北京航空航天大学计算机学院

硕士学位论文中期检查报告

论文题目：面向低轨卫星网络的轻量化链路状态路由协议设计与实现

专　　业：计算机技术

研究方向：计算机网络

研 究 生：单乾

学　　号：ZY2206311

指导教师：王志远

北京航空航天大学计算机学院

2024 年 6 月 20 日

目 录

面向低轨卫星网络的轻量化链路状态路由协议设计与实现

1 论文工作计划

本章主要介绍论文开题时指定的论文工作计划，包括研究目标、主要研究内容、研究进度。

1.1 论文研究目标

本研究针对低轨卫星网络拓扑动态性强、负载分布不均衡的挑战，设计并实现一种面向低轨卫星网络的轻量化链路状态路由协议，提升低轨卫星网络中的端到端传输性能。此外，还需在仿真软件与模拟平台中开展性能测试与实际部署，并与已有的星间路由协议进行比较。

1.2 论文主要研究内容

为实现上述研究目标，本论文将深入研究低轨卫星网络的拓扑特征，设计并实现一种轻量化的链路状态路由协议。主要内容如下：

(1) **研究内容一：局部化细粒度星间链路状态传播机制**。每颗卫星精确掌握自身附近一定跳数范围内的链路连通性及负载信息，同时根据拓扑形状可预测性大致推测较远处的链路连通性信息，并基于这些信息分布式地进行路由决策，从而在保证端到端传输效果的前提下，实现较小的通告开销。其包含下列子内容。

1. **基于可控传播距离的链路状态通告**：通过设置链路状态的传播距离，实现链路状态的局部化通告。
2. **面向链路负载的链路状态语义扩展**：将链路状态所包含的信息从单纯的

通断信息扩展为通断信息与负载状态信息，使链路状态语义的粒度得到扩展与细化。

3. **基于链路负载的链路状态传播触发：**设计触发机制，使本协议快速感知链路连通性变化（即链路断开、恢复） 与链路负载变化（即转发队列排队长度增加、减少），并进行相应通告。

(2) **研究内容二：低时间复杂度星间路由计算机制：**考虑到低轨卫星网络的拓扑形状可预测性，若不存在链路故障，则整个拓扑呈网格形。可以利用此特性设计不同于陆地互联网的路由计算机制，减小路由计算的时间开销。其包含下列子内容。

1. **基于目的卫星位置的下一跳计算机制：**设计一种路由计算机制，当一颗卫星在计算路由表时，对于任意目的卫星，无需收集链路状态信息，只需根据两者在空间中的相对位置即可快速确定通往目的卫星的下一跳出口方向。此机制可快速计算出到目的卫星的下一跳地址，并可确保端到端路径的跳数最小。
2. **基于广度优先搜索的最短路径计算机制：**设计另一种路由计算机制，利用链路状态数据库（Link-State Database, LSDB）中保存的链路状态信息，通过广度优先搜索算法（Breadth first search, BFS）首先计算到所有目的卫星的最短路径，并根据最短路径并生成路由表。

(3) **研究内容三：性能验证与协议实现**。本研究首先进行性能验证，在确定本研究所提协议性能优于现有方法后，开展协议实现工作。其包含下列子内容：

1. 基于仿真软件的性能验证：选取相应的基线方法，在仿真软件中实现基线方法与本协议的功能，比较不同方法的传输表现。

2. 基于模拟平台的协议实现：在实验室搭建的基于容器技术的模拟平台中，完整实现一套真实计算机系统中可用的路由协议，包含用户态模块、内核态模块及两模块之间的消息传递机制。

3. 基于模拟平台的协议测试：在模拟平台中实现基线方法，比较不同方法

的传输表现。

这里要说明的是，在模拟平台中的协议测试是有必要的。这是因为以往的研究中，在提出星间路由协议后往往通过纯软件仿真 (simulate)，并不能完全复现操作系统内核网络协议栈及无线信道的全部逻辑与流程，且模拟软件中的代码并不能直接移植到真实的计算机系统中。与之相对的，模拟 (emulate) 则通过物理设备或虚拟化技术构建实际的网络，对真实网络进行复刻与近似。本研究即在通过仿真验证协议性能后，进一步在基于容器技术的模拟平台上实现真实可用的协议代码。

此外要说明的是，本研究的研究内容二有必要在且只能在模拟平台中进行实现。这是因为在以 OMNeT++[5]为代表的离散事件仿真软件中，计算路由表的过程往往被视为一个原子事件，因此计算路由表的耗时难以进行准确仿真，而在较大规模的低轨卫星网络中，计算路由表的耗时经测试可能达到数百微秒，不能忽略不计。

本研究整体内容如图 1 所示。

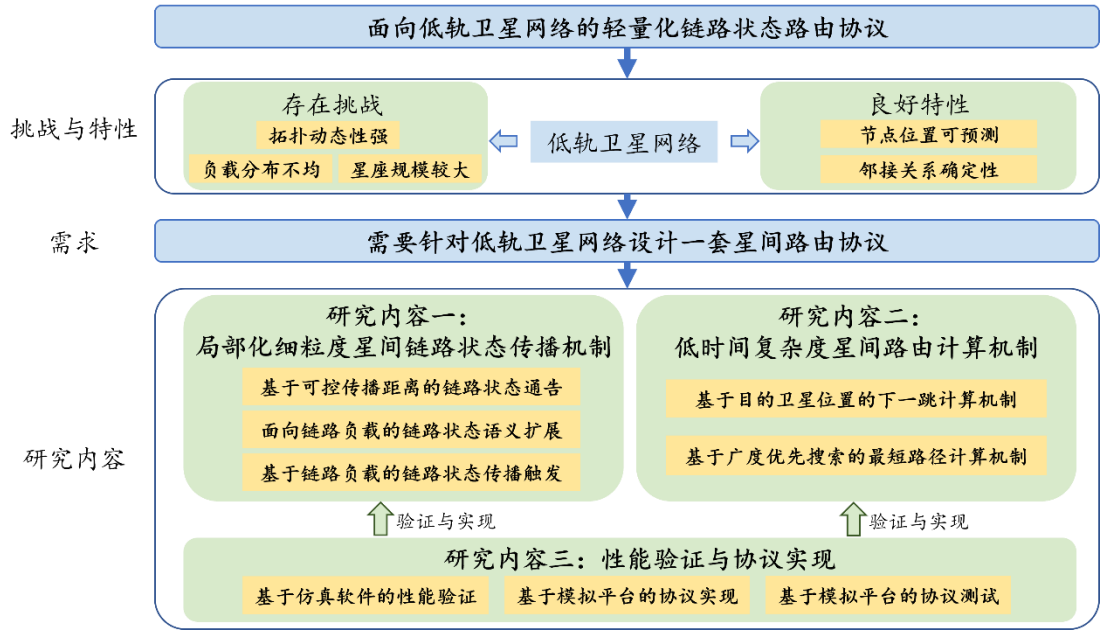


图 1 本研究整体内容

其中，研究内容一已写成论文《Routing in LEO Satellite Networks: How Many

Link-State Updates Do We Need?》并发表于国际学术会议 IEEE Satellite 2023。

1.3 论文研究进度

本论文研究进度如表 1 所示。其中标注“√”的表示已完成、“O”表示正在进行中、“×”表示未完成。

表 1 研究进度

时间	研究内容		完成
2022.9~2023.2	文献调研		√
2023.3~2023.6	研究内容一	在仿真软件中开发	√
2023.7~2023.8	研究内容三的子内容 1	仿真软件中进行性能验证	√
2023.9~2023.12	研究内容三的子内容 2	对模拟平台二次开发	√
2024.1~2024.6	研究内容一	在模拟平台中开发	√
2024.7~2024.12	研究内容二	在模拟平台中开发	O
2025.1~2025.3	研究内容三的子内容 3	模拟平台中进行测试	×
2025.4~2025.5	撰写毕业论文		×

2 已经完成的工作

本章介绍本研究目前已完成的工作，主要包括局部化细粒度星间链路状态传播机制的设计、低时间复杂度星间路由计算机制的设计、局部化细粒度星间链路状态传播机制在仿真软件中的性能验证、局部化细粒度星间链路状态传播机制在模拟平台中的实现。

