一种基于容器的低轨卫星网络协议测试床

潘 恬 李星辰 薛文浩 边子政 黄 韬 刘韵洁
(北京邮电大学网络与交换技术国家重点实验室 北京 100876)

摘 要 近年来,以 SpaceX 为代表的商业航天公司提出了多个超大规模低轨卫星星座组网计划.卫星路由器是构建卫星互联网的关键基础设施.考虑到卫星路由器高昂的研制和发射成本,在卫星发射之前预先在地面上做好卫星星座的组网仿真和半实物接入验证具有重要意义.相比传统地面网络的仿真验证,卫星星座组网仿真面临高动态拓扑、超大组网规模、真实流量承载仿真以及半实物接入验证等需求和挑战.目前已知的地面网络通用仿真工具(如 QualNet、NS2/NS3、OPNET、Mininet)均无法同时满足上述仿真要求,实现对超大规模动态卫星网络节点的真实协议栈验证.我们针对以上需求,吸收离散事件模拟和虚拟化仿真的思想,以容器为载体,设计并实现了一种低轨卫星网络协议测试床.在控制平面,系统基于卫星星座的数学建模,在内部时钟节拍驱动下生成链路通断的离散事件,精确描述星座拓扑的规律变化.在数据平面,系统使用 Docker 容器实现卫星和地面终端,使用 Linux 虚拟网络设备实现星间和星地链路,并基于隧道协议实现系统的分布式部署和半实物仿真,使其具备优秀的横向扩容能力,从而解决超大组网规模下的单机仿真性能瓶颈.为了充分挖掘多核处理器的计算潜力,系统基于多线程对仿真过程中产生的大量离散仿真事件进行高并发调度.实验结果显示,在多核处理器主机上,该仿真测试床能够同时运行 3276 个网络节点并承载 1.6 Gbps 的真实流量.与单线程实现比较,多线程的任务并发处理机制使得 CPU 利用率提高了 45%,仿真场景创建时间缩短了 56%.

关键词 低轨卫星网络;容器;虚拟网络设备;拓扑变化;多线程;分布式部署;半实物仿真;数字孪生
中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2022.02029

A Docker-Based LEO Satellite Network Testbed

PAN Tian LI Xing-Chen XUE Wen-Hao BIAN Zi-Zheng HUANG Tao LIU Yun-Jie
(*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing* 100876)

Abstract In recent years, commercial space companies represented by SpaceX have proposed numerous mega-scale low-earth-orbit (LEO) satellite constellation networking plans. Satellite routers are the key infrastructure for the satellite Internet. As the manufacturing and launching costs of satellite routers are considerably high, it is necessary to conduct large-scale network emulation as well as hardware-in-loop verification on the ground before launching the satellites into space. Compared with traditional network simulation/emulation solutions, the ground emulation of mega-scale satellite constellation networks has the requirements and challenges of highly dynamic topologies, mega-scale networks, real traffic emulation, and hardware-in-loop verification. At present, existing ground network simulation/emulation tools (such as QualNet, NS2/NS3, OPNET, Mininet) cannot meet the above requirements simultaneously to verify the real protocol stack of mega-scale dynamic satellite networks. To address the above requirements, we design and implement a container-based LEO satellite network testbed by integrating discrete

收稿日期:2020-06-26;在线发布日期:2021-03-09.本课题得到国家重点研发计划(2019YFB1802600)、国家自然科学基金青年科学基金(61702049)、中央高校基本科研业务费资助.**潘 恬**,博士,副教授,主要研究方向为卫星互联网协议、数据中心网络、高速可编程网络设备.E-mail: pan@bupt.edu.cn. **李星辰**,硕士研究生,主要研究方向为低轨卫星网络、网络仿真验证系统. **薛文浩**,硕士研究生,主要研究方向为低轨卫星网络、空间网络安全. **边子政**,硕士研究生,主要研究方向为高速可编程网络设备、网络虚拟化技术. **黄 韬**,博士,教授,主要研究领域为未来网络体系结构、软件定义网络、内容中心网络. **刘韵洁**,教授,中国工程院院士,主要研究领域为未来网络体系结构、网络操作系统、软件定义网络.

event simulation technology and virtualization-based emulation technology. In the control plane, the system is based on mathematical modeling of the satellite constellation and produces discrete events of the satellite link on and off under the drive of an internal clock, which precisely reflects the regular topology changes of the network. In the data plane, the system leverages Docker containers to implement the satellite and ground nodes, and leverages Linux virtual network devices to implement the inter-satellite and satellite-ground links. The system enables distributed deployment and hardware-in-loop emulation based on the tunneling protocol, making the system have good horizontal expansion capabilities to break the performance bottleneck of single-host emulation. To fully exploit the computing capability of the multi-core processors, the system has developed a concurrent task scheduling mechanism based on multithreading for performant scheduling of discrete events during the emulation. The evaluation results show that on a multi-core host, the network testbed can carry 3276 virtual nodes and 1.6 Gbps real traffic. Compared with the single-threaded implementation, the concurrent task scheduler based on multithreading improves CPU utilization by 45% and shortens emulation scenario creation time by 56%.

