倾斜低轨星座下的备份路由算法改进

摘要：

由于倾斜低轨星座拓扑结构可以抽象成网格模型，[1]中的备份路由算法没有考虑倾斜低轨星座拓扑结构的特殊性，在网格模型场景下的主动通知开销过大。本文改进了其算法，使其主动通知的开销为一个较小的常数，大大降低了通知开销。仿真结果表明，改进后的备份路由算法的主动通知开销降低很多，同时由于数据包变小，要处理的信息也变少，路由收敛速度也得到了一定的提升。

介绍：

随着低轨卫星制造和发射成本降低，加上多颗低轨卫星组成的低轨星座能够对全球实现全面覆盖，对于某些偏远地区，尤其是地面网络无法覆盖的区域，低轨卫星具有重要的战略意义，因此低轨卫星成为近些年来的研究热点，著名的低轨星座有starlink以及oneweb。但是由于低轨卫星的链路连接不稳定的问题一直存在，设计一种高效、弹性的路由算法是一项巨大的挑战。传统的动态路由协议OSPF采用LSA实现链路信息的同步，通过Hello报文超时来判断链路是否中断，而Hello报文的超时时间达到几十秒。因此在空间网络当中，如果直接应用OSPF将会导致大量的时间进行路由收敛，如果链路经常中断更会直接导致路由不可用。

对此[1]中提出了针对单链路中断的备份路由算法，对每个目的地提前计算好一条备份路径，当发生链路中断时采取主动通知的方式切换成备份路径，使得能够在几百毫秒内重新恢复全局路由。由于是主动通知的方式，并且通知的内容是受影响的网段，所以当规模增大时，通知开销就显得比较大。因此本文提出了下面的针对网格模型场景下的新的备份路由算法来降低通知开销，以适应大规模场景。

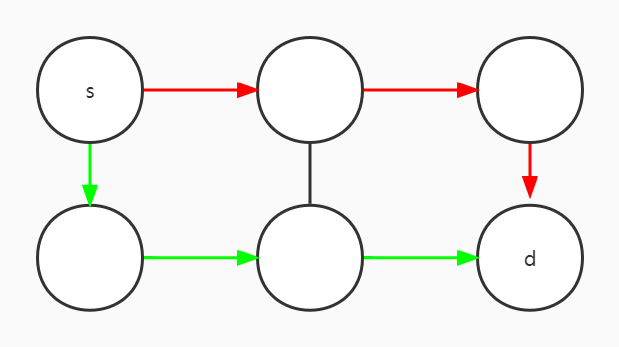
新的备份路由算法：

拓扑图G=(V,E),其中V代表节点集合，E代表链路集合，节点编号为1,2...n，我们假设中断的链路不影响整个拓扑的连通性。

备份路由算法步骤：

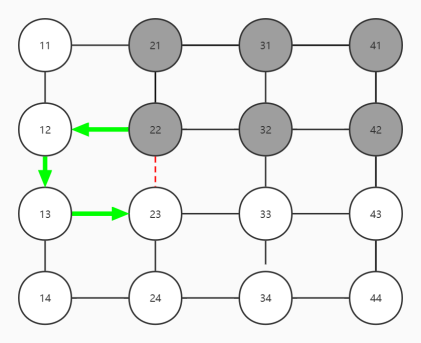
1. 对E中的每一条链路el，假设它发生了中断，el两端节点设为k和i，其中k>i,以编号大的卫星节点k建立最短路径树T(k)。
2. 将i所在的子树分离，设为T(i)，并将其涂成黑色，剩下的节点涂成白色。
3. 遍历每个黑色节点，找到最短的路径重新连接节点k和i，假设最终选定的黑色节点是s，对应的白色节点是d。
4. 从节点s回溯到i，找到一条路径s,s1,s2...st,i，从节点d回溯到k，找到一条路径d,d1,d2,...dm,k。那么就找到了一条最短路径重新连接两个不同颜色的树，这条路径就是备份路径。之后计算路径上的节点对应的备份下一跳，i的备份下一跳是st，st的备份端口是st-1，依次往下一直到s，s的备份下一跳设为d。同理k到d的备份下一跳也这样设定。
5. 每个节点将自己的的备份下一跳存储起来，和对应的中断链路建立映射。
6. 每个节点如果收到中断链路的信息后查找自己对应的备份下一跳，如果有的话，就切换成备份路由，否则继续使用主路由。