2.1 局部化细粒度星间链路状态传播机制的设计

本部分对应研究内容一。以往的动态星间路由协议大都要求链路状态需要全网洪泛、所有卫星需要保证 LSDB 是全局同步的，这带来较大的通告开销。然而考虑到低轨卫星网络拓扑形状的可预测性，整个网络拓扑的大致形状是可以轻易推断的，因此卫星可以不关心一条距离自己较远的链路的实时状态，只需实时关

注自身附近一定范围内链路状态即可。为此，本研究设计了一套局部化细粒度星间链路状态传播机制（Localized Fine-grained Link-state Dissemination, LoFi），具体包括基于可控传播距离的链路状态通告、面向链路负载的链路状态语义扩展、基于链路负载的链路状态传播触发三部分。

2.1.1 基于可控传播距离的链路状态通告

以往的动态星间路由协议大都要求链路状态需要全网洪泛、所有卫星需要保证 LSDB 是全局同步的，这带来较大的通告开销。然而考虑到低轨卫星网络拓扑形状的可预测性，整个网络拓扑的大致形状是可以轻易推断的，因此卫星可以不关心一条距离自己较远的链路的实时状态，只需实时关注自身附近一定范围内链路状态即可。即每颗卫星通过链路状态交互，精确掌握自身附近一定跳数范围内的拓扑形状，同时根据拓扑形状可预测性大致推测较远处的拓扑形状。在这种情况下，由于各卫星并不掌握全局实时信息，于是所做出的路由决策可能并非全局最优，但随着数据报文的传递，各卫星分布式地进行路由决策，将能够使数据报文沿着相对较优的路径转发。基于上述假设，本研究设计了基于可控传播距离的链路状态通告机制。

为了实现将链路状态限制在局部传播，本研究拟采用对链路状态通告（Link-State Advertising, LSA）设置生存时间（Time to Live, TTL）的方式。假设初始 TTL 为n，当卫星检测到直连的链路状态发生变化时，会生成一个 TTL 为n的 LSA 并向邻居传播，传播中 TTL 逐跳递减直至减为 0。则各卫星会将其直接相连的链路状态信息传播给n跳范围内的邻居（在无链路故障时共$2n^2-2n+1$颗卫星），由此达到减小协议通告开销的目的。$n=1$的情况如图 2。如图 2 左侧所示，当卫星S_i和S_j之间的链路L_{ij}断开时，S_i和S_j各自都事件触发式地生成 LSA，其中分别包含两颗卫星直接相连的 4 条链路状态信息。该信息被封装在 LSU 报文中向外传播 1 跳，则所有紫色的卫星都能实时掌握L_{ij}的链路状态。如图 2 右侧所示，$n=1$时对于卫星S_k来说，其可以收到 1 跳范围内的邻居即S_i、S_q、S_l、S_p发出的 LSA，

这些 LSA 涵盖了图 2 右侧的所有绿色链路，即任一卫星可掌握 2 跳范围内的实时链路状态。

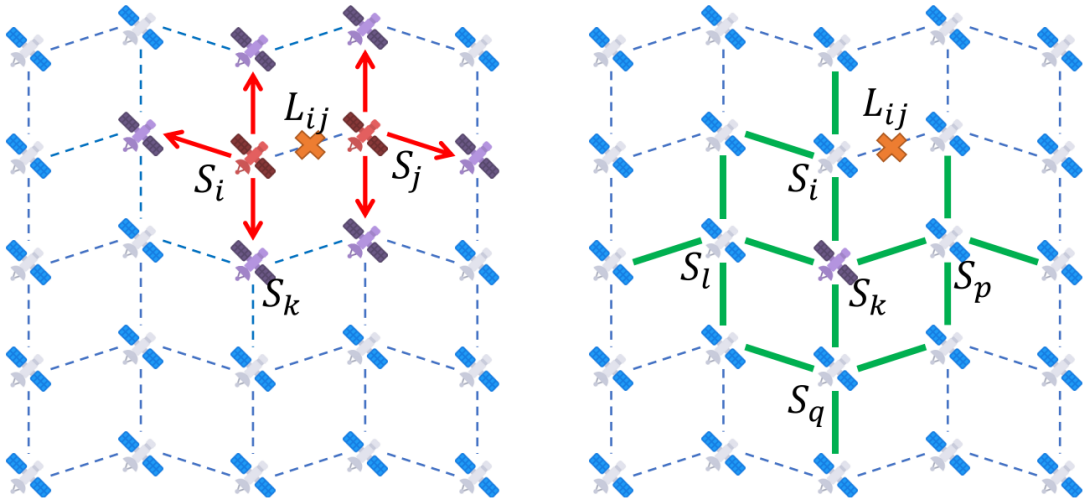


图 2 链路状态传播机制示意

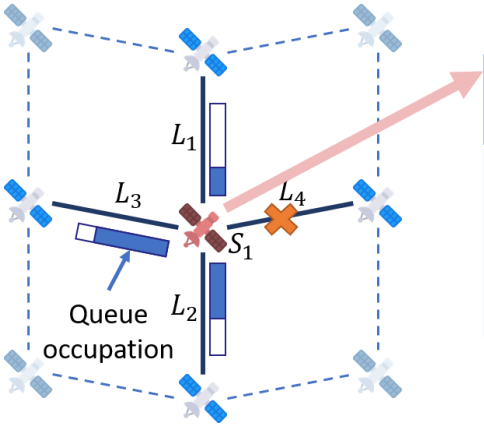
由此可知，初始 TTL 值$n \in \{0,1,2,3,\ldots\}$的设置将是本研究的一个重要参数。若初始 TTL 设为n，则每颗卫星可获知$n+1$跳范围内的实时链路状态。随着n的增大，卫星所掌握的信息增多，做出的路由决策可能更优，但协议控制报文的通信开销也会更大，所以必须在其间做出权衡。

2.1.2 面向链路负载的链路状态语义扩展

在传统的链路状态路由协议中，LSA 一般并不包含链路负载信息，其 Link Data 字段包含的一般是在协议配置阶段设置的固定链路花费 (cost) ，除非修改路由协议配置，否则此字段的值一般不会变化。这导致以往的动态星间路由协议中，LSDB 中大多只储存链路通断状态，不储存链路负载状态，于是基于 LSDB 做出的路由决策往往未考虑到负载状况。在此情况下，负载均衡机制大多作为一个单独的模块存在，负责对做出的路由决策进行修正。这还导致卫星之间除了交互链路状态信息之外，又需单独定义链路负载报文并交互，也增大了协议控制报文通信开销。

本研究的链路状态信息不只包含链路通断情况，也包含链路负载情况，即链

路负载状态与链路通断状态共同在局部进行传播，从而使本研究所提的协议内生地支持负载均衡功能。其中链路负载情况一般由该链路的转发队列占用率定义。如图 3 所示，一颗卫星S_1生成的 LSA 包含与它直接相连的所有星间链路（Inter-satellite Link, ISL），即(L_1, L_2, L_3, L_4)的链路状态，包括各 ISL 的通断情况与负载情况。



Generated link-state message

Advertising satellite	ISL	Link-state data	
		connectivity	load
S_1	L_1	true	0.3
	L_2	true	0.6
	L_3	true	0.8
	L_4	false	--

图 3 链路状态语义扩展示例

为了使 Router-LSA 能够携带扩展的链路状态语义，本研究将 Router-LSA 中的 Link Data 字段定义为对应链路的当前时刻花费值$Cost_{link}$。该花费值综合考虑传播时延与排队时延，即

$$Cost_{link} = d_{link}^{Propa} + d_{link}^{Queue}$$

其中d_{link}^{Propa}、d_{link}^{Queue}分别代表该链路的传播时延与排队时延，其中d_{link}^{Propa}由卫星根据拓扑形状可预测性计算得知，即

$$d_{link}^{Propa} = \frac{L_{link}}{c}$$

其中L_{link}是根据拓扑形状可预测性得到的两颗卫星的空间距离，c是光速。而d_{link}^{Queue}与转发队列占用长度与链路带宽有关，即

$$d_{link}^{Queue} = \frac{q(t)_{link}}{BW}$$

其中$q(t)_{link}$表示进行链路状态局部通告时转发队列的占用长度，为时变值；BW为链路带宽，为已知的固定值。

通过此种方式，每颗卫星可由收到的 LSA 得到局部范围内所有链路的传输时延，同时根据拓扑形状可预测性推算较远处链路的传输时延，并以此选择总时延最短、负载较小的路径进行数据包转发。