Keywords LEO satellite network; Docker container; virtual network device; topology change; multithreading; distributed deployment; hardware-in-loop emulation; digital twin

1 引 言

随着卫星通信技术和计算机软硬件技术的发展,星上处理和星上交换能力日益增强,卫星逐步具备了构建大规模空间网络的能力[1].低轨卫星网络具有链路损耗小、时延低、星座覆盖范围广、链路冗余度高、鲁棒性强、便于开展多路径流量负载均衡等优点,已成为当前计算机网络的重要研究方向之一[2].近年来以美国 SpaceX 为代表的商业航天公司提出了多个超大规模低轨卫星星座组网计划.例如,SpaceX 公司的"星链"计划拟发射总共 42000 颗低轨卫星,既为地面民用设施提供随遇接入的网络服务,也构成了对陆海空天的全维目标跟踪覆盖[3].OneWeb 公司预计发射 720 颗低轨卫星,面向个人消费者和社区提供宽带互联网服务[4].亚马逊公司的"Kuiper"计划也规划部署 3236 颗卫星,构建包括地面网关、客户终端、软件定义网络和地面测控中心的卫星网络系统[5].低轨卫星网络拓扑的高动态特性给星座组网的架构和协议设计带来了诸多技术挑战.首先,在拓扑频繁变化的情况下,基于洪泛收敛的地面网络路由协议很难直接部署到低轨卫星网络中,因为每次拓扑变化都将触发全网范围内的链路状态更新通告,消耗大量宝贵的星间链路带宽[6];其次,单颗低轨卫星往往覆盖较大的地表面积,能够同时服务百万量级的地面终端,且地面终端头顶的直连卫星始终处于频繁切换状态,如何设计一个轻量化的终端移动性管理方案成为亟待解决的问题[7-9];再者,低轨卫星网络拓扑通常呈现网状结构,路径冗余度高但单条链路带宽有限,这给流量负载均衡的设计提出了更高的要求;最后,在节点编址、端到端传输控制、高精度网络测量、网络虚拟化、网内计算缓存等方面,低轨卫星网络的技术路线和地面网络也存在较多差异.

以上研究工作都需要依赖真实的低轨卫星网络环境以提供正确的卫星网络链路特性.然而,搭建真实的网络环境通常开销巨大,尤其是低轨卫星网络这样包含大量节点并在太空中运行的网络系统.例如,"铱星"系统虽然仅仅包含 66 颗卫星,但项目前后总共投入了 50 多亿美元[10].近年来,随着低轨卫星的规模化生产和一箭多星等技术的应用,单颗卫星的制造和发射成本得以降低,然而,包含 42000 颗卫星的"星链"计划仍然是一个烧钱的项目,一般的网络研究团队无法负担其超大规模星座的发射和运维成本.因此,一种可行的研究思路是采用虚拟的节点和链路替代目标网络,借助搭建仿真网络实现对目标网络物理特征的还原,并基于仿真网络平台开展组网协议方案的验证和对比.为此,我们亟需一种低轨卫星网络的仿真测试环境,既能还原低轨卫星网络拓扑的高动态性(包括星间链路的规律和非规律通断,卫星覆盖地表区域的实时变化等),又能承载网络中的真实业务流量,以达到验证低轨卫星组

网架构和网络协议的目的.

当前主流的仿真网络构建方法包括两大类：基于离散事件的模拟(Simulation)和基于虚拟化的仿真(Emulation)[11]. 其中，基于离散事件的模拟技术借助数学模型的定义对网络特性和协议行为进行描述. 卫星网络的模拟器则通过数学模型对星座的时序、运动、拓扑和协议行为等进行刻画. 离散事件模拟器相对容易部署，系统通常以软件的形式安装在单台主机上. 然而，这也使得离散事件模拟器所能承载的网络节点规模受到单台宿主机计算性能的限制，在给定的仿真时间约束下，仅适合完成小型网络的功能验证. 此外，考虑到离散事件模拟器中的数据流均是模型计算产生而非真实业务流量，因此其保真度受到数学模型精度的影响[12]. 相较而言，基于虚拟化的网络仿真器具有较高的保真度，因为仿真环境里的每个虚拟网络节点都具备真实的网络协议栈，均能视作真实网络里的设备，所处理的数据流也均为业务流量报文. 其不足之处首先是不支持对网络特性和协议行为的数学建模，在仿真器中加载新协议通常会涉及到对协议栈的二次开发，周期较长；其次是仿真系统搭建过程较为复杂，且随着仿真规模的扩大将会线性消耗大量的计算资源. 考虑到低轨卫星星座拓扑的动态性，面向低轨卫星网络的虚拟化仿真系统既需要对卫星星座运行规律进行数学化定义，又需要能够承载真实的业务流量，还应当支撑星座规模增大时仿真容量的按需扩容. 单独采用离散事件模拟或虚拟化仿真均不能同时满足上述需求. 本文的思路是将这两种技术结合，从而实现两者的优势互补.

我们针对以上需求，基于离散事件模拟与虚拟化仿真的思想，以容器为载体，设计了一种低轨卫星网络的协议仿真测试床. 该系统基于 Linux 操作系统研发，使用 Docker 容器[13]对单颗卫星或单个地面终端进行抽象，从而能够在 Docker 内部开发定制化的网络协议栈，并对真实数据包流量进行处理和转发. Docker 之间的虚拟网络链路也能够基于 STK[14]导出的动态星间拓扑和星地拓扑进行通断配置. 具体的，在控制平面，系统基于星间和星地拓扑的实时变化规律，在系统内部时钟的驱动下持续输出星间链路和星地链路的通断事件，并下发到数据平面. 在数据平面，系统使用 Docker 容器承载卫星节点和地面终端节点；使用 Linux 虚拟网络设备[15]实现星间链路和星地链路，并在通断事件的触发下精确模拟网络拓扑变化；系统还基于隧道协议设计了分布式部署和半实物仿真的方案，使得系统具备优秀的横向性能扩展能力. 该系统既可以运行在单台物理主机上，也可以分布式部署在多台物理主机上实现仿真容量的横向扩展，还可以引入外部流量进行半实物仿真. 为了提升单主机的仿真容量，系统基于多线程实现了对离散仿真事件的高并发处理. 测试结果显示，仿真系统在多核处理器主机上能够承载大量网络节点的真实流量处理任务. 基于多线程的高效任务调度机制通过提升 CPU 的利用率大大缩短了卫星网络的仿真耗时. 除了卫星网络的仿真验证，本文基于容器和虚拟网络设备的仿真思路也可用于其他大规模物联网系统的快速组网验证.

本文的主要贡献总结如下：

(1) 设计并实现了一种基于容器的低轨卫星网络协议测试床，能在容器内部灵活定制协议栈，并可以通过控制容器间的虚拟链路通断逼真的模拟低轨卫星星座的拓扑变化. 在获得外部输入的卫星星座拓扑参数后，该设计思路也可用于高轨卫星或高低轨混合卫星星座的网络协议栈仿真.

