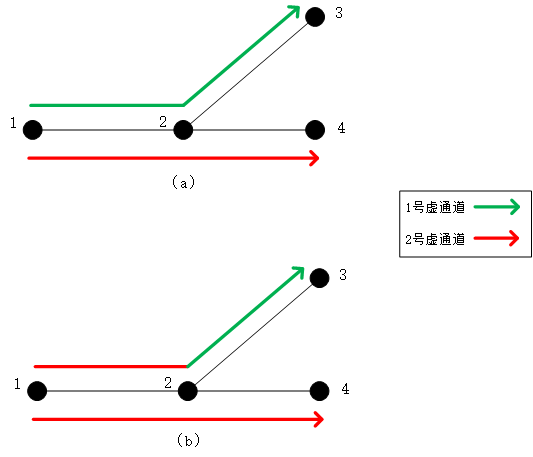
1. opa控制器程序（Storm Lake，STL）通过向交换机发送SC2SC指导交换机进行虚通道的切换

生成SC2SC的过程，STL与Dally的dor论文中的方法有较大差异：

Dally的论文中的虚通道切换概括来说，是已知数据包的目的节点和当前所在的节点，若要不发生路由死锁，求数据包下一跳应该使用的虚通道。

STL是已知数据包在当前交换机的输入端口、输入虚通道、输出端口，若要不发生路由死锁，求数据包下一跳应该使用的虚通道。



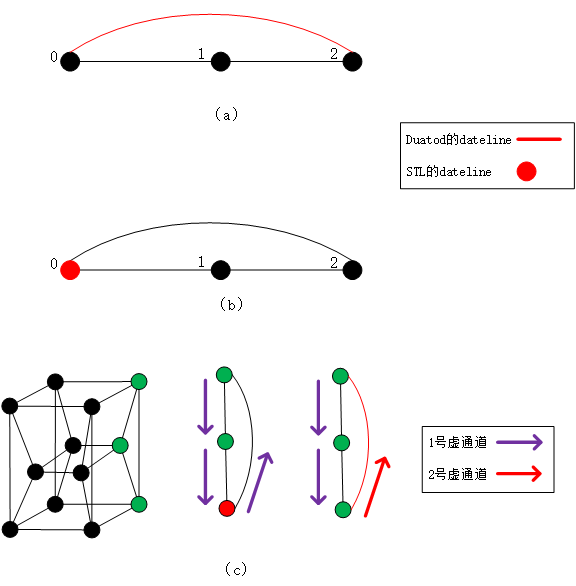
如上图，若已知目的节点，则可以像（a）一样给1->3和1->4分配不同虚通道。若不知道目的节点，则只能像（b）一样，在分叉点之后才可以给两条路径分配不同的虚通道。

1. 在torus中通过在dateline处切换虚通道避免维度内死锁

dateline的选取有两种方法：

方法一，选择一条特殊的链路作为dateline。

方法二，（STL采用的方法）选择一个特殊的节点作为dateline：



如上图所示：这两种dateline的选择方式在硅元内部对应不同的虚通道使用方式。STL选择一个特殊节点作为dateline，在硅元内部路由时只需使用一个虚通道。若选择一条特殊链路作为dateline，在硅元内部路由时则需要使用两条虚通道。

1. 关于拓扑发现流程的修改：

注：新算法称为dorbiu--电光以及dorbiusi--电光电，快速链路端口成为biu端口

先将sm所在节点添加到sm\_topop->node\_head链表上

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

-sm\_topop->routingModule->funcs.discover\_node（） （1）

-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（）； （2）

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

}

}

for\_all\_nodes （）在遍历sm\_topop->node\_head链表的时候，会不停的将新发现的节点添加到该链表上，链表随着循环不断增长，直至所有节点都被添加到该链表。

1. 主要检查节点与邻居的连接是否符合配置（维度，端口，环）
2. 发包探索当前节点的当前端口上是否连接有节点，有则加入sm\_topop->node\_head链表。
3. 如果新发现的节点是交换节点，则根据当前节点坐标生成新发现的邻居节点坐标