2.1.3 基于链路负载的链路状态传播触发

链路负载状态需要“及时”地通告给邻居卫星，即上述链路状态局部传播机制除了在链路通断状态变化时触发，还需要在负载状态满足一定条件时触发。

传统的动态路由协议中，负载状态往往是定时通告，即每隔Δt时间通告一次。此种定时通告的方式有较大局限性，即如果负载状态在一段较长的时间内保持基本稳定，这段时间内定时通告的内容基本相同，会带来较多的协议通信开销；此外如果Δt时间内链路负载状态发生了剧烈变化，负载均衡状态则无法及时更新。

针对上述局限性，本研究拟采用事件触发的方式。考虑一颗卫星，其有 4 个转发队列，分别记作$i \in \{1,2,3,4\}$。设$\{Q_i: \forall i = 1,2,3,4\}$表示各个转发队列的最大长度（可以是该转发队列能容纳的最大字节数或最大数据报数）。每个转发队列中的待转发数据包总量$q_i(t)$是时变值，表示对应链路的实时负载。在本研究中，只有当转发队列占用长度超过一定阈值时，才会触发链路状态的传播。为此，本研究使用参数$\delta \in [0,1]$表示监测链路负载的精度。在任意时刻t，卫星的所有转发队列中，如果有任意一个转发队列i满足

$$|q_i(t) - \tilde{q}_i| \geq Q_i \cdot \delta$$

则触发链路状态局部传播。其中$\tilde{q}_i$ 表示记录的上次局部传播时的队列占用长度，$i \in \{1,2,3,4\}$。

可见$\delta \in [0,1]$是本研究的第二个重要参数。当其设置为一个较小的值时，队列占用长度每发生较小变化即会触发链路状态局部传播，使得卫星对链路负载状态较敏感，能够达到较好的负载均衡效果，但带来的通告开销也会较大；设置为一个较大值时则反之。于是δ值的设置将成为本研究的另一处权衡。

2.1.4 小结

此部分主要介绍了本研究所提的局部化细粒度星间链路状态传播机制 LoFi。该机制的主要参数为元组(n,δ)。其中，$n \in \{0,1,2,\dots\}$表示链路状态（包括链路通断信息和链路负载信息）的传播范围；$\delta \in [0,1]$表示对链路负载的敏感程度，δ越大则对链路负载越不敏感、但协议控制开销也越小，δ越大则反之。通过调整(n,δ)的值，本研究所提的局部化细粒度星间链路状态传播机制可泛化为多种链路状态路由协议。例如，当参数设置为$(\infty,1)$时，本机制退化为经典的开放最短路径优先协议（Open Shortest Path First, OSPF）[1]。

2.2 低时间复杂度星间路由计算机制的设计

本部分对应研究内容二。在较大规模的低轨卫星网络中，以 OSPF 为代表的传统链路状态路由协议计算一次路由表的耗时经测试可能达到数百微秒。考虑到低轨卫星网络的拓扑动态性，路由表会被频繁地重计算，带来较大的计算时间开销。然而上述路由协议忽略了低轨卫星网络的邻接关系确定性。具体来说，低轨卫星网络的拓扑形状整体呈网格形，且每个节点可能的邻居集合是确定的。如图 4 所示，对于一颗轨道编号为x，轨内编号为y的卫星，其所有邻居编号都已确定。

针对此特性，本研究针对低轨卫星网络拓扑设计了一套低时间复杂度星间路由计算机制，具体包括两种机制：基于目的卫星位置的下一跳计算机制、基于广度优先搜索的最短路径计算机制。

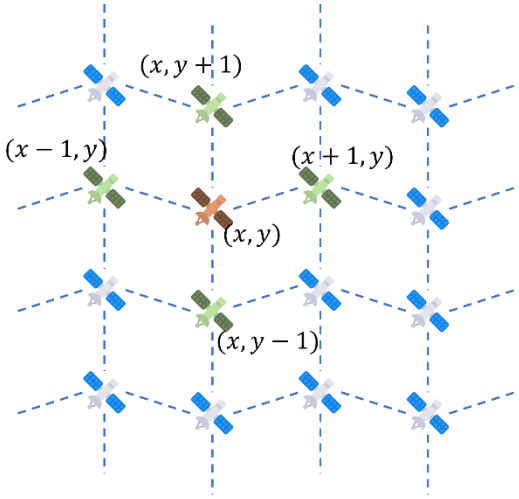


图 4 网格形拓扑示意

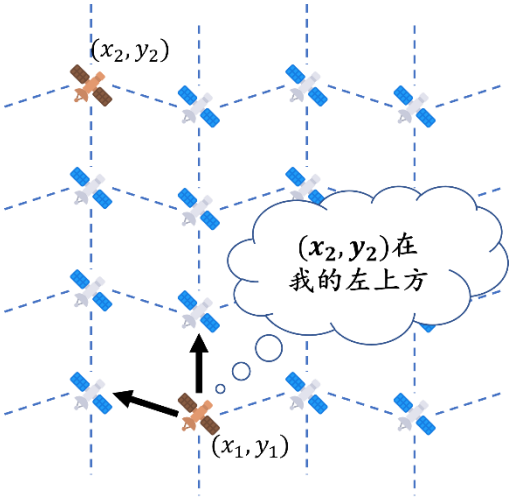


图 5 基于目的卫星位置的下一跳计算示意

2.2.1 基于目的卫星位置的下一跳计算机制

此机制在计算路由表时，不考虑所收集的链路状态信息具体内容（即不考虑拓扑中边的实时连通性与负载情况），而是直接根据目的卫星的相对自身所处的位置，即目的卫星的编号与自身编号的关系进行判断，直接得出通往目的卫星的下一跳方向，如图 5 所示。具体的路由表计算流程如算法 1 所示。

算法 1 基于目的卫星位置的下一跳计算算法

Algorithm 1: 基于目的卫星位置的下一跳计算算法

Input: 当前卫星编号(x_1, y_1)，轨道数M，每个轨道卫星数N，星座类型T

Output: 路由表T_{new}，其中保存着到每个目的卫星的下一跳

1: $\{n_x^+, n_x^-, n_y^+, n_y^-\} \leftarrow$ 轨道编号增大/减小、轨内编号增大/减小方向的邻居，$T_{new} \leftarrow \emptyset$

2: **for each** (x_2, y_2) in $\{1, \dots, M\} \times \{1, \dots, N\}$ **do**

3: // 确定轨间方向，需考虑星座类型

4: **if** $T =$ Walker-star **then**

5: **if** $x_2 > x_1$ **then**

6: $T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_x^+\}$

else if $x_2 < x_1$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_x^-\}$

end if

else if $T =$ Walker-delta **then**

if $x_2 > x_1$ **and** $x_2 - x_1 \geq \lfloor M/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_x^-\}$

else if $x_2 > x_1$ **and** $x_2 - x_1 \leq \lfloor M/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_x^+\}$

else if $x_2 < x_1$ **and** $x_2 - x_1 \geq \lfloor M/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_x^+\}$

else if $x_2 < x_1$ **and** $x_2 - x_1 \leq \lfloor M/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_x^-\}$

end if

end if

// 确定轨内方向，无需考虑星座类型

if $y_2 > y_1$ **and** $y_2 - y_1 \geq \lfloor N/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_y^-\}$

else if $y_2 > y_1$ **and** $y_2 - y_1 \leq \lfloor N/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_y^+\}$

else if $y_2 < y_1$ **and** $y_1 - y_2 \geq \lfloor N/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_y^+\}$

