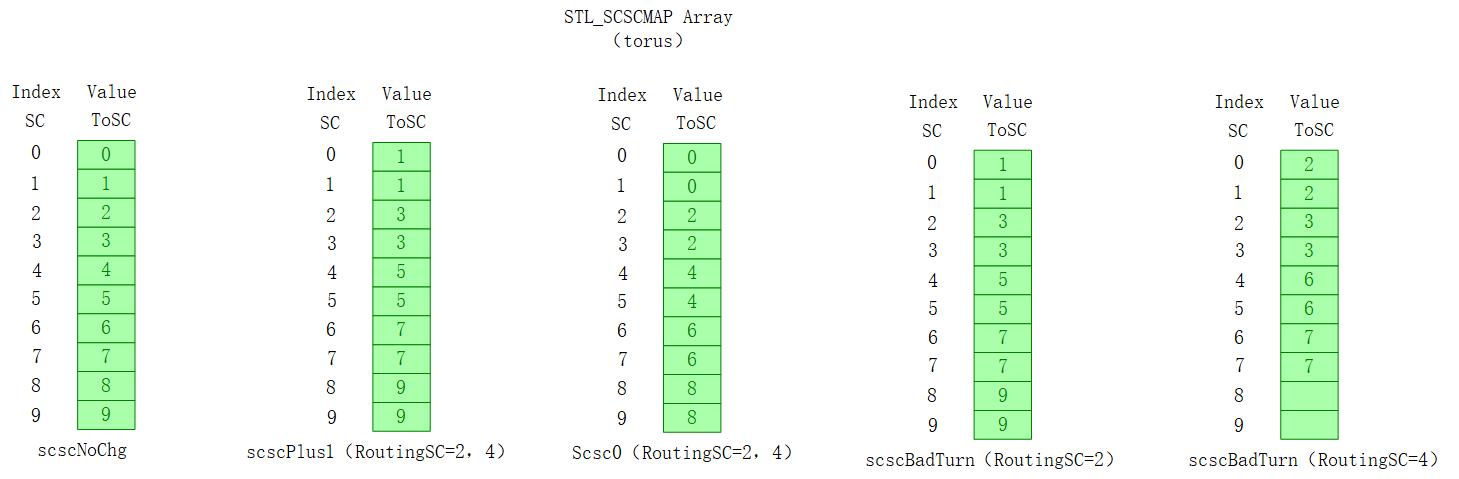
-//创建四种map

//STL\_SCSCMAP scscNoChg;

//STL\_SCSCMAP scscPlus1;

//STL\_SCSCMAP scsc0;

//STL\_SCSCMAP scscBadTurn;



-// STL\_SCSC\_MULTISET \*scsc为一个数组，len=8

//分别存储输入端口和输出端口在8种情况下的STL\_SCSCMAP

//情况1：

//输入端口与HFI相连，输出端口任意：STL\_SCSCMAP为scscNoChg

//情况2：

//输入端口与交换机相连，输出端口与HFI相连：STL\_SCSCMAP为scsc0

//情况3：

//非法转角（没有按照维度顺序那样从低维到高维）：STL\_SCSCMAP为scscBadTurn

//情况4：

//如果是mesh网络，维度内以及切换维度（按维度顺序）：

//STL\_SCSCMAP为scscNoChg

//情况5：

//如果是torus或torus-mesh混合网络，切换维度（按维度顺序）：

// STL\_SCSCMAP为scsc0

//情况6：

//如果是torus或torus-mesh混合网络，输入输出端口不在维度内dateline上：

// STL\_SCSCMAP为scscNoChg

//情况7：

//如果是torus或torus-mesh混合网络，输入输出端口在维度内dateline上，且不是

//回头路：

//STL\_SCSCMAP为scscPlus1

//情况8：

//如果是torus或torus-mesh混合网络，输入输出端口在维度内dateline上，且是

//回头路：

//STL\_SCSCMAP为scscNoChg

最终会为每个端口都生成一个SC2SC表，如下，表格所在的端口为输入端口，每一行分别对应不同输出端口的情况，输出端口为port0则在第一行中寻找输出SC。。。。。



1. 主要数据结构
2. 配置相关的数据结构

（1）存储解析完成后配置参数的数据结构

typedef struct \_FMXmlCompositeConfig {

FMXmlInstance\_t \*fm\_instance\_common;