可以证明找到的这条备份路径一定可以恢复全局路由，首先需要知道的一点是在网格模型场景下的最短路径有只有两种情况，一是先向左或者向右走，再向上或者向下到达目的地，二是先向上或者向下走，再向左或者向右到达目的地。图一展示了s到d的最短路径两种情况，分别是红色路径和绿色路径。

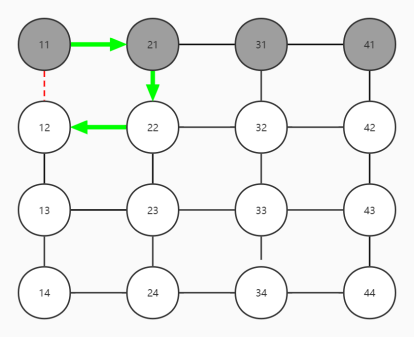


图一

按照上面的算法步骤可以将全局拓扑划分成两个不同颜色区域，这两个颜色区域都是矩形或者两个矩形拼接而成的图形，这样的话同一颜色区域内部的路由所经过的结点的颜色都是一致的，即同一颜色区域内是按照主路由转发的。图二展示了节点22和节点23之间发生了链路中断后的涂色情况，其中绿色路径代表找到的备份路径，其中的白色区域是两个矩形拼接而成，黑色区域是单个矩形。图三展示了节点11和节点12发生链路中断后的涂色情况，绿色路径代表备份路径，黑色区域和白色区域都是单个矩形。



图二



图三

证明：不妨假设黑色节点需要发往白色节点一些数据

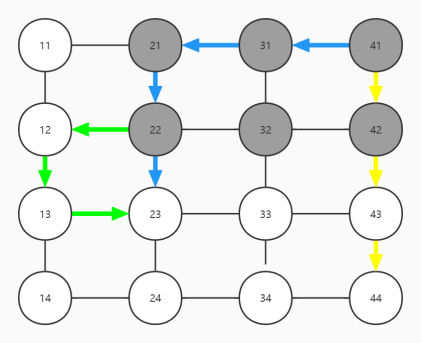
发往白色节点的数据包经过的路径可以分为两类：

一是按照主路由发包，并且在黑色区域内没有遇到任何一个使用了备份路由的节点，显然主路由是没有回环并且一定也是可以发到白色区域的，由于同一颜色区域一直使用主路由，那么到达白色区域后是可以顺利到达目的地的。

二是遇到了至少一个使用了备份路由的节点，那么到达使用了备份路由路由器的数据包将会按照那条备份路径转发数据包，最终也会到达白色区域。

白色节点发往黑色节点同理。

图三展示了黑色节点41发往白色节点44和23所经过的路径，同样假设节点22和23之间发生了链路中断，绿色路径代表找到的备份路径。黑色节点41发往白色节点44的路径是41->42->43->44，由于在黑色区域内没有遇到使用备份路由的节点所以是按照主路由发包的，即41->42->43，白色节点43在收到数据后沿着主路由递交给白色节点44。黑色节点41发往白色节点23的路径是41->31->21->22->12->13->23，在黑色区域内是沿着主路由41->31->21->22发包的，由于在黑色区域内遇到了使用了备份路由的黑色节点22，黑色节点22会沿着备份路径22->12将数据递交给白色节点12，而不是按照主路由递交给白色节点23，白色节点12沿着主路由12->13->23将数据递交给白色节点23。