else if $y_2 < y_1$ **and** $y_1 - y_2 \leq \lfloor N/2 \rfloor$ **then**

$T_{new}[(x_2, y_2)] \leftarrow T_{new}[(x_2, y_2)] \cup \{n_y^-\}$

end if

end for

return T_{new}

算法 1 遍历星座中的所有卫星，对每个卫星按规则执行判断。具体来说，若当前卫星为(x_1, y_1)，对于每颗目的卫星(x_2, y_2)：

1. 首先在 3~20 行，根据x_1与x_2的关系，判断下一跳应该是向x增大的方向还是减小的方向。其中根据星座类型是 Walker-star 星座还是 Walker-delta 星座又有所区别。这是因为 Walker-star 星座的所有轨道升交点赤经分布在$[0, 180°]$范围内，即在地球一侧的所有卫星在向北极方向运动，而另一侧的所有卫星在向南极方向运动，于是这两侧存在着一条缝隙，缝隙两侧的轨道不会建立轨间链路。而 Walker-delta 星座的所有轨道升交点赤经分布在$[0, 360°]$范围内，即所有轨道升交点绕地球均匀分布，并不存在缝隙，不同轨道上轨内编号相同的卫星间可以组成环。两种星座类型的示意图如图 6 所示。

2. 之后在 21~30 行，根据y_1与y_2的关系，判断下一跳应该是向y增大的方向还是减小的方向。

3. 此时在路由表中存储着通往(x_2, y_2)的所有候选下一跳，将其返回即可。

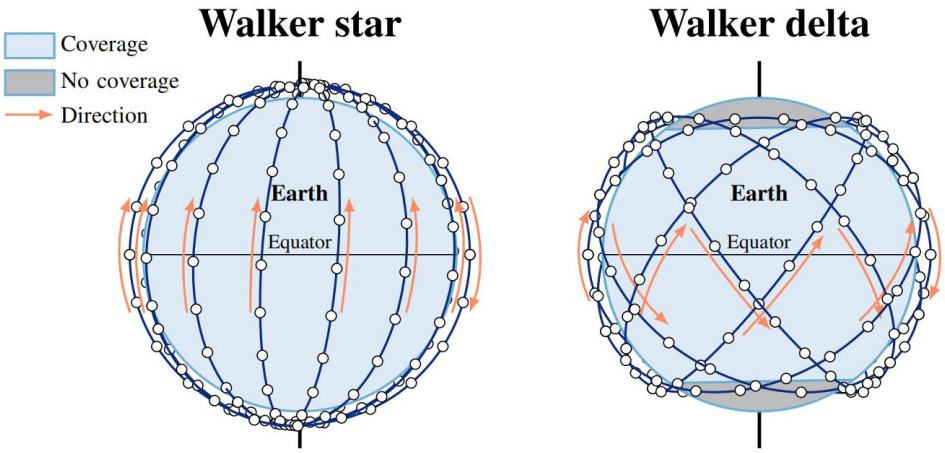


图 6 两种星座类型示意图[8]

在上述流程中，每次判断的时间复杂度为$O(1)$，则总时间复杂度为$O(MN)$且常数极小，因此执行耗时很短，且可以确保从当前节点到目的节点的转发路径跳数最小。要特别说明的是，上述判断规则虽然时间开销很小，但由于没有考虑链路状态信息，因此不能保证转发路径的总时延最小，也不能保证在链路故障率较

高时的数据包传输性能。

2.2.2 基于广度优先搜索的最短路径计算机制

此机制在计算路由表时，考虑收集到的链路状态信息的具体内容，根据低轨卫星网络中所有星间链路的传播时延，可以计算出传播时延最短路。本机制与传统链路状态路由协议的主要区别在于计算最短路的算法不同，传统链路状态路由协议主要使用堆优化的 Dijkstra 算法[2]来计算路由表，而本机制则使用基于广度优先搜索的最短路径算法来计算路由表。具体的路由表计算流程如算法 2 所示。注意，此处输入的拓扑图G实际上就是卫星所持有的链路状态数据库的形式化表示，同时为了计算得到传播时延最短路，拓扑图中的边权统一定义为星间链路的传播时延。

算法 2 基于广度优先搜索的最短路径算法

Algorithm 2: 基于广度优先搜索的最短路径算法

Input: 源节点c, 拓扑图G

Output: 路由表T_{new}，其中保存着到每个目的卫星的下一跳

搜索集合$S \leftarrow \{c\}$，搜索队列$Q \leftarrow \emptyset$，距离数组$d \leftarrow \{\infty\}$，$T_{new} \leftarrow \emptyset$
Q.push(c)，$d[c] \leftarrow 0$
while Q is not empty **do**
    $u \leftarrow Q$.pop()
    for each $v \in u$.neighbors **do**
        if $v \notin S$ **then**
            $S \leftarrow S \cup \{v\}$
            $d[v] \leftarrow d[u] + G[u][v]$
            Q.push(v)
            if $u = c$ **then**
                $T_{new}[v] \leftarrow \{v\}$

12: **else**

13: $T_{new}[v] \leftarrow T_{new}[u]$

14: **end if**

15: **else if** $v \in S$ **and** $d[v] = d[u] + G[u][v]$ **then**

16: $T_{new}[v] \leftarrow T_{new}[v] \cup T_{new}[u]$

17: **else if** $v \in S$ **and** $d[v] > d[u] + G[u][v]$ **then**

18: $T_{new}[v] \leftarrow T_{new}[u]$

19: **end if**

20: **end for**

21: **end while**

22: **return** T_{new}

算法 2 采用广度优先搜索的思想逐层搜索路径，可以搜索得到从源节点c到A的所有最小跳数路径，并会选出这些最小跳数路径中总传播时延最小的路径。具体来说：

1. 初始化部分（第 3 行），将源节点c加入搜索集合S和搜索队列Q，初始化距离数组d，设置所有节点的初始距离为∞，并将源节点c的距离设置为 0，初始化新的路由表T_{new}为空。

2. 主循环部分（第 4 行~20 行）：只要搜索队列Q不为空，就从队列中弹出一个节点u进行处理。其又细分为：

1) 第一次访问邻居节点v（第 7 行~14 行）：如果v未被访问过，将v加入搜索集合S。之后更新距离数组，将$d[v]$设置为$d[u] + G[u][v]$。之后更新路由表：如果u是源节点c，将v直接作为下一跳节点；否则，继承u的下一跳节点信息。之后将v加入搜索队列Q。

2) 邻居节点v已被访问，且当前路径长度等于已有路径长度（第 15 行~16 行）：如果$d[v] = d[u] + G[u][v]$，更新路由表，将v的下一跳节点集合并上u的下一跳节点。

3) 邻居节点v已被访问，且当前路径长度小于已有路径长度（第 17 行~19 行）：如果$d[v] > d[u] + G[u][v]$，更新距离数组，将$d[v]$设置为$d[u] + G[u][v]$。之后更新路由表，将v的下一跳节点信息更新为u的下一跳节点。

3. 最后返回路由表T_{new}。

在算法正确性方面，当低轨卫星星座满足特定条件时可以证明：传播时延最短路一定在最小跳数路径集合中[7]。而算法 2 能正确选出会选出这些最小跳数路径中总传播时延最小的路径。由此可知，在星间链路故障率处于较低水平时，算法 2 选出的路径即为传播时延最短路。

在时间复杂度方面，算法 2 的初始化时间复杂度为$O(V)$，其中V是节点的总数；每个节点至多入队一次并出队一次，因此队列操作的时间复杂度为$O(V)$；每个节点会遍历邻居节点，而拓扑图G呈网格型，每个节点最多有 4 个邻居，因此遍历操作的时间复杂度为$O(E)$，其中E是边的总数。因此总时间复杂度为$O(V + V + E) = O(V + E)$。而传统链路状态协议使用的堆优化的 Dijkstra 算法的时间复杂度方面，初始化时间复杂度为$O(V)$；每次从堆中取出节点的操作时间复杂度是$O(\log V)$，每个节点最多执行一次，因此堆操作时间复杂度为$O(V\log V)$；每条边最多会导致一次堆的更新，堆的更新操作时间复杂度是$O(\log V)$，因此邻居更新总时间复杂度为$O(E\log V)$，总时间复杂度为$O(V + V\log V + E\log V) = O((V + E)\log V)$。