FMXmlInstance\_t \*fm\_instance[MAX\_INSTANCES];

XmlDebug\_t xmlDebug;

} FMXmlCompositeConfig\_t;

typedef struct \_FMXmlInstance {

FMXmlConfig\_t fm\_config;

SMXmlConfig\_t sm\_config;

SMDPLXmlConfig\_t sm\_dpl\_config;

SMMcastConfig\_t sm\_mc\_config;

SmMcastMlidShare\_t sm\_mls\_config;

SMMcastDefGrpCfg\_t sm\_mdg\_config;

VFXmlConfig\_t vf\_config;

DGXmlConfig\_t dg\_config;

AppXmlConfig\_t app\_config;

PMXmlConfig\_t pm\_config;

FEXmlConfig\_t fe\_config;

} FMXmlInstance\_t;

typedef struct \_SMXmlConfig { //重点，内容较多

。。。。。。

char routing\_algorithm[STRING\_SIZE];

。。。。。。

} SMXmlConfig\_t;

（2）解析过程中用到的数据结构实例

typedef struct IXmlParserState {

// TBD - later support input from a memory buffer

XML\_Parser parser;

int flags; /\* parser option flags \*/

unsigned depth; /\* how many nested（嵌套的） elements deep >= 1 \*/

unsigned skip; /\* skip all elements until we return to this depth \*/

IXmlParserStackEntry\_t current; /\* state of current element being parsed \*/

XML\_Char \*content; /\* text contents of current element \*/

unsigned len; /\* number of characters in content \*/

IXmlParserStack\_t stack; /\* stack of parent elements' states \*/

unsigned error\_cnt;

unsigned warning\_cnt;

void \*context; /\* caller supplied context \*/

IXmlParserPrintMessage printError;

IXmlParserPrintMessage printWarning;

} IXmlParserState\_t;

typedef struct IXmlParserStackEntry {

char \*tag; // locally allocated copy

const IXML\_FIELD \*field;

const IXML\_FIELD \*subfields;

void \*object;

unsigned tags\_found; // total subfield tags encountered including dups

uint64 fields\_found; // bit mask of indexes into subfields

} IXmlParserStackEntry\_t;

typedef struct \_IXML\_FIELD {

const char \*tag; /\* tag name in XML file or "\*" \*/

char format; /\* format for tag, see below for choices \*/

int offset; /\* offset in C structure for tag's value \*/

int size; /\* size of field in C structure \*/

IXML\_FORMAT\_FIELD\_FUNC format\_func; /\* custom function to format output \*/

struct \_IXML\_FIELD \*subfields; /\* fields within a container tag \*/

IXML\_START\_TAG\_FUNC start\_func; /\* function when input <tag> \*/

IXML\_END\_TAG\_FUNC end\_func; /\* function when input </tag> \*/

} IXML\_FIELD;

static IXML\_FIELD TopLevelFields[] = {

{

tag:"Config",

format:'K',

subfields:ConfigFields,

start\_func:ConfigXmlParserStart,

end\_func:ConfigXmlParserEnd

},

{ NULL }

};

static IXML\_FIELD ConfigFields[] = {

{ tag:"XmlDebug", format:'k', subfields:XmlDebugFields, start\_func:XmlDebugParserStart, end\_func:XmlDebugParserEnd },

{ tag:"Common", format:'K', subfields:CommonFields, start\_func:CommonXmlParserStart, end\_func:CommonXmlParserEnd },