这里需要解决一个问题，新加入的biu端口（在配置文件中通过PortBiu指定）不能用来发现节点以及为新节点生成坐标。同时，还需要在节点的路由数据结构中加入一个指针，该指针负责指向其兄弟节点。将指针指向兄弟节点的工作需要在（3）中完成。当然，新加入的biu端口肯定是不符合dor配置的规则的，所以要跳过（1）。

对数据结构\_DorNode修改：

typedef struct \_DorBiuNode {

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

Node\_t \*node;

int multipleBrokenDims;

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

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

//--------------------zp start----------------//

Node\_t \*brother; //the node connected by biu link

//--------------------zp stop-----------------//

} DorBiuNode\_t;

在拓扑发现及坐标生成时，跳过biu端口：

sm\_setup\_node（）中：

If（路由算法为dorbiu或dorbiusi）{

If（邻居节点未发现过&&当前端口不是biu端口）{

正常发现的流程（程序原本的代码）

}else if（邻居节点已经被发现过）{

核对之前记录的该节点的信息，与该次获得的信息是否匹配（程序原本代码），主要是opa的安全性检查，防止有黑客向网络注入虚假消息，欺骗控制器。

}else if（邻居节点未发现过&&当前节点是biu端口）{

return VSTATUS\_OK； //跳过biu端口上的拓扑发现

}

}

|  |  |  |
| --- | --- | --- |
|  | 该邻居节点未被发现过 | 该邻居节点已被发现过 |
| 当前端口是biu端口 | 跳过（return） | 核对节点信息 |
| 当前端口不是biu端口 | 执行正常发现流程 | 核对节点信息 |

discover\_node\_port（）中：

代码A：

if (neighborNodep == NULL) {

IB\_LOG\_ERROR0("Failed to find neighbor node");

//-----------------------zp start----------------------//

if(portp->index==state->portbiu){

return VSTATUS\_OK;

}

//-----------------------zp stop------------------------//

return VSTATUS\_BAD;

}

代码B

//-----------------------------zp start-------------------------//

if(portp->index==state->portbiu&&portp->portno==state->portbiu){

((DorBiuSiNode\_t \*)nodep->routingData)->brother=neighborNodep;

((DorBiuSiNode\_t \*)neighborNodep->routingData)->brother=nodep;

return VSTATUS\_OK;

}

//-----------------------------zp stop--------------------------//

代码B负责将两个通过biu链路相连的节点通过指针brother互相连接起来。

代码A则是处理这种情况，当sm\_setup\_node（）通过当前节点（点a）的biu端口找到邻居节点（点b）时，该邻居节点有可能并不在链表sm\_topop->node\_head中（对应上面：邻居节点未发现过&&当前节点是biu端口），这时传入参数neighborNodep为空，此时我们暂且跳过。当对链表sm\_topop->node\_head的遍历进行到点b时（这是必然会发生的事），再执行代码B，将两个节点连接起来。

discover\_node（）中：

if (neighborType == NI\_TYPE\_SWITCH) {

//---------------------zp start---------------------//

DorBiuSiDiscoveryState\_t\* state=(DorBiuSiDiscoveryState\_t\*)context;

if((p->index==state->portbiu)&&(neighbor\_portno==state->portbiu)){

continue;

}

//---------------------zp stop----------------------//

}

若当前节点的当前端口是biu端口，跳过检查。

1. 关于组划分流程的添加：

Dor中是不会对硅元内交换节点进行分组的，而分组信息是我的dorbiusi算法在生成LFT的时候所必须的。所以我在post\_process\_discovery（），一个在拓扑发现完成之后、LFT生成之前执行的函数，中添加一个分组函数:

\_record\_communication\_node（）。

同时需要对数据结构\_DorNode进行修改：

typedef struct \_DorBiuSiNode {

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

Node\_t \*node;

int multipleBrokenDims;

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

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

//--------------------zp start----------------//

Node\_t \*brother; //the node connected by biu link

Node\_t \*comNode; //the communication node

//--------------------zp stop-----------------//

} DorBiuSiNode\_t;

配置文件中添加了一个参数NumBiuSi指定了每个硅元中有多少个分组。

并通过一个数组指定在NumBiuSi取不同值的时候，硅元内那些点是属于一组，以及通讯节点是哪个点：

//-------------------------------zp start------------------------//

//define a array to map communication node to node

int communication\_node\_map[6][12][4]={

。。。。。。

{ //numBiuInSi=4,communication node =(0,0,0),(1,2,1),(1,0,0),(0,2,1)