(2) 提出基于 VXLAN 隧道协议实现仿真系统分布式扩展及半实物仿真的思路，进而通过分布式部署极大提升系统的仿真总容量，并通过半实物仿真支持真实卫星路由器设备的串入仿真验证.

(3) 充分挖掘多核处理器的并发处理潜力，基于多线程机制极大提升单个物理主机上的仿真性能. 针对大量并发任务间存在复杂依赖关系的场景，提出“任务树”的抽象机制，在更高的层次上对多线程并发处理的底层调度原语进行封装，降低程序员维护大量并发任务间依赖关系的难度.

(4) 本文提出的仿真方案将真实世界中的低轨卫星星座映射到运行在物理机集群上的虚拟卫星网络节点上，通过对虚拟网络协议栈的实验验证反馈有效信息给真实世界的低轨卫星网络组网设计，并能够基于半实物仿真实现真实网络和虚拟网络的交互融合. 该方案在一定程度上初步具备了当下热点概念“数字孪生”[16]的雏形.

2　背景及相关工作

2.1　低轨卫星网络

低轨卫星星座由规律分布的 LEO 卫星构成，通常使用基于倾斜轨道的 Walker Delta 星座[17]组网方案，主要特征参数包括轨道倾角、轨道数量、单轨道卫星数、卫星总数、轨道高度、相位因子、极区阈值

以及运行周期等.其中,极区阈值定义了相邻轨道相邻卫星星间链路连通和断开的边界条件(一个典型值为 70 度).当卫星处在极区之外时,相邻轨道星间链路保持连通;当卫星进入极区时,卫星天线跟踪速度和摆动角度的限制使得相邻轨道相邻卫星间无法保持正常通信.一组完整的卫星星座参数可以精确刻画卫星星座的物理特征,也便于软件通过数学模型预测给定星座中每颗卫星的实时运动状态.

低轨卫星网络包含卫星节点和地面终端节点.典型的地面终端有地面移动用户和地面信关站.星间链路(Inter-Satellite Link,ISL)存在于相邻卫星之间,星地链路(Satellite-Ground Link,SGL)存在于地面终端与头顶卫星之间.卫星可分为弯管式(Bend-Pipe)卫星和星上处理(On-Board Processing)卫星,前者只能为地面节点中继信号,后者具备星上路由和交换能力.考虑到越来越多的低轨卫星具备星上路由和交换能力,本文研究对象为星上处理卫星.低轨卫星通常携带 4 根星间天线和 1 根对地天线.地面终端一般具有 1 根用户天线.完整的低轨卫星网络包括星上网络和地面接入网络.其中,星上网络由卫星星座中的星间链路组成,地面接入网络由地面终端和头顶卫星连接的星地链路组成.例如,在铱星星座中,每颗卫星拥有 4 条 ISL,其中 2 条 ISL 实现同轨道前后相邻两颗卫星的连接,2 条 ISL 实现侧向轨道左右相邻两颗卫星的连接.大部分相邻轨道上的卫星运动方向一致,仅存在两条相邻轨道上的卫星运动方向相反(这两条轨道间形成反向缝).由于运动方向相反,相对速度过大,天线互相跟踪困难,因此反向缝两侧卫星无法建立有效的星间链路.

2.2 低轨卫星网络的建模仿真

2.2.1 卫星组网建模工具

卫星工具包(Satellite Tool Kit,STK)[14]可以针对低轨卫星网络开展星座参数分析、数学模型构建和卫星场景可视化等工作.STK 能够基于卫星星座参数对给定星座随时间变化的卫星位置、星座拓扑以及卫星覆盖区域进行数学建模,精确预测出任意时刻的卫星星座运动状态.然而,STK 本身并不携带 TCP/IP 协议栈,因此无法对网络协议和报文处理进行仿真分析.当在 STK 上完成星座分析建模后,我们能够获得不同时刻的实时卫星位置和星座拓扑文件.这些信息可以输入到专门的协议仿真软件中,提供卫星网络仿真所需星座运动规律的真实数据支撑,实现联动仿真.

2.2.2 卫星网络仿真工具

为了逼真复现网络设备对报文的协议处理过程,有两类仿真方法可以通过在主机上构建虚拟的网络环境还原真实的目标网络.一类是基于离散事件的网络模拟,它能够利用数学建模和统计分析来模拟网络行为,也被称为 Simulation.例如,NS2/NS3[18]能够通过数学模型,支持多层网络协议栈的模拟,被广泛用于网络协议的学术研究,但受限于性能,无法在有限时间内完成超大规模网络场景的高效仿真.一部分卫星网络仿真工作基于 NS2/NS3 构建[19],继承了 Simulation 技术的优缺点.QualNet[20]和 OPNET[21]是商用的离散事件模拟器,被广泛用于地面通信网络系统的仿真测试和比对验证.相比 NS2/NS3,QualNet 和 OPNET 支持更完整的协议簇,仿真速度更快,仿真保真度更高且单机仿真容量更大.但商业软件的闭源性质使得其使用成本昂贵,缺乏自主可控能力,且功能和性能较难进一步扩展.例如,目前 QualNet 并不支持多机分布式仿真,因此受限于单机性能无法模拟马斯克"星链"计划中的超大规模卫星网络.另外,在 QualNet 的仿真框架上通过二次开发支持全新的组网架构(如 SDN)也需要较大的工作量.

另一类方法基于虚拟化技术在物理主机上创建资源隔离的虚拟网络节点,从而构建能够承载真实流量的仿真网络环境,也被称为 Emulation.经典的网络仿真器有 Mininet[22],它基于 Linux 的网络命名空间构建,使用 OVS[23]交换机对物理交换机进行仿真,具备真实协议栈处理能力,在软件定义网络的功能验证中被广泛应用,但不适合仿真运行分布式协议(如 OSPF)的网络,也无法准确地复现低轨卫星网络中拓扑动态变化的真实场景.