{ tag:"Fm", format:'K', subfields:FmFields, start\_func:FmXmlParserStart, end\_func:FmXmlParserEnd },

{ NULL }

};

static IXML\_FIELD CommonFields[] = {

{ tag:"Shared", format:'K', subfields:CommonSharedFields, start\_func:CommonSharedXmlParserStart, end\_func:CommonSharedXmlParserEnd },

{ tag:"Sm", format:'K', subfields:SmFields, start\_func:SmXmlParserStart, end\_func:SmXmlParserEnd },

{ tag:"Fe", format:'K', subfields:FeFields, start\_func:FeXmlParserStart, end\_func:FeXmlParserEnd },

{ tag:"Pm", format:'K', subfields:PmFields, start\_func:PmXmlParserStart, end\_func:PmXmlParserEnd },

{ tag:"Applications", format:'k', subfields:VfsAppFields, start\_func:VfsAppXmlParserStart, end\_func:VfsAppXmlParserEnd },

{ tag:"DeviceGroups", format:'k', subfields:VfsGroupsFields, start\_func:VfsGroupsXmlParserStart, end\_func:VfsGroupsXmlParserEnd },

{ tag:"VirtualFabrics", format:'k', subfields:VfsFields, start\_func:VfsXmlParserStart, end\_func:VfsXmlParserEnd },

{ tag:"PmPortGroups", format:'k', subfields:PmPgsFields, start\_func:PmPgsXmlParserStart, end\_func:PmPgsXmlParserEnd },

{ NULL }

};

static IXML\_FIELD FmFields[] = {

{ tag:"Shared", format:'K', subfields:FmSharedFields, start\_func:FmSharedXmlParserStart, end\_func:FmSharedXmlParserEnd },

{ tag:"Sm", format:'K', subfields:SmFields, start\_func:SmXmlParserStart, end\_func:SmXmlParserEnd },

{ tag:"Fe", format:'K', subfields:FeFields, start\_func:FeXmlParserStart, end\_func:FeXmlParserEnd },

{ tag:"Pm", format:'K', subfields:PmFields, start\_func:PmXmlParserStart, end\_func:PmXmlParserEnd },

{ tag:"Applications", format:'k', subfields:VfsAppFields, start\_func:VfsAppXmlParserStart, end\_func:VfsAppXmlParserEnd },

{ tag:"DeviceGroups", format:'k', subfields:VfsGroupsFields, start\_func:VfsGroupsXmlParserStart, end\_func:VfsGroupsXmlParserEnd },

{ tag:"VirtualFabrics", format:'k', subfields:VfsFields, start\_func:VfsXmlParserStart, end\_func:VfsXmlParserEnd },

{ tag:"PmPortGroups", format:'k', subfields:PmPgsFields, start\_func:PmPgsXmlParserStart, end\_func:PmPgsXmlParserEnd },

{ NULL }

};

。。。。。。。。。。。。。。。。。。

（3）Biu在配置文件新加入项的路径

配置文件路径

static IXML\_FIELD ConfigFields[] = {

{ tag:"Common", format:'K', subfields:CommonFields, start\_func:CommonXmlParserStart, end\_func:CommonXmlParserEnd },

}

static IXML\_FIELD CommonFields[] = { （1）

{ tag:"Sm", format:'K', subfields:SmFields, start\_func:SmXmlParserStart, end\_func:SmXmlParserEnd },

}

static IXML\_FIELD SmFields[] = { （2）

{ tag:"MeshTorusTopology", format:'k', subfields:SmDorRoutingFields, start\_func:SmDorRoutingXmlParserStart, end\_func:SmDorRoutingXmlParserEnd },

}

static IXML\_FIELD SmDorRoutingFields[] = { （3）

{ tag:"Debug", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, debug) },

{ tag:"UseEscapeVLs", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, escapeVLs) },

{ tag:"UseFaultRegions", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, faultRegions) },