{0,0,0,0},//(0,0,0)

{100,1,0,0},//(1,0,0)

{1,0,2,1},//(0,0,1)

{101,1,2,1},//(1,0,1)

{10,0,0,0},//(0,1,0)

{110,1,0,0},//(1,1,0)

{11,0,2,1},//(0,1,1)

{111,1,2,1},//(1,1,1)

{20,0,0,0},//(0,2,0)

{120,1,0,0},//(1,2,0)

{21,0,2,1},//(0,2,1)

{121,1,2,1}//(1,2,1)

},

。。。。。。

};

//-------------------------------zp stop ------------------------//

\_record\_communication\_node（）里值得注意的地方是：

该函数依靠将sm\_topop->switch\_head链表中节点坐标与数组communication\_node\_map进行比对，找到每个节点对应的通信节点。但是拓扑发现结束之后，节点坐标中的数值并不一定都是正数，比如一个2x3的torus网络，拓扑发现结束之后，各点坐标分别是：（0，0）、（1，0）、（0，1），（1，1）、（0，-1）、（1，-1）。

可以使用（坐标值%维度长度+坐标值）%维度长度，将副坐标值转换为相应的正坐标值。

直接比较两坐标值是否相同的时候，也可以不转换，直接判断（坐标值1-坐标值2）%维度长度==0，如果为求余结果为0，则两坐标值是等价的。

1. 关于LFT生成的流程修改：

此处主要涉及两个函数get\_port\_group（）、\_is\_path\_realizable（），一个数据结构

typedef struct \_DorTopology {

。。。。。。

uint32\_t \*dorLeft;

uint32\_t \*dorRight;

uint32\_t \*dorBroken;

。。。。。。

} DorTopology\_t;

Dorbiusi路由算法中需要比较从源节点到目的节点，“三段路由”的路径长度与传统dor的路径长度。“三段路由”指的是：从源节点到目的节点的通讯节点的兄弟节点，从该兄弟节点经快速链路到目的节点的通讯节点，从目的节点的通讯节点到目的节点。

这样，我们需要一个数据结构来存储从任意一点到任意一点的路径长度。所以我对dorLeft、dorRight、dorBroken进行了改造。这三个数本来是作为位图使用的，表示从任意一点i到任意一点j是否是连通的（dorBroken），若连通，则需要i向左转发（dorLeft）还是向右转发（dorRight）。当然，具体是在i哪个纬度进行转发，还是通过比较i、j的坐标值来决定的。

这里不把dorLeft、dorRight、dorBroken作为位图使用，而是仅仅作为普通的uint32数组，dorLeft、dorRight每个元素保存的是从节点i到节点j，向左转发（dorLeft）的跳步数、向右转发（dorRight）的跳步数。

此处需要将之前的位图函数：ijBiuSet（）、ijBiuClear（）、ijBiuTest（）的实现进行修改，保证其功能不变。并添加新函数：ijBiuGet（），功能是获取从i到j的跳步数。

此处还涉及到对\_process\_swIdx\_change（）函数的修改（因为这个 函数没有通过调用ijBiuSet（）、ijBiuClear（）、ijBiuTest（）操作位图，而是直接通过位运算操作位图），这个函数的功能是在网络中交换机发生变化（如有些交换机被替换掉了，即从sm\_topop->switch\_head链表中被除去的时候），在位图中将相应的行和列删除（比较有意思），这样就不用重新生成一个位图了。

(插一个位图的图示)

准备工作已经完成，然后就是在get\_port\_group（）函数中进行相应的修改了，这一段的主要功能就是比较从源节点到目的节点，“三段路由”的路径长度与传统dor的路径长度，选择较短的路径，生成对应的输出端口。

六、关于SC2SC表生成的流程修改：

此处主要添加了一个操作，为新加入的biu端口生成SC2SC表，此处虚通道的切换规则正好可以借鉴程序中原本就有的scscNoChg（下快速链路时使用）和scscPlus1（上快速链路时使用）。见函数\_generate\_scsc\_map（）。