2.3 虚拟化技术

2.3.1 容 器

容器实现了轻量级的虚拟化,能够为程序搭建一个资源隔离的运行环境.尽管容器和虚拟机都提供了资源隔离的能力,但虚拟机本质上是对底层硬件的虚拟化,而容器是对操作系统的虚拟化.也就是说,每个容器并不需要包含单独的操作系统,多个容器可以像进程一样跑在相同的操作系统之上,因此容器更加轻巧,资源利用率更高,更加方便在云端创建和迁移.开源容器引擎 Docker[13]利用 Linux 所提供的控制组(Control Groups,Cgroups)和命名空间(Namespaces)机制轻量化的实现了程序运行时的资源隔离.具体的,Cgroups 能够基于 Linux 内核的

进程资源管理机制精细化地管控每个进程所使用的CPU、内存等物理资源. Namespaces 是 Linux 内核中实现内核资源隔离的技术，包括了多个内核资源的命名空间. 其中，网络命名空间实现了网络资源的隔离，每个容器也因此拥有了独立的 IP 地址、路由表、协议栈等虚拟网络资源.

2.3.2　虚拟网络设备

在网络虚拟化技术出现之前，计算机系统一般通过物理网卡和线缆介质进行互连. 随着网络虚拟化技术的产生，一台物理主机之上能够运行大量的虚拟机或容器实例，也就需要“虚拟互连介质”(即虚拟网络设备[15])负责将这些虚拟网络节点连接起来，构建大规模的虚拟网络拓扑. Linux 操作系统对虚拟网络设备有着非常全面的支持，典型的虚拟网络设备包括 Veth-pair、Bridge 和 Vxlan. 其中，Veth-pair 包含成对出现的虚拟以太网接口(Virtual Ethernet Interface，VETH)，一端的 VETH 可以向另一端的 VETH 发送流量，并被另一端 VETH 连接的内核协议栈处理. 因此，Veth-pair 通常用来连接两个不同的网络命名空间(例如 Docker)或虚拟网络设备(例如 Bridge). Bridge(即网桥)是一种虚拟以太网交换机，类似物理交换机，一端连着协议栈，另一端有多个端口，流量基于 MAC 地址在各端口间转发. Vxlan 是一类特殊的 VETH，通过 Vxlan 可以快速构建基于虚拟可扩展局域网协议(Virtual eXtensible Local Area Network，VXLAN)[24] 的 Overlay 网络. 创建 Vxlan 时需要绑定本地和远端的 IP 地址并指定 VXLAN 的网络标识符(VXLAN Network Identifier，VNI)以建立传输隧道.

3　低轨卫星网络仿真的挑战

3.1　功能方面的挑战

3.1.1　星座拓扑变化的模拟

低轨卫星网络拓扑(包括星间拓扑和星地拓扑)始终处于变化状态，仿真系统需要准确还原这种变化，具体包括拓扑变化事件的精准生成、网络节点与链路的抽象以及链路物理通断的模拟.

拓扑变化事件包括星座网络的拓扑变化和地面接入网络的拓扑变化. 星座网络的拓扑变化由卫星位置随时间变化导致的星间链路通断触发. 地面接入网络的拓扑变化由卫星和地面终端相对位置变化导致地面终端头顶直连卫星发生切换触发. 仿真系统需要模拟卫星以及地面终端的位置更新，并进一步推导得出星间链路以及星地链路的实时连通关系，最终得到星上网络和地面接入网络的实时拓扑，从而实现对网络拓扑变化事件的精准生成.

卫星和地面终端具有不同的天线模型，卫星有多根天线而地面终端只有一根天线. 星间链路和星地链路具有不同的信号传播模型，星间链路独占两侧天线(因为一颗低轨卫星的某个星间天线同时只会和一个邻居卫星建立连接)，而星地链路对卫星对地天线是非独占的，对地面终端的用户天线是独占的(因为一根卫星对地天线能够通过信号广播同时和多根地面终端天线建立连接，而每根地面终端的用户天线在稳定通信时只连接一颗头顶卫星). 仿真系统构建的卫星节点、星间链路、地面终端节点和星地链路均需要符合上述客观行为特征.

星间链路和星地链路的通断实现方式不同. 由于星间链路天线的独占性，其通断可在任意一端或两端的天线处进行配置. 与之不同的是，星地链路因共享卫星对地天线，其通断只能在地面终端侧的天线处进行配置. 仿真系统需要能够在拓扑变化事件的驱动下，对相关仿真节点及链路进行针对性的配置，以实现对不同类型网络链路通断的正确仿真.

3.1.2　真实协议栈与真实流量的仿真

在基于 IP 协议的低轨卫星网络中，地面终端负责生成业务流量，应具备完整的 TCP/IP 协议栈. 低轨卫星需要完成报文的星上处理，包括报文收发、报文头协议解析、星间路由计算、星上报文交换、移动性管理等，其协议栈至少应包含物理层、数据链路层和网络层. 也就是说，系统中的仿真节点应当具备完整的网络协议栈功能，且卫星节点还应具备星上路由和交换的能力.

通过建模和统计来模拟网络行为的手段在模型的精确性和仿真的可扩展性上均存在局限性. 首先，数学和统计模型不足以逼真刻画真实世界中所有的复杂流量模型和协议处理场景；其次，当仿真系统支持新协议和新功能时，需要进行相应协议处理的建模工作，这种仿真场景构建通常耗时较长、可扩展性欠佳. 另外，工程领域常常希望在真实卫星路由器设计制造完毕后使用真实设备替代仿真网络中的一部分仿真节点，通过半实物仿真在逼真的环境中使用真实业务流量对真实设备及其网络协议栈进行功能验证和性能测试. 这就要求在仿真系统内部支持真实业务流量的生成和协议栈处理，同时允许从外部接入真实设备和真实业务流量以实现半实物仿真.