{ tag:"Dimension", format:'k', subfields:SmDimensionFields, start\_func:SmDimensionStart },

{ tag:"RouteLast", format:'k', end\_func:SmDorRouteLastEnd },

{ tag:"WarnThreshold", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, warn\_threshold) },

#ifdef BIU

{ tag:"DimensionBiu", format:'k', subfields:SmDimensionBiuFields, start\_func:SmDimensionBiuStart },

#endif

{ NULL }

};

static IXML\_FIELD SmDimensionBiuFields[] = { （4）

{ tag:"PortBiu", format:'u', IXML\_FIELD\_INFO(SmDimensionBiu\_t, port)},

{ NULL }

};

指的是创建数据结构的地方

指的是数据结构的引用关系

数据结构路径

FMXmlCompositeConfig\_t configp；

typedef struct \_FMXmlCompositeConfig {

FMXmlInstance\_t \*fm\_instance\_common; （1）

FMXmlInstance\_t \*fm\_instance[MAX\_INSTANCES];

int num\_instances;

XmlDebug\_t xmlDebug;

} FMXmlCompositeConfig\_t;

typedef struct \_FMXmlInstance {

FMXmlConfig\_t fm\_config;

SMXmlConfig\_t sm\_config; （2）

SMDPLXmlConfig\_t sm\_dpl\_config;

SMMcastConfig\_t sm\_mc\_config;

SmMcastMlidShare\_t sm\_mls\_config;

SMMcastDefGrpCfg\_t sm\_mdg\_config;

VFXmlConfig\_t vf\_config;

DGXmlConfig\_t dg\_config;

AppXmlConfig\_t app\_config;

PMXmlConfig\_t pm\_config;

FEXmlConfig\_t fe\_config;

} FMXmlInstance\_t;

typedef struct \_SMXmlConfig {

.。。。。。

SmDorRouting\_t smDorRouting; （3）

。。。。。。

} SMXmlConfig\_t;

typedef struct \_SmDorRouting {

uint8\_t debug;

uint8\_t dimensionCount;

uint8\_t numToroidal;

uint8\_t routingSCs;

uint32\_t warn\_threshold;

SmDimension\_t dimension[MAX\_DOR\_DIMENSIONS];

DorTop\_t topology;

uint8\_t escapeVLs;

uint8\_t faultRegions;

XMLMember\_t routeLast; // device group indicating HFIs that should be routed last.

#ifdef BIU

SmDimensionBiu\_t dimensionbiu; （4）

#endif

} SmDorRouting\_t;

typedef struct \_SmDimensionBiu {

uint8\_t port;

} SmDimensionBiu\_t;

2、线程相关的数据结构（主要用在sm\_main创建各种线程的时候）

sm\_l.h

typedef struct {

Thread\_t handle;

Threadname\_t name;

void (\*function)(uint32\_t, uint8\_t \*\*);

uint8\_t \*id;

} SMThread\_t;

vs\_g.h（操作系统抽象层）

typedef struct New\_Thread

{

unsigned char name[VS\_NAME\_MAX];

uint32\_t magic;

unsigned int user\_flag;

int priority;

struct Thread \*next; /\* XXXXXX maybe will use these \*/

struct Thread \*prev;

uint64\_t opaque[OPAQUE\_THREAD\_ELEMENTS];

}

Thread\_t;

最终会存放在这里

vs\_thr.c (操作系统抽象层)

typedef struct {

uint32\_t magic;

void (\*start\_function)(uint32\_t, uint8\_t \*\*);

pthread\_t thread;

Threadname\_t create\_pid;

Threadname\_t create\_ppid;

uint32\_t argc;

uint8\_t argv\_copied;

uint8\_t \*\*argv;

uint32\_t dead;

pthread\_cond\_t dead\_cond;

pthread\_mutex\_t dead\_mutex;

pthread\_attr\_t attr;

char name[64];

} Implpriv\_Thread\_t;