考虑到对于一个有M个轨道、每个轨道有N颗卫星的星座，$V = MN$，$E \approx 2V$，则算法 2 的时间复杂度为$O(MN)$，堆优化的 Dijkstra 算法的时间复杂度为$O(MN\log(MN))$，因此本研究中基于广度优先搜索的最短路径计算机制的时间复杂度优于传统链路状态路由协议。同时此机制考虑到了链路状态信息，有一定能力应对可能存在的链路故障。

2.2.3 小结

此部分介绍了本研究设计的两种低时间复杂度的星间路由计算机制：基于目

的卫星位置的下一跳计算机制和基于广度优先搜索的最短路径计算机制。前者通过卫星位置直接判断下一跳方向，时间复杂度为$O(MN)$且常数极小，但没有考虑实时链路状态；后者基于广度优先搜索计算最短路径，时间复杂度为$O(MN)$，优于传统的 Dijkstra 算法，且考虑了实时链路状态。

2.3 局部化细粒度星间链路状态传播机制在仿真软件中的性能验证

本部分对应研究内容三的第一个子内容：基于仿真软件的性能验证，在OMNeT++平台[5]中进行软件仿真以验证性能。OMNeT++是一个可扩展的、模块化的、基于组件的 C++仿真库和框架，可以实现离散事件仿真。其中提供了 inet 套件用于仿真各网络协议栈的功能。

本研究基于 inet4.4 套件，编写仿真代码实现了本研究所提协议的功能，并构建了了一个类铱星星座 (6 个轨道，每个轨道 11 颗卫星，轨道高度 780km) 用于测试。为方便开展性能验证，该星座的星间链路带宽设置为 10Mbps、转发队列总长度设置为 1000 个数据包、传输实验中的每个数据包大小设置为 1KB。此外，在仿真时还考虑了突发性的星间链路故障。具体来说，本研究假设链路故障的发生服从参数为λ的泊松分布，并假设每次链路故障事件持续 5 秒钟。通过调整参数λ，可以实现不同的链路故障率。

2.3.1 链路故障事件的生成

考虑到低轨卫星网络的拓扑动态性较强，本研究需要生成星间链路可能存在的链路故障，具体来说，需要根据给定的星间链路故障率，依概率生成相应的星间链路故障/恢复事件。设星间链路在单位时间内处于故障的时间为F，则对于给定的星间链路故障率R，应有：

$$R = E(F)$$

其中E表示期望。同时，本研究中假设单位时间内链路故障发生的次数X服从泊松分布，即$X \sim P(\lambda)$，其中λ待求。又考虑到在实际的星间激光链路中，发射器与接收器安装在同一组镜面中[6]，则当星间链路发生故障时，链路两侧的卫星可以

同时检测到，并调整激光天线指向尝试恢复故障，同时重计算路由表以确保转发性能，于是可以将每次链路故障的持续时间设为一个确定的参数T，则有：

$$F = X \cdot T$$

又由于星间链路仅在正常工作时有可能故障，不会在故障期间在此延长故障，即在$1-R$时间内即会发生X次故障，且X满足：

$$E(X) = \lambda = R/T$$

则链路每正常工作一单位时间，有：

$$E(X) = \frac{R}{T(1-R)} = \lambda$$

其中T为确定的参数、R为输入值，则解得λ。于是可得链路单次正常工作的持续时间（即从一次故障恢复到另一次故障发生的时间间隔）Δt服从指数分布：

$$\Delta t \sim Exp(\frac{T(1-R)}{R})$$

根据上述指数分布可模拟链路故障事件的发生，对每条星间链路，其链路故障生成流程如图 7 所示。

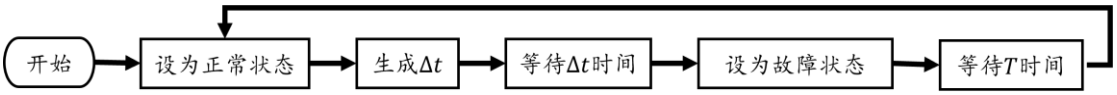


图 7 链路故障事件生成流程

2.3.2 轻负载下的性能验证

为了在轻负载下验证本研究所提的局部化细粒度星间链路状态传播机制，本研究随机选取源-目的节点对，起点与终点的曼哈顿距离为 6 跳。源节点向目的节点以每秒 100 个数据包的速率向目的节点发送 UDP 数据包。在上述工作负载下，可以确保网络中不存在拥塞，因此端到端传输时延主要由传播时延构成。本部分主要关注的性能指标为控制开销（协议控制报文引起的传输开销）、数据包丢包率和数据包的端到端时延，并与基线方法 OPSPF[3]开展对比。

图 8 展示了在轻负载条件下，链路故障率分别为 5%和 15%时，本协议所提的局部化细粒度星间链路状态传播机制 LoFi 与基线方法 OPSPF 的性能。具体来

说，本部分固定$\delta = 1$（即无负载感知），并将n取值范围设定为$\{1, 2, \ldots, 7\}$。图 8 左侧展示了在不同协议下控制开销和数据包丢包率之间的关系，图 8 右侧展示了控制开销和端到端延迟之间的关系。可以观察到：

1.对于 LoFi$(n, 1)$，随着n增加，通告开销增大，丢包率、时延呈变好趋势，且收益呈边际递减趋势。在实验条件下，当$n \geq 4$时，数据包传输性能的边际提升几乎为 0。

2. 链路故障率为 5%时，LoFi(3,1)的丢包率、端到端时延与 OPSPF 基本一致，且通告开销比 OPSPF 减少 78%。

3. 链路故障率为 15%时，LoFi(4,1)的通告开销比 OPSPF 减少了 66%，且丢包率与 OPSPF 基本一致，而端到端时延仅比 OPSPF 增加了 3.9%。

这表明在轻负载条件下，**局部化细粒度星间链路状态传播机制可大幅降低通告开销，同时实现较好传输效果。**

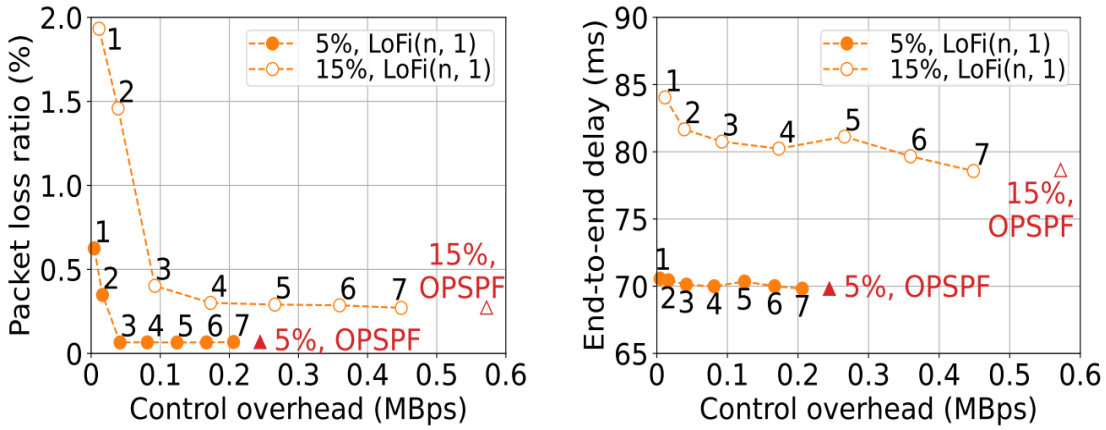


图 8 轻负载条件下的性能

2.3.3 重负载下的性能验证

为了在重负载下验证本研究所提的局部化细粒度星间链路状态传播机制(LoFi)，本研究选取 4 对有着重叠链路的选取源-目的节点对。每个源节点的发送速率为每秒 500 个数据包，链路故障率固定为 10%。在上述工作负载下，可以确保网络中存在数据包排队与拥塞。本部分主要关注的性能指标为控制开销、数据包丢包率和数据包的端到端时延，并与基线方法 OPSPF[3]和 ELB[4]开展对比。