3.2　性能方面的挑战

3.2.1　超大规模组网节点

低轨卫星网络通常包含大规模的组网节点. 一

方面，星上网络节点数目庞大，例如 SpaceX 公司提出的“星链”计划在整个项目周期内拟发射 42 000 颗低轨卫星. 另一方面，地面接入网络所涉及的地面终端节点数目巨大，因为每颗低轨卫星往往覆盖较大的地表面积，同时服务海量的地面终端. 因此，仿真系统应当具备海量组网节点的承载能力. 要做到这一点，需要两个维度的优化：首先，仿真系统部署在单台主机上时能够支持较高的节点仿真容量；同时，系统也能够通过横向扩展(Horizontal Scaling)等手段支持超大规模星座在多台主机上的分布式部署使得总体仿真性得到进一步提升.

仿真系统在单台主机上部署时，仿真规模受到单台主机计算和存储性能的限制. 如果仿真规模是 1000 个组网节点而单机的仿真节点容量上限仅为 100，那么我们需要通过以 10 倍性能为目标提升单机的硬件配置(例如更换更强大的多核 CPU 以及更多的内存)来有效提升单机的仿真性能. 然而这种垂直扩容(Vertical Scaling)的方式代价高昂并且不够灵活. 通常来讲，性能更高的硬件模块因为专用度更高、产量更少，较难摊薄研发制造成本，导致价格居高不下. 另一方面，这种垂直扩容的方式总会遇到性能瓶颈，因为单机性能无法无止境的提升. 因此，系统应更多的考虑使用横向扩容的手段，通过多机分布式部署的方式以较高的性价比来实现卫星星座仿真节点处理规模的扩张.

3.2.2 仿真事件的高效调度

系统在仿真时将生成大量的离散仿真事件(例如节点创建和释放、链路通断、节点位置变更、节点路由更新等). 这些仿真事件将会触发对网络节点与链路的多个配置动作，这些配置动作通常是比较耗时的 I/O 密集型操作，例如 Docker 容器和 Veth-pair 的创建过程均为秒级. 此外，不同仿真事件可能是并发产生的(例如不同虚拟网络节点的创建)，也有一定概率存在前后依赖的串行执行关系(例如链路断开事件应在链路创建完成之后). 仿真系统应在满足串行依赖关系的前提下尽可能进行高并发的事件调度，从而充分挖掘多核 CPU 的并行计算潜力，缩短总体仿真时间.

4 仿真系统设计与实现

4.1 基于控制平面与数据平面分离的仿真架构

仿真系统在架构上包含分离的控制平面和数据平面，控制平面和数据平面分别基于上述的离散事件模拟技术和虚拟化仿真技术实现. 具体的，我们在控制平面中根据星座的真实运动规律，使用数学模型对网络拓扑变化进行刻画，在内部时钟或用户外部输入的触发下生成链路通断事件；我们在数据平面上基于 Docker 容器承载网络节点，基于 Linux 虚拟网络设备实现网络链路. 控制平面产生链路通断事件触发数据平面对仿真节点和链路进行配置，实现星座拓扑变化的实时重现. 图 1 展示了系统的控制平面和数据平面以及两者的交互.

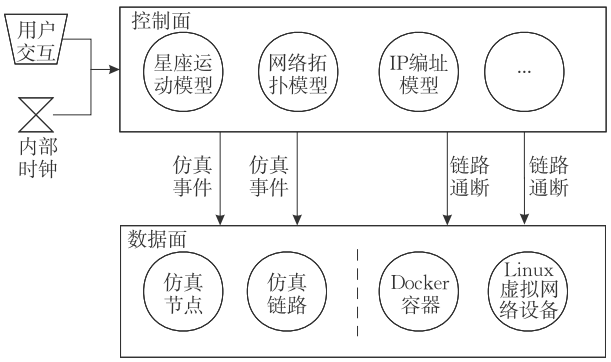

图 1 仿真系统的控制平面与数据平面

系统具备对协议栈与网络流量的仿真能力：容器的选用使得仿真节点具备了完整且互相独立的 TCP/IP 协议栈，虚拟网络设备的选用则为仿真节点提供了收发报文的能力. 在容器中运行路由守护进程可以使仿真节点具备路由学习能力，从而模拟卫星路由器节点. 在容器中运行收发包进程可以产生和接收内部业务流量，从而模拟地面终端节点. 在容器中运行 VXLAN 协议构建隧道则使得接入外部业务流量实现分布式部署或半实物仿真成为可能.

仿真系统通过 Docker 容器环境提供了完整的 Linux 网络协议栈，因此通过修改协议栈代码理论上可以支持各层协议的定制化开发. 值得说明的是，因为现阶段的仿真系统基于有线链路的可控通断来仿真无线星间/星地链路的建立和拆除，所以对物理层的误码率和数据链路层的专用协议的支持仍显不足. 此外，如果想完全在用户态进行协议开发，也可以借助 DPDK 这样的用户态协议处理软件，直接从虚拟网卡处抓包进行转发处理. 如果需要进行 SDN 网络的搭建，则可以另外引入控制器，并将控制器集中计算得到的路由信息通过 Linux route 命令下发到各个 Docker 容器的路由表中.

Docker 容器非常的轻量化，因此仿真系统在单机部署的情况下就能够支持较大规模的仿真节点容量. 即便如此，当仿真超大规模卫星网络(如“星链”

计划的上万颗卫星)时，如果仿真节点数量超出了单机物理性能的极限，则可将仿真节点分散部署到多台物理主机上，基于隧道技术进行互连互通，使得系统具备大规模组网的仿真能力，且随着组网节点的持续加入能够做到性能的横向扩展.此时，仿真系统的数据平面采用分布式部署的方式.值得说明的是，在现阶段的仿真系统中，无论数据平面基于单机还是分布式部署，控制平面均采用单机部署.这样一方面可以简化设计实现，另一方面可以减少维护数据一致性的同步开销.另外，在控制平面上，我们基于多线程技术对仿真事件进行并发调度以最大程度提升仿真计算效率.