3、锁相关的数据结构

typedef volatile struct

{

uint32\_t type; /\* type of lock (VLOCK\_SPIN, VLOCK\_RWTHREAD or VLOCK\_THREAD) \*/

uint32\_t status; /\* status of lock (VLOCK\_LOCKED or VLOCK\_FREE) \*/

uint32\_t magic; /\* magic of lock (VLOCK\_THREAD/SPINLOCK\_MAGIC) \*/

/\* implementation private data \*/

uint64\_t opaque[OPAQUE\_LOCK\_ELEMENTS];

}Lock\_t;

typedef struct

{

uint8\_t name[VS\_NAME\_MAX];

union {

struct { // for THREAD

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

} lock;

struct { // for RWTHREAD

pthread\_rwlock\_t rwlock;

ATOMIC\_UINT reader\_count;

} rwlock;

struct { // for recursive SPIN

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

Threadname\_t owner\_thread;

uint32\_t spin\_count;

} spin;

} u;

} Implpriv\_Lock\_t;

4、拓扑发现相关的数据结构

nodep->routingData：

// DOR Topology Information

typedef struct \_DorNode {

int8\_t coords[SM\_DOR\_MAX\_DIMENSIONS];

Node\_t \*node;

int multipleBrokenDims;

struct \_DorNode \*left[SM\_DOR\_MAX\_DIMENSIONS];

struct \_DorNode \*right[SM\_DOR\_MAX\_DIMENSIONS];

} DorNode\_t;

State： DorDiscoveryState\_t

Dim： DorDimension\_t

dorNode：DorNode\_t ：Node\_t ->routingData

topop：Topology\_t

nodep：Node\_t

portp：Port\_t

dorTop：DorTopology\_t： \_Topology ->routingModule->data

typedef struct \_Topology {

uint32\_t lid; // the last lid used

uint32\_t lmc; // the global LMC

。。。。。。。。。

uint32\_t state; // SM state

uint32\_t maxLid; // maximum Lid assigned

uint16\_t \*cost; // array for path resolution

Node\_t \*node\_head; // linked list of nodes

Node\_t \*node\_tail; // ditto

Node\_t \*ca\_head; // linked list ofFI nodes

Node\_t \*ca\_tail; // ditto

Node\_t \*switch\_head; // linked list of SWITCH nodes

Node\_t \*switch\_tail; // ditto

。。。。。。

uint32\_t bytes; // cost/path array length

。。。。。。

cl\_qmap\_t \*nodeMap; // Sorted GUID tree of all nodes

cl\_qmap\_t \*portMap; // Sorted GUID tree of all ports

cl\_qmap\_t \*nodeIdMap; // Sorted Node tree based on the locally assigned node id.

cl\_qmap\_t \*quarantinedNodeMap; // Sorted GUID tree of all nodes

。。。。。。。

//forces setting of client rereg bit for all ports at end of sweep

uint8\_t force\_client\_reregistration;

。。。。。。

RoutingModule\_t \* routingModule;

。。。。。。

bitset\_t \*smaChanges;

VirtualFabrics\_t \*vfs\_ptr;

。。。。。。

} **Topology\_t;**

typedef struct \_RoutingModule {

const char \* name;

uint16\_t alg;

RoutingFuncs\_t funcs;

// Private data used by topology algorithm.

void \*data;

Status\_t (\*load)(struct \_RoutingModule \*);

Called when module is unset as current.