图 9 展示了在重负载条件下，LoFi$(n, 0.05)$、LoFi$(n, 1)$、OPSPF 和 ELB 的控制开销、数据包丢包率和端到端时延。可以观察到：

1. LoFi$(n, 0.05)$与 LoFi$(n, 1)$相比，数据包丢包率和端到端时延均显著下降。

2. LoFi(n, δ)的端到端传输性能随着n的增加而变好。在实验条件下，当$n = 3$时可以以较小的控制开销获得几乎最优的数据包丢包率和端到端时延。

3. 与 ELB 相比，LoFi(3,0.05)可以减少 94.4%的数据包丢包、67.6%的端到端时延、85.7%的控制开销。

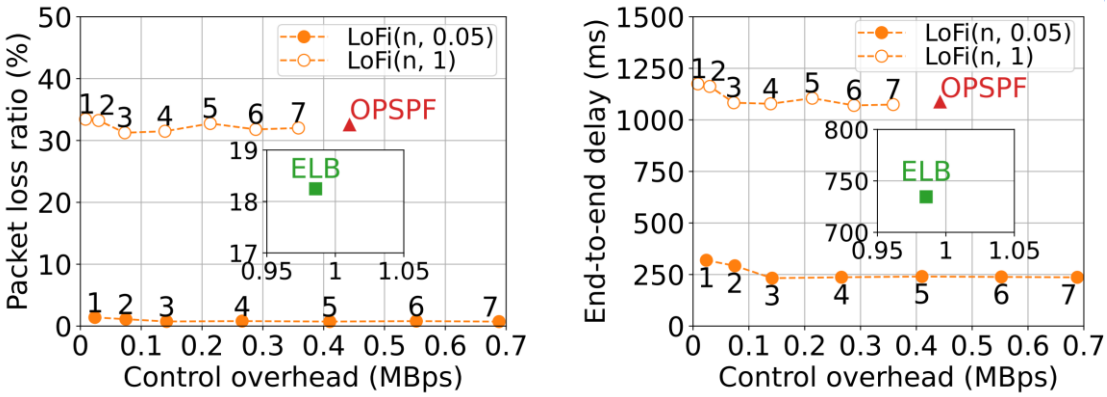


图 9 重负载条件下的性能

此外，本研究还比较了重负载条件下，链路故障率分别为{0,5%, 10%, 15%, 20%}时，LoFi(4,0.05)、LoFi(4,0.2)和 ELB 的性能，如图 10 所示。可以观察到：

1. 随着链路故障率增加，LoFi(4,0.05)、LoFi(4,0.2)和 ELB 的控制开销、数据包丢包率、端到端时延均会变差。

2. LoFi(4,0.05)、LoFi(4,0.2)在控制开销、数据包丢包率、端到端时延均优于 ELB。

3. LoFi(4,0.05)、LoFi(4,0.2)相比，随着δ增大，LoFi 对链路负载的变化变得更不敏感，控制开销会变小，但数据包丢包率、端到端时延均会变大。

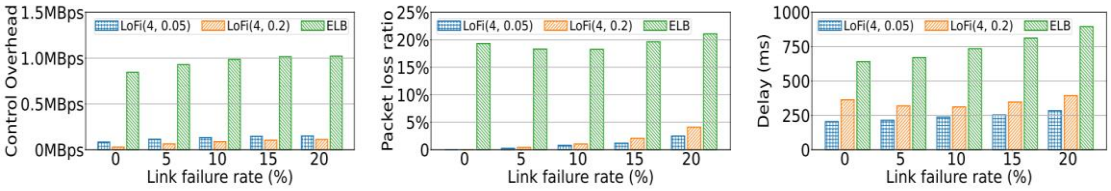


图 10 重负载条件下的性能

以上结果表明在重负载条件下，本研究提出的**局部化细粒度星间链路状态传播机制具有更好网络流量负载均衡能力，的能够实现更优的控制开销、数据包丢包率与端到端时延。**

2.4 局部化细粒度星间链路状态传播机制在模拟平台中的实现

本研究在通过仿真验证协议性能后，进一步在基于容器技术的模拟 (emulate) 平台上实现真实可用的协议代码，目前已实现局部化细粒度星间链路状态传播机制的全部三个子内容。本部分首先简要介绍所使用的模拟平台框架，之后着重介绍基于可控传播距离的链路状态通告、基于链路负载的链路状态传播触发在模拟平台中的实现。

需要说明的是，在具体实现中，面向链路负载的链路状态语义扩展与基于链路负载的链路状态传播触发这两个子内容的耦合程度较高，因此在介绍基于链路负载的链路状态传播触发时一并介绍。

2.4.1 模拟平台框架

本研究使用的模拟平台框架如图 11 所示，其主要由 Docker 容器网络与宿主机上的外部控制程序组成。具体来说：

1. 在 Docker 容器网络中，使用 Docker 容器模拟卫星，用户定义的 Docker 网桥模拟星间链路，通过为 Docker 容器的虚拟网卡配置排队规则实现对星间链路带宽与时延的模拟。Docker 容器的用户空间中运行 FRRouting 路由套件，内核版本为 Linux6.8.7。其中，FRRouting 是一个适用于 Linux 平台的免费开源互联网路由协议套件，主要使用 C 语言编写而成，其实现了 OSPF、BGP 等多种网络

协议，可与本地 Linux/Unix IP 网络堆栈无缝集成。

2. 在外部控制程序中，接口故障生成模块依概率生成突发性链路通断事件，并打开/关闭相应 Docker 容器的虚拟网卡，空间位置计算模块定时模拟每个容器所处的空间位置，并更新虚拟网卡排队规则的时延配置。

本研究基于上述模拟平台，对路由套件与内核 IPv4 模块进行二次开发。具体来说，局部化细粒度星间链路状态传播机制（研究内容一）中，基于可控传播距离的链路状态通告需要对路由套件进行二次开发，面向链路负载的链路状态语义扩展与基于链路负载的链路状态传播触发需要对 IPv4 模块进行二次开发；低时间复杂度星间路由计算机制（研究内容二）则主要对路由套件进行二次开发。

最终，路由套件将负责处理协议控制报文、维护链路状态数据库、计算路由表并下发至内核，IPv4 内核模块将负责根据路由表转发数据报文、感知链路负载情况、将负载信息上交至路由套件。