系统对目标低轨卫星网络的仿真分为三个步骤：创建仿真网络、网络动态仿真和释放仿真网络.具体而言，系统首先根据星座参数创建仿真网络，包括创建控制平面中的数学模型以及数据平面中的仿真节点和仿真链路.然后，系统开启内部时钟进行动态仿真.控制平面在内部时钟的激励下产生链路通断事件，触发数据平面对仿真节点和仿真链路实施动态配置，进而实现星座规律性拓扑变化的重现.在仿真过程中，也可以通过外部用户交互产生异步链路通断事件，实现链路突发故障的模拟.动态仿真结束后，系统需要及时释放仿真网络占用的控制平面和数据平面资源，避免发生内存泄漏.

在仿真过程中，仿真网络拓扑的高动态变化为所承载空间网络协议的验证提供了逼真的低轨卫星网络环境.路由协议、传输控制协议以及应用层的业务流量模型均能够在数据平面得以部署.例如卫星节点中的路由守护进程能够感知到链路状态的变化并根据协议规范进行路由收敛；地面终端节点的端到端通信时延也能够反映相关协议簇在拓扑高动态变化的卫星网络环境下的真实性能表现.

4.2　低轨卫星网络拓扑变化的计算方法

为了简单起见，本文以极地轨道举例，描述低轨卫星网络拓扑变化的计算方法.其他更加复杂的卫星星座(如倾斜轨道星座)的运动规律可参考相应的文献[17].需要说明的是，我们的仿真系统并不局限于某个特定的卫星星座.只要该星座的运动规律可以用数学精确描述或者能够提供链路通断的时序文件，仿真系统均可以通过计算链路状态或读入通断时序文件进行仿真.

低轨卫星网络包括地面接入网络和星上网络.其中，地面接入网络的拓扑变化规律相对简单.假设地面终端连接头顶仰角最大的卫星，且同时只可以和一颗卫星建立星地链路，那么星地链路数等于地面终端数.在此前提下，只需找到当前时刻地面终端的头顶直连卫星并对比前一时刻的直连卫星，即可决定是否触发星地链路的通断事件.其中，卫星仰角可基于卫星与地面终端的经纬度和卫星高度计算.下面重点讨论星上网络拓扑变化的计算.

判断星上网络是否发生拓扑变化可分为以下步骤：首先获取当前时刻卫星的经纬度信息，然后基于卫星间相对位置关系计算对应星间链路的通断状态，最后对比前一时刻的星间链路状态判断是否触发星间链路的通断事件.

其中，卫星的位置变化应满足星座运动规律，可基于 STK 对卫星星座进行数学建模，实现任意时刻星座中任意卫星位置的获取.基于卫星前后时刻的位置变化也可以判断该卫星当前的运动方向，如果纬度增大则卫星向北运动，反之则向南运动.

在低轨卫星星座中，星间链路只能在相邻两颗卫星之间建立.为了求解所有星间链路的通断状态，首先分析低轨卫星间相邻的两种情况：一种是两颗卫星属于同轨道前后相邻的关系，另一种是两颗卫星属于邻轨道左右相邻的关系.在典型的低轨卫星星座(如 LEO-48 星座)中，根据星座运动规律，一颗低轨卫星的周围 6 个邻居(同轨道前后 2 颗，左右邻轨道左前向、左后向、右前向、右后向共 4 颗)构成一个固定的卫星集合，即无论该卫星如何运动，与之相邻的 6 颗卫星都不会改变.当然，考虑到反向缝的特殊性，处于反向缝两侧的卫星之间不会形成稳定的相邻关系，因此在反向缝两侧轨道上运动的卫星只存在 4 个而非 6 个邻居卫星.

星间链路以两侧的卫星作为唯一标识，遍历每颗卫星并将其和相邻卫星之间的全部星间链路插入到集合中，就能够得到星间链路的完整集合.然后，通过判断集合中每一条星间链路的通断情况即可得出当前时刻的星上网络拓扑.考虑到以下三方面原因，星间链路并非始终处于连通状态：

(1)天线摆动角度的限制

侧向邻居卫星相对当前卫星的位置随着卫星的运动会产生变化.为了保持邻轨道左右相邻卫星间的正常通信，两颗卫星的天线会不断微调角度保持互相指向关系.然而，当卫星运动到极区内时，由于天线摆动角度超过最大限制，天线将无法继续保持互相跟踪，此时相邻轨道的侧向链路将处在断开状态.作为对比，同轨道邻居卫星的相对位置始终保持不变，因此同轨道星间链路始终保持连通.

(2)卫星天线数量的限制

一颗低轨卫星需要 6 根星间天线才能够同时和 6 颗相邻卫星构建星间链路. 为了兼顾制造成本和组网需求,卫星通常只会携带 4 根星间天线,两根为同轨前后向天线,两根为邻轨侧向天线(例如左前向和右后向天线). 因此,在任意时刻,集合中的侧向星间链路至少会有一半处于断开状态.

(3)左右邻居跨极区反转

如图 2,跨极区前,卫星向北运动,G 的左前向天线指向 C,右后向天线指向 F,假设此时星间链路(G,C)和(G,F)连通,(G,D)和(G,E)断开. 跨极区后,卫星向南运动,G 的左前向天线和右后向天线分别指向 E 和 D,星间链路(G,C)和(G,F)由连通变为断开,(G,D)和(G,E)由断开变为连通. 由此可见,侧向星间链路会随着左右邻居的反转而交替连通,其连通状态在卫星特定运动方向下保持不变(例如卫星 G 跨极区前后,连通的链路均为左前向和右后向邻轨星间链路).