Status\_t (\*unload)(struct \_RoutingModule \*);

Status\_t (\*copy)(struct \_RoutingModule \* dest, const struct \_RoutingModule \* src);

Status\_t (\*release)(struct \_RoutingModule \* rm);

} RoutingModule\_t;

typedef struct \_RoutingFuncs {

Status\_t (\*pre\_process\_discovery)(struct \_Topology \*, void \*\*);

Status\_t (\*discover\_node)(struct \_Topology \*, struct \_Node \*, void \*);

Status\_t (\*discover\_node\_port)(struct \_Topology \*, struct \_Node \*, struct \_Port \*, void \*);

Status\_t (\*post\_process\_discovery)(struct \_Topology \*, Status\_t, void \*);

。。。。。。。

} RoutingFuncs\_t;

typedef struct \_DorTopology {

// mesh sizing

uint8\_t numDimensions;

uint8\_t dimensionLength[SM\_DOR\_MAX\_DIMENSIONS];

// per-dimension coordinate bounds. coordinates are assumed

// contiguous within these bounds

int8\_t coordMaximums[SM\_DOR\_MAX\_DIMENSIONS];

int8\_t coordMinimums[SM\_DOR\_MAX\_DIMENSIONS];

// for each dimension, true if toroidal

uint8\_t toroidal[SM\_DOR\_MAX\_DIMENSIONS];

// the configured number of toroidal dimensions

uint8\_t numToroidal;

// count of toroidal dimensions found till now

uint8\_t toroidal\_count;

// minimum number of SCs required to route the fabric correctly

uint8\_t minReqScs;

// maps the index of all dimensions to an index into only the

// toroidal dimensions

uint8\_t toroidalMap[SM\_DOR\_MAX\_DIMENSIONS];

// flat 2d bitfield indexed on switch indices marking whether

// the cycle-free DOR path is realizable between nodes

size\_t dorClosureSize;

uint32\_t \*dorLeft;

uint32\_t \*dorRight;

uint32\_t \*dorBroken;

uint32\_t closure\_max\_sws; //the switch count when the closures were computed

DorNode\_t \*datelineSwitch;

} DorTopology\_t;

typedef struct \_DorDiscoveryState {

uint8\_t nextDimension;

uint8\_t toroidalOverflow;

uint8\_t scsAvailable; // min of SC support of all fabric ISLs

DorDimension\_t \* dimensionMap[256]; // indexed by egress port

} **DorDiscoveryState\_t;**

typedef struct \_DorDimension {

uint8\_t ingressPort;

uint8\_t dimension;

int8\_t direction;

uint8\_t hyperlink;

cl\_map\_obj\_t portObj;

} DorDimension\_t;

typedef struct \_Node {

uint32\_t index; // index in linked list

。。。。。。

STL\_NODE\_INFO nodeInfo; // NodeInfo (for SA)

STL\_NODE\_DESCRIPTION nodeDesc; // NodeInfo (for SA)

STL\_SWITCH\_INFO switchInfo; // SwitchInfo (for SA)// TBD make a pointer and only allocate for

//switches **（十，1）**

STL\_PORT\_STATE\_INFO \*portStateInfo;

STL\_CONGESTION\_INFO congestionInfo;

uint8\_t path[64]; // directed path to this node

PORT \*lft;

。。。。。。

Port\_t \*port; // ports on this node

。。。。。。。

char\* nodeDescString; // Only used when nodeDesc is 64 chars (not null terminated).

cl\_map\_obj\_t mapObj; // Quickmap item to sort on guids

cl\_map\_obj\_t nodeIdMapObj; // Quickmap item to sort on guids

cl\_map\_obj\_t mapQuarantinedObj; // Quickmap item to sort on guids

。。。。。。

void \*routingData; // Routing specific data.

。。。。。。

struct \_Node \*next; // linked list pointer (for\_all\_nodes())

struct \_Node \*prev; // linked list pointer (for\_all\_nodes())

struct \_Node \*type\_next; // linked list pointer (NodeType based)

struct \_Node \*type\_prev; // linked list pointer (NodeType based)

struct \_Node \*sw\_next; // linked list pointer (tier based)

struct \_Node \*sw\_prev; // linked list pointer (tier based)

struct \_Node \*old; // Pointer to a copy of this node in an older topology. MAY BE NULL.