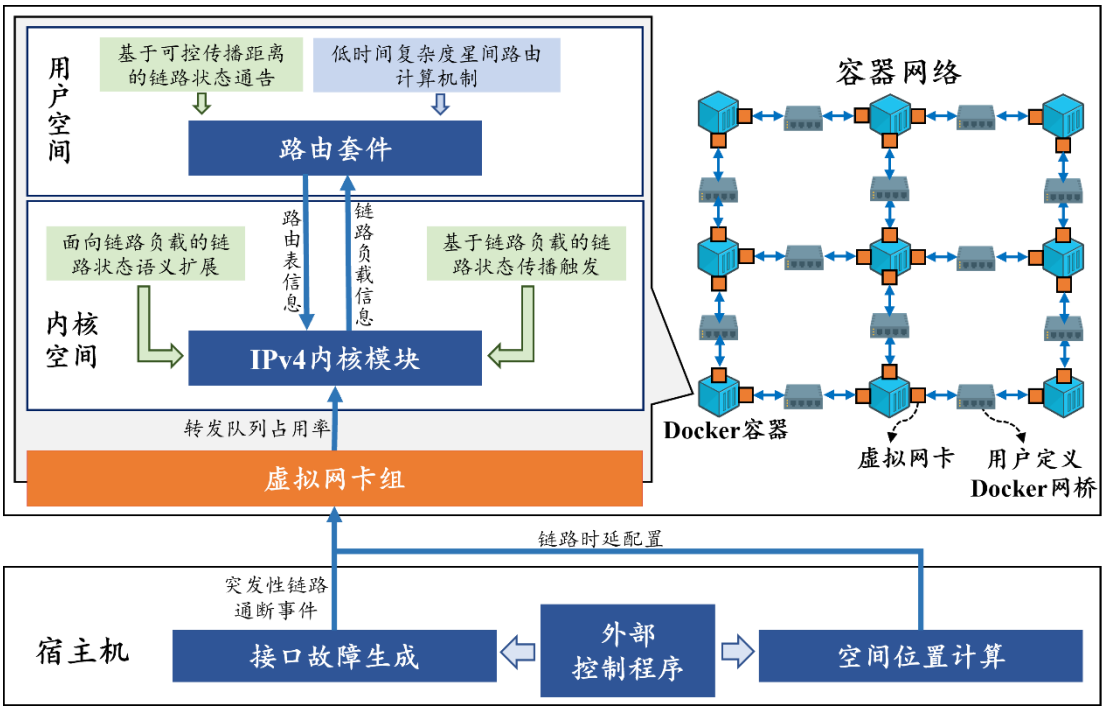


图 11 模拟平台框架示意

2.4.2 基于可控传播距离的链路状态通告在模拟平台中的实现

本研究中基于可控传播距离的链路状态通告通过在路由套件中硬编码实现。首先，本研究使用 RFC2328[1]中的路由器链路状态通告（Router-LSA）描述链路状态及作为协议控制报文，其格式定义如表 2 所示。一条 Router-LSA 描述与路由器（即卫星）相连的所有链路的状态，其中的一个 (Link ID, Link Data) 元组表示一条链路状态信息。在 RFC2328 中，Router LSA 的第 23 字节为全 0，本研究将其修改为 LSA 的 TTL 字段。

表 2 Router-LSA 格式定义

byte1	byte2	byte3	byte4
LS age		Options	LS type
Link State ID			
Advertising Router			
LS sequence number			
LS checksum		length	
flags		TTL	# links
Link ID			
Link Data			
…			

规定控制报文格式后，基于可控传播距离的链路状态通告的具体执行流程如图 12 所示。对于邻居卫星发来的 LSA，或自身因直连的链路状态发生变化而产生的 LSA，依次进行如下操作：

1. 检查 LSA 的校验和，若检查通过则继续执行，否则结束；

2. 检查 LSA 的 TTL 字段是否大于等于 1，若大于等于 1 则继续执行，否则跳转到 4；

3. 将 LSA 的 TTL 字段减 1，并更新校验和，之后向所有符合要求的邻居发送更新 TTL 与校验和之后的 LSA；

4. 根据收到的 LSA，查询自身 LSDB 中持有的该 LSA 对应实例。若自身 LSDB 未持有对应实例或自身持有的实例比收到的 LSA 更旧，则继续执行，否

则结束；

5. 根据收到的 LSA 更新自身 LSDB，同时会触发路由表重计算；

6. 结束。

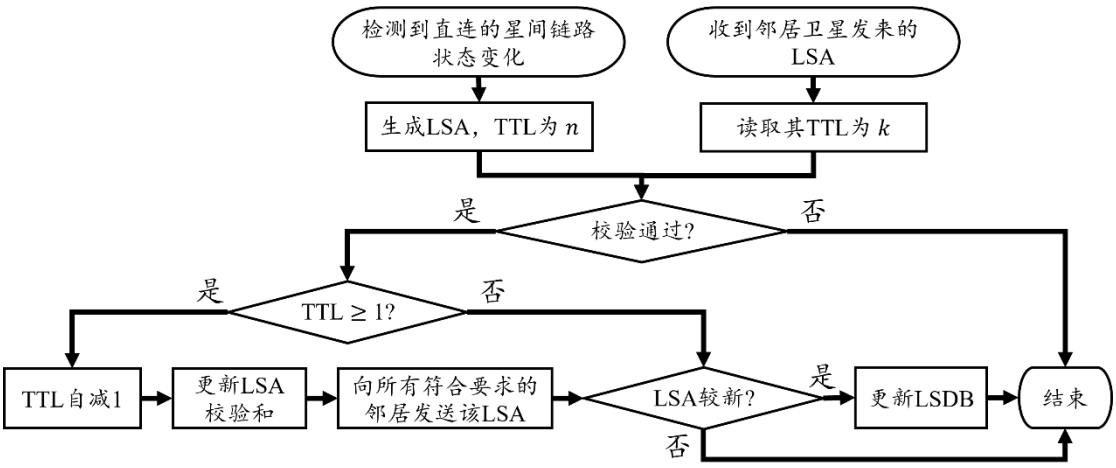


图 12 基于可控传播距离的链路状态通告执行流程

2.4.3 基于链路负载的链路状态传播触发在模拟平台中的实现

本研究中基于链路负载的链路状态传播触发的实现包括三部分：编写可加载 Linux 内核模块、编写内核模块与路由套件的通信机制、在路由套件中硬编码。下面分别介绍这三部分。

(1) 可加载 Linux 内核模块的实现

可加载 Linux 内核模块技术是一种允许开发者动态地向内核添加功能的机制，内核模块可以在运行时被加载或卸载而无需修改 Linux 内核本身，便于开发与实现。本部分的可加载内核模块需要获取链路负载情况并根据链路负载情况实现事件触发。

首先介绍如何获取链路负载情况，即面向链路负载的链路状态语义扩展与基于链路负载的链路状态传播触发的耦合部分。在模拟平台中，Docker 容器的每个虚拟网卡都配置有一个排队规则，在 Linux 内核中的数据结构分别为 struct net_device 与 struct Qdisc。其中：

1. struct net_device 定义在内核头文件 linux/netdevice.h 中，有一个成员变量

struct Qdisc __rcu *qdisc，即为该虚拟网卡对应的排队规则。这里的__rcu 属性表示该成员变量使用读-复制-更新（Read-Copy-Update, RCU）机制进行同步，需要使用适当的 RCU API 如 rcu_dereference()才能正确获取变量值。此外，可以通过内核 API 函数 dev_get_by_name()直接根据网卡名（如"eth1"）获取对应的 struct net_device 指针

2. struct Qdisc 定义在内核头文件 net/sch_generic.h 中，其有成员变量 struct gnet_stats_queue __percpu *cpu_qstats 和 struct gne_stats_queue qstats，保存着队列的全局统计信息。这里的__percpu 属性表示对于每个 CPU 核心，都有一个独立的 struct gnet_state_queue 实例，用于多核的情况。

3. struct gnet_stats_queue 定义在内核头文件 linux/gen_stats.h 中，其有成员变量__u32 qlen 和__u32 backlog，分别表示队列中实时排队的数据包数量和总字节数。此外，通过内核 API 函数 qdisc_qlen_sum()即可获得一个排队规则中实时的排队数据包数量。

于是，通过分析内核数据结构之间的关系以及调用适当的内核 API，可以在内核模块中根据虚拟网卡名直接获取在该网卡上排队的数据包数量。

下面介绍如何实时监测负载并实现事件触发。为了确保负载监测的实时性，本研究在内核的 ip_output()函数返回时设置一个内核探针。ip_output()函数是 Linux 内核网络协议栈中的函数，负责在单播数据包查询路由表完成后将其发送到对应的出接口。也就是说，对于每一个被成功转发的数据包，其在从网络层经过路由决策后，传输到链路层前必定会调用一次 ip_output()。因此可以在每次调用 ip_output()时监测链路负载。同时，内核探针（kernel probe, kprobe）技术是一种内核调试功能，允许开发者在运行的内核中动态地插入探测点，以跟踪和分析内核函数的执行状态。当内核函数执行到探测点时，可以执行开发者自定义的回调函数。于是，通过在 ip_output()函数返回时设置内核探针检查当前链路负载与上次记录的链路负载之差是否超过阈值δ，即可实现事件触发机制。

(2) 自定义 netlink 通信的实现

内核模块与路由套件的通信机制通过自定义 netlink 实现，netlink 是一种特殊的套接字，提供异步、双工通信，可用于内核空间与用户空间进行通信，且便于开发自定义功能。在实现中，有两部分部分需要用到 netlink 通信，其一是内核模块接受用户空间传来的参数δ，netlink 消息需携带δ的值，其二是内核模块向用户空间上交实时链路负载，netlink 消息需携带负载值。二者实现逻辑类似，此处仅介绍内核模块向用户空间上交实时链路负载的通信实现。