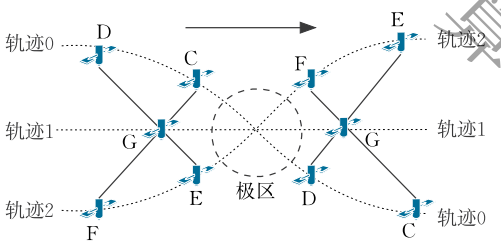

图 2 卫星 G 左右相邻卫星跨极区后的反转过程

综上,可以归纳得到判断 4 天线模型(假设左前向/右后向连通)下的星间链路通断状态的一般计算方法:首先判断星间链路类型,如果链路是同轨道前后向链路,那么直接返回链路连通;然后判断侧向星间链路两侧卫星是否处于极区内,如果至少存在一颗卫星处于极区,那么直接返回链路断开;最后判断侧向星间链路在当前卫星运动方向下是否属于始终保持连通的左前向/右后向星间链路,如果属于返回链路连通,否则返回链路断开.

4.3 基于虚拟化技术的仿真网络构建

相较虚拟机,Docker 容器更加轻量化,因此,我们可以在单台物理机上启动大量的容器节点来仿真较大规模的网络. 同时,容器具备独立的资源空间和完整的内核协议栈,且具有完备的管理接口,因此每个容器节点都能够根据用户配置装载原生的或定制化的协议栈以处理真实的网络流量. Linux 虚拟网络设备(包括 Veth-pair、Macvlan、Bridge、Vxlan 等)可以互相连接,也可以归属于某个容器,或者连接到内核协议栈上,实现网络链路的构造和流量报文的收发. 总之,使用容器与虚拟网络设备相结合的方法能够对网络节点和链路进行完整的模拟,从而轻量化的搭建复杂的虚拟仿真网络. 图 3 展示了仿真网络的单机部署模型. 我们将在下文详述仿真网络中卫星节点和星间、星地链路的构建技术细节.

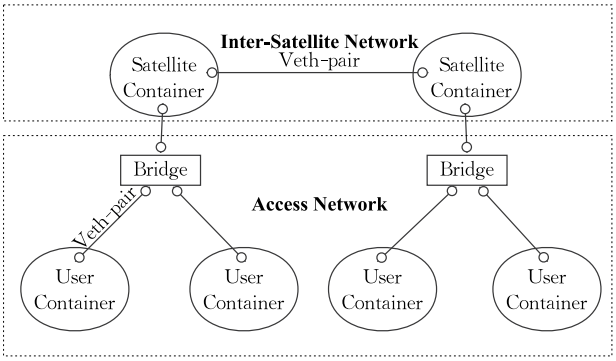

图 3 仿真网络的单机部署模型

4.3.1 节点仿真模型

在低轨卫星网络中,卫星负责报文的星上处理;地面终端负责网络业务流量的产生和消费,以及与接入卫星的交互. 容器很好的提供了对星上报文处理仿真和地面终端业务流量仿真的支持. 此外,不论是卫星还是地面终端都会受到低轨卫星网络实时拓扑变化的影响,产生星间链路和星地链路的通断事件. 通过精确配置虚拟网络设备的通断,我们可以逼真模拟低轨卫星网络的拓扑变化过程.

低轨卫星节点有 4 根星间天线和 1 根对地天线,终端节点有 1 根用户天线. 如前所述,星间天线与用户天线是非复用的,任意时刻只能参与 1 条链路的构建. 考虑到低轨卫星具有较大的对地覆盖面积,单根卫星对地天线可以同时参与到多条星地链路的构建,因此是可复用的. 非复用天线能够通过 Veth-pair 构造:将 Veth-pair 一侧的 VETH 与容器绑定(即归属到容器),成为容器的内部接口(Container's Inner Interface,CII),另一侧的 VETH 以容器外部接口(Container's Outer Interface,COI)的形式暴露到外部. 可复用天线的"广播"功能可以基于 Bridge 进行抽象. 把非复用天线的 COI 绑定到 1 个 Bridge 即可实现单颗卫星同时与多个地面终端连接的仿真效果. 综上所述,卫星仿真节点包含 1 个 Docker、5 条 Veth-pair 与 1 个 Bridge;终端仿真节点包含 1 个 Docker 和 1 条 Veth-pair. 我们也可以把在卫星节点中用于表示星间链路的 4 条 Veth-pair 从卫星节点中抽取出来,专门由星间链路模块管理.

在该模型下，卫星节点可简化为只由 1 个 Docker、1 条 Veth-pair 和 1 个 Bridge 构成. 图 4 展示了卫星和地面终端的节点仿真模型.

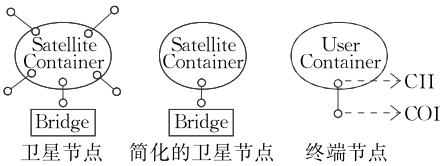

图 4　卫星节点和地面终端节点的仿真模型

4.3.2　链路仿真模型

(1) 星间链路

我们把两根来自不同卫星节点的星间天线(即两个非复用天线的 COI)通过 Bridge 相连即可实现星间链路的模拟. 考虑到该 Bridge 的作用仅仅是 COI 互连，上述实现可进一步简化：我们可以把一条 Veth-pair 两端的 VETH 分别归属到两侧的卫星容器. 在该简化模型中，卫星的星间链路与星间天线融合成一个整体，卫星节点得以简化，同时也省去了额外 Bridge 的引入. 当链路两侧卫星节点部署在不同物理主机时，只需将 Veth-pair 替换为一对互相匹配的 Vxlan. 图 5 展示了星间链路的仿真模型.

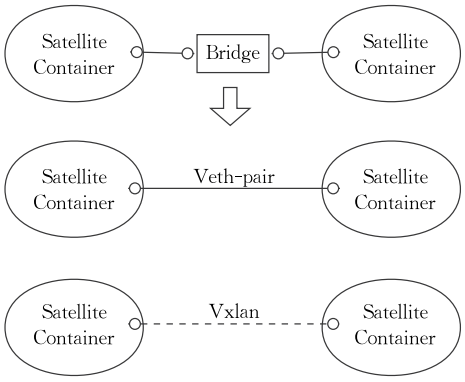

图 5　星间链路的仿真模型

(2) 星地链路

考虑到卫星节点已经包含了具有广播功能的卫星对地天线且终端节点已经包含了用户天线，我们把用户天线的 COI 连接到卫星节点的对地广播 Bridge 上就能够模拟出一条具有广播属性的星地链路. 当卫星节点与终端节点部署在不同的物理主机上时，需要创建一个 Bridge 和一对 Vxlan 进行分布式扩展. 在终端节点所在主机中，将用户天线的 COI 以及 Vxlan 接口绑定到新创建的 Bridge 上. 在卫星节点所在的主机中，将 Vxlan 接口绑定到卫星节点的 Bridge 上. 图 6 展示了卫星和终端分别在单台主机上和在不同主机上的星地链路仿真模型.