。。。。。。

} **Node\_t;**

typedef struct \_DorNode {

int8\_t coords[SM\_DOR\_MAX\_DIMENSIONS];

Node\_t \*node;

int multipleBrokenDims;

struct \_DorNode \*left[SM\_DOR\_MAX\_DIMENSIONS];

struct \_DorNode \*right[SM\_DOR\_MAX\_DIMENSIONS];

} DorNode\_t;

typedef struct \_Port {

uint8\_t index; // my port number

uint8\_t state; // portState (IB\_PORT\_DOWN ... IB\_PORT\_ACTIVE)

uint32\_t nodeno; // node index of other end

uint32\_t portno; // port index of other end

uint8\_t path[64]; // path to get to this port

PortData\_t \*portData;

} Port\_t;

typedef struct \_PortData {

uint64\_t guid; // port GUID

uint64\_t gidPrefix; // Gid Prefix

uint8\_t gid[16]; // Gid

。。。。。。

uint32\_t lid; // base lid

uint8\_t lmc; // LMC for this port

uint8\_t vl0; // VLs supported

uint8\_t vl1; // VLs actual

。。。。。。

STL\_PORT\_INFO portInfo; // PortInfo (for SA)

。。。。。。。。

cl\_map\_obj\_t mapObj; // quickmap item to sort on GUIDs

。。。。。。

struct \_Node \*nodePtr; // Ptr to the node record this port belongs.

。。。。。。

void \*routingData; // Private data used by topology algorithm.

uint16\_t routingCost; // Used for enhanced hypercube

int32\_t initWireDepth; // Initial wire depth to use for buffer calculations.

} PortData\_t;

**（十，1）**

typedef struct {

uint32 LinearFDBCap; /\* RO Number of entries supported in the \*/

/\* Linear Unicast Forwarding Database \*/

uint32 MulticastFDBCap; /\* RO Number of entries supported in the \*/

/\* Multicast Forwarding Database \*/

STL\_LID\_32 LinearFDBTop; /\* RW Indicates the top of the Linear \*/

/\* Forwarding Table \*/

/\* POD: 0 \*/

STL\_LID\_32 MulticastFDBTop; /\* RW Indicates the top of the Multicast \*/

/\* Forwarding Table \*/

/\* POD: 0 \*/

uint32 CollectiveCap; /\* RO Number of entries supported in the \*/

/\* Collective Table \*/

/\* Reserved in Gen1 \*/

STL\_LID\_32 CollectiveTop; /\* RW Indicates the top of the Collective Table \*/

/\* POD: 0 \*/

/\* Reserved in Gen1 \*/

uint8 PortGroupCap; /\* RO Specifies the maximum number of \*/

/\* entries in the port group table \*/

uint8 PortGroupTop; /\* RW The current number of entries in \*/

/\* port group table. \*/

struct { /\* RW (see STL\_ROUTING\_MODE) \*/

uint8 Supported; /\* Supported routing mode \*/

uint8 Enabled; /\* Enabled routing mode \*/

} RoutingMode;

union {

uint16 AsReg；

struct {

uint16 Enable: 1, /\* RW Enable/Disable AR \*/

uint16 Pause: 1, /\* RW Pause AR when true \*/

uint16 Algorithm: 3, /\* RW 0 = Random, 1 = Greedy, \*/

/\* 2 = Random Greedy. \*/

uint16 Frequency: 3, /\* RW 0-7. Value expands to 2^F\*64ms. \*/

uint16 LostRoutesOnly: 1, /\* RW. Indicates that AR should only be done \*/

/\* for failed links. \*/

uint16 Threshold: 3, /\* CCA-level at which switch uses AR. \*/

uint16 Reserved: 4

}s

} AdaptiveRouting；

SWITCH\_CAPABILITY\_MASK CapabilityMask; /\* RO \*/

。。。。。。。。

} PACK\_SUFFIX STL\_SWITCH\_INFO;