Docker 容器技术利用 Linux 的网络命名空间来创建隔离的网络环境，每个命名空间为容器提供了独立的网络接口和网络栈，确保容器拥有独立的网络设备、IP 地址和端口号，从而实现网络层面的隔离和安全性。因此在模拟平台中，每个 Docker 容器在 Linux 内核中对应着一个网络命名空间，对应内核数据结构为 struct net，定义在内核头文件 net/net_namespace.h 中。为了确保每个 Docker 容器的用户空间与内核空间独立通信而不影响其他容器，netlink 套接字也需要定义在网络命名空间中，即在 struct net 中新增一个成员变量 struct sock *sk 表示 netlink 套接字。需要说明的是，此处不可避免地破坏了内核源代码，但不会影响内核其他功能的执行。

为了确保新建一个容器时，对应的 netlink 套接字也会随之初始化，需要在内核模块中定义一个具有__net_init 宏的初始化函数并实现 netlink 套接字初始化逻辑（主要通过 netlink_kernel_create() 函数），之后使用内核 API 函数 reister_pernet_subsys()注册该初始化函数。其中，__net_init 的作用主要是通过编译器的扩展，将标记为__net_init 的函数放置在特殊的初始化代码段中，这个代码段会在网络命名空间初始化时被执行，由此即可保证在加载此内核模块后新建的所有容器的网络命名空间中 netlink 套接字都被正确初始化。

此外，在初始化 netlink 套接字时需要为其正确指定 netlink 协议号与回调函数。首先在指定协议号时，根据内核头文件 uapi/linux/netlink.h 的规定，netlink 协议号不能超过 32，且 0~16、18~22 号已被使用，因此如果想要实现自定义的 netlink 协议，需要为其指定一个未被使用过的 netlink 协议号，在本部分实现中将其指

定为 30。之后定义的回调函数整体逻辑为，当链路负载变化超过δ时，调用 nlmsg_put()函数构建一条 netlink 消息，其携带着所有队列的实时占用长度，之后调用 netlink_unicast()函数将其发往用户空间，由路由套件进行相应处理。

(3) 路由套件中的实现

路由套件中的实现较为简单，使用 C 语言的第三方库 libnl 接收内核发来的 netlink 消息，读取其中携带的所有队列实时占用长度，并据此更新链路时延以及重计算路由表。

综合考虑上述三部分，基于链路负载的链路状态传播触发的执行流程如图 13 所示。在加载内核模块后，依次进行如下操作：

1. 输入链路负载变化的阈值δ并通过 netlink 消息将其发送至内核；

2. 内核调用 ip_output()函数转发一个网络数据包；

3. ip_output()函数返回时监测链路负载变化，如果当前负载与上次记录的负载之差超过阈值δ，则内核模块生成携带当前负载信息的 netlink 消息，并发送给用户空间的路由套件。否则执行 2；

4. 用户空间路由套件接收 netlink 消息，读取负载信息，并据此更新链路状态数据库；

5. 执行 2。

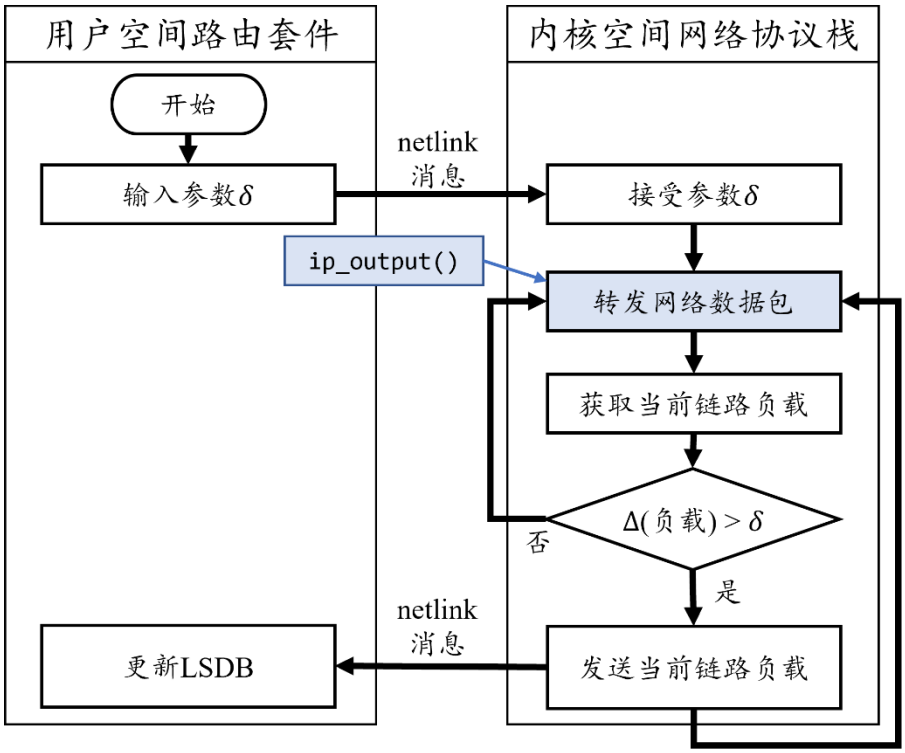


图 13 基于链路负载的链路状态传播触发执行流程

2.4.4 小结

本部分介绍了局部化细粒度星间链路状态传播机制的全部三个子内容在模拟平台中的实现。其中，基于可控传播距离的链路状态通告通过硬编码在路由套件中实现，包括 LSA 的 TTL 字段修改及具体执行流程；基于链路负载的链路状态传播触发通过编写可加载 Linux 内核模块、实现内核模块与路由套件的通信机制以及在路由套件中进行相应处理来实现。

3 下一阶段工作计划

3.1 未完成的工作

目前研究内容一与研究内容二的设计已完成，研究内容三的子内容一已完成，研究内容三的子内容二部分完成：已完成研究内容一对应的实现、未完成研究内

容二对应的实现，研究内容三的子内容三未完成。

3.2 下一阶段计划

预计 2025 年 5 月毕业。2024 年 7 月至 12 月需要在模拟平台中实现低时间复杂度星间路由计算机制（研究内容二），2025 年 1 月至 3 月需要在模拟平台中对研究内容一和研究内容二在模拟平台中开展测试，2025 年 4 月至 5 月需要撰写毕业论文。

3.3 未来工作难点

星间路由计算机制需要在路由套件中通过 C 语言实现，涉及多种复杂数据结构，因此实现较为复杂且优化困难，代码工作量较大。

4 主要参考文献

[1] Moy J. RFC2328: OSPF Version 2[J]. 1998.

[2] Dijkstra E W. A note on two problems in connexion with graphs[M]//Edsger Wybe Dijkstra: His Life, Work, and Legacy. 2022: 287-290.

[3] Pan T, Huang T, Li X, et al. OPSPF: orbit prediction shortest path first routing for resilient LEO satellite networks[C]//ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE, 2019: 1-6.

[4] Taleb T, Mashimo D, Jamalipour A, et al. Explicit load balancing technique for NGEO satellite IP networks with on-board processing capabilities[J]. IEEE/ACM transactions on Networking, 2008, 17(1): 281-293.

[5] Mészáros L, Varga A, Kirsche M. Inet framework[J]. Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem, 2019: 55-106.

[6] Lakshmi K S, Kumar M P S, Kavitha K V N. Inter-satellite laser communication system[C]//2008 International Conference on Computer and Communication Engineering. IEEE, 2008: 978-983.

[7] Chen Q, Yang L, Guo D, et al. LEO Satellite Networks: When Do All Shortest Distance Paths Belong to Minimum Hop Path Set?[J]. IEEE Transactions on Aerospace and Electronic Systems, 2022, 58(4): 3730-3734.

[8] Leyva-Mayorga I, Soret B, Matthiesen B, et al. NGSO constellation design for global connectivity[J]. arXiv preprint arXiv:2203.16597, 2022.