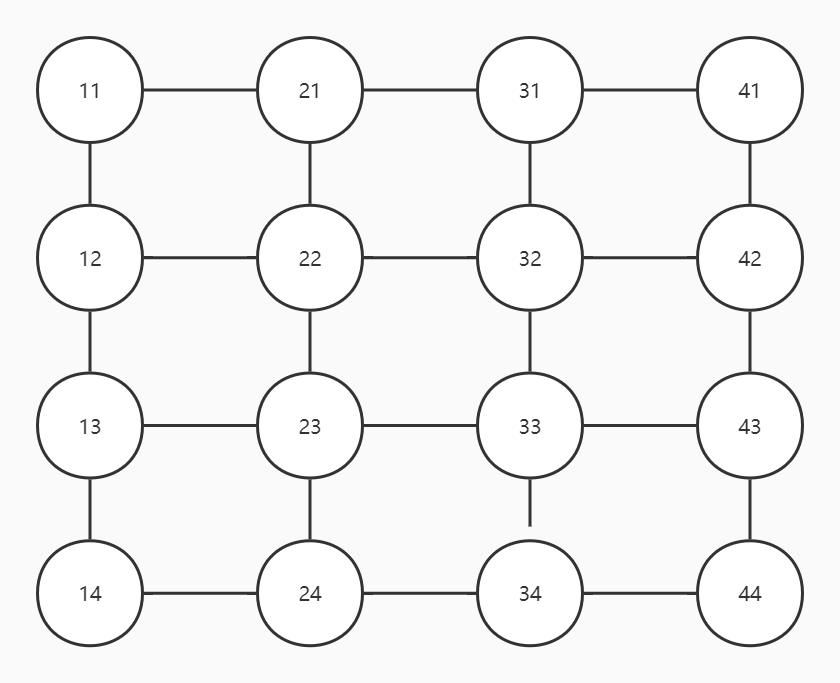


图四

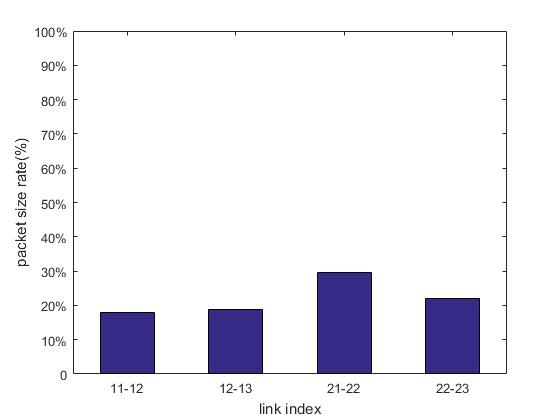
仿真结果：

我们采用Docker搭建了图四的场景，用OVS模拟链路的中断与恢复，和之前的算法在数据包大小和收敛时间两方面进行了比较。考虑到场景的对称性，我们中断恢复了其中的的四条链路节点11和12之间的链路，节点12和13之间的链路，节点21和22之间的链路，节点22和23之间的链路。

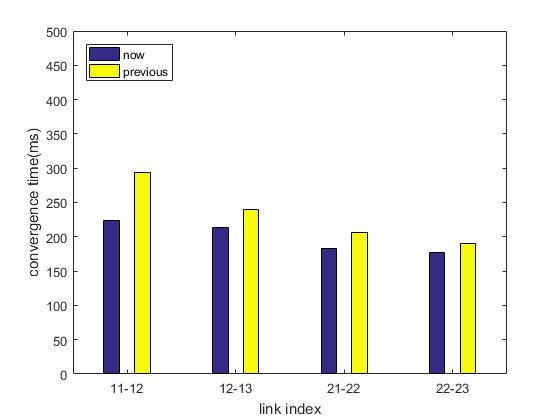
仿真结果表明我们的算法需要主动通知的消息只需要包含中断链路两端节点编号即可，而这个数据是固定大小17字节。相反[1]中的算法主动通知的是受影响的所有网段，随着规模的增大，这个数据量会显著地增大，图五表明即使是在16节点的小规模场景下，我们的数据包大小也远比[1]中的小，随着场景规模的增大这个优势会愈发明显。并且由于[1]中数据包的较大，处理数据包中的信息所花费的时间相对较长，这会导致重收敛时间变长，而且我们的数据包较小，要处理的信息较少，收敛速度也相对较快，规模较大的情况下我们的算法更有优势。图六表明我们的算法在路由重收敛速度方面也得到了一定的提升。



图五



图六



图七