5、SC2SC数据结构

typedef struct {

STL\_PORTMASK IngressPortMask[STL\_MAX\_PORTMASK]; /// STL\_PORTMASK---uint64

// STL\_MAX\_PORTMASK---4

STL\_PORTMASK EgressPortMask[STL\_MAX\_PORTMASK];

STL\_SCSCMAP SCSCMap; /\* RW/HSPE \*/

/\* -SPE: POD: SCn\_to\_SCn (1-to-1) \*/

} PACK\_SUFFIX STL\_SCSC\_MULTISET;

typedef struct {

STL\_SC SCSCMap[STL\_MAX\_SCS]; /\* RW/HSPE \*/ //// STL\_SC --uint8，STL\_MAX\_SCS --32

/\* -SPE: POD: SCn\_to\_SCn (1-to-1) \*/

} PACK\_SUFFIX STL\_SCSCMAP;

static IXML\_FIELD SmDorRoutingFields[] = {

{ tag:"Debug", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, debug) },

{ tag:"UseEscapeVLs", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, escapeVLs) },

{ tag:"UseFaultRegions", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, faultRegions) },

{ tag:"Dimension", format:'k', subfields:SmDimensionFields, start\_func:SmDimensionStart },

{ tag:"WarnThreshold", format:'u', IXML\_FIELD\_INFO(SmDorRouting\_t, warn\_threshold) },

{ NULL }

};

typedef struct \_SmDorRouting {

uint8\_t debug;

uint8\_t dimensionCount;

uint8\_t numToroidal;

uint8\_t routingSCs;

uint32\_t warn\_threshold;

SmDimension\_t dimension[MAX\_DOR\_DIMENSIONS];

DorTop\_t topology;

uint8\_t escapeVLs;

uint8\_t faultRegions;

} SmDorRouting\_t;

dor->routingSCs = dor->escapeVLs ? 2 : 1;

dor->routingSCs = dor->escapeVLs ? 4 : 2;

1. 其他重要点
2. sm\_transition（）SM状态机转换





sm\_transition（）完成了两块功能：

状态的转换

计数器（counter）的更新

1. STL中的锁和信号量：

信号量：

topology\_sema

topology\_rcv\_sema

sa\_sema

锁：

old\_topology\_lock

new\_topology\_lock

tid\_lock

sa\_lock

handover\_sent\_lock

sm\_mcSpanningTreeRootGuidLock

sm\_datelineSwitchGUIDLock

1. 位图

Opa中的位图使用的是一个int型的数组来保存数据，与int数组操作中每个索引对应一个int型数据不同，位图操作中每个索引对应一个bit（位）。

static \_\_inline\_\_ void ijSet(uint32\* ijBitmap, int ij) {

ijBitmap[ij >> 5] |= 1 << (ij & 0x1f);

}

static \_\_inline\_\_ void ijClear(uint32\* ijBitmap, int ij) {

ijBitmap[ij >> 5] &= ~((uint32\_t)(1 << (ij & 0x1f)));

}

static \_\_inline\_\_ int ijTest(uint32\* ijBitmap, int ij) {

return ((ijBitmap[ij >> 5] & (1 << (ij & 0x1f))) ? 1 : 0);

}

Ij是索引，ijBitmap是位图的起始地址。上面三个操作分别为将位图中索引对应的”位”置1，将位图中索引对应的”位”清0，返回位图中索引对应的”位”的值。

1. 打印日志

cs\_log.h

IB\_LOG\_INFINI\_INFO\_FMT(\_\_func\_\_, "Number of dimensions: %d", dorTop->numDimensions);

用之前在cs\_log.h中将#define DEFAULT\_LOG\_MASK NONDEBUG\_LOG\_MASK改为：

#define DEFAULT\_LOG\_MASK (NONDEBUG\_LOG\_MASK|VS\_LOG\_INFINI\_INFO)

1. （非业务逻辑部分，与主线任务无关）有待学习的地方
   1. 锁的使用
   2. 内存池的实现
   3. Map数据结构的实现