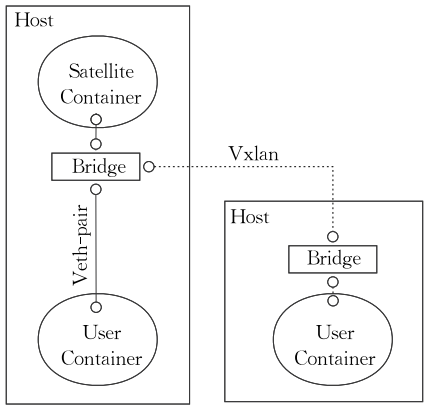

图 6　星地链路的同主机和跨主机仿真模型

4.3.3　链路通断仿真

链路的通断有很多种实现方式，例如，对于链路断开，我们可以直接删除或关闭相关的虚拟网络设备，也可以解除 VETH 与 Bridge 之间的绑定关系. 不同的实现方式均包含成对出现的连通和断开命令，需要按照一一匹配的方式执行.

(1) 星间链路通断

星间链路对其中的 Veth-pair 是独占的，链路的通断可采用 Veth-pair 的创建和删除来实现，也可采用两端 VETH 的打开或关闭来实现，后者具有更高的仿真效率(见 5.4 节). 由于在仿真过程中星间链路两侧的卫星节点不会改变，且仿真节点不会发生主机间的迁移(见 4.4 节)，星间链路不会发生同主机模型与跨主机模型间的切换.

(2) 星地链路通断

考虑到卫星对地天线是可复用的，单条星地链路的通断不应改变卫星对地天线与其他终端的连接状态. 我们可以通过将地面终端 COI(或 Vxlan)绑定到卫星 Bridge 或从卫星 Bridge 上解绑来实现星地链路的通断. 另外，星地链路与卫星的映射关系不固定，当进行分布式仿真或半实物仿真时可能会发生同主机模型与跨主机模型间的切换. 如图 7 所示，主机 A 上的终端节点 h0 发生星地移动切换时需要在位于不同主机的头顶卫星节点 s0、s1、s2、s3 之间进行跨主机的星地链路切换，步骤如下：

① 直连卫星由 s0 切换为 s1：把 h0-coi 从 sbr0 上断开，在主机 A 上新建一个 Bridge(hbr0)和一个 Vxlan 设备(h0-A)，并将 h0-A 绑定到 hbr0 上；与之对应的，在主机 B 上新建一个 Vxlan 设备(h0-B)，并绑定到 sbr1 上；将 h0-A 和 h0-B 配对相连；将 h0-coi 重新绑定到 hbr0 上.

② 直连卫星由 s1 切换为 s2：在主机 B 内部将

h0-B 重新绑定到 sbr2 上即可.

③ 直连卫星由 s2 切换为 s3:删除 h0-A 和 h0-B 这两个设备,在主机 A 和主机 C 上重新创建互相配对的 h0-A′和 h0-C 设备,并把 h0-A′和 h0-C 分别绑定到 hbr0 与 sbr3 上,其余操作同步骤①.

④ 直连卫星由 s3 切换为 s0:删除 hbr0 和 Vxlan 对,然后把 h0-coi 重新与 sbr0 绑定.

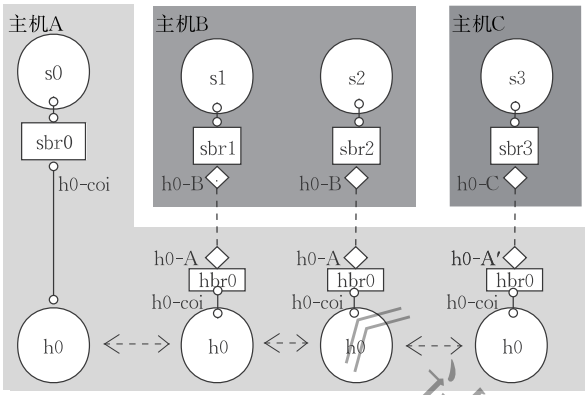

图 7 星地链路同主机与跨主机模型间的切换

4.4 分布式部署与半实物仿真

4.4.1 虚拟化资源的管理

Docker 容器与 Linux 虚拟网络设备都能够通过 Shell 脚本进行配置,这种配置方法灵活简单且具有良好的可扩展性.我们的仿真系统同样基于 Shell 管理虚拟化资源.由控制平面产生的仿真事件本质上是配置数据平面虚拟化资源的一系列 Shell 操作.在单机部署下,系统的控制平面和数据平面处在同一台物理主机中,控制平面使用本地 Shell 脚本就能够配置数据平面上的虚拟化资源.针对多机部署的场景,数据平面分布在多个物理主机中,控制平面部署在另一台物理主机中,控制平面可以基于 SSH 向数据平面所在主机发起跨主机的远程 Shell 调用.此时,我们把控制平面所在的主机称为控制主机,数据平面所在的主机称为载体主机.

4.4.2 仿真节点与物理主机的映射(分布式部署)

在分布式部署的场景下,仿真节点将分散在多台物理主机上.此处存在 M 个仿真节点负载在 N 台物理主机上的分配策略优化问题.假设每台物理主机包含相同的物理资源,那么分配策略存在两个优化目标:首先,仿真节点应尽可能均匀的分布在多台物理主机上,使得物理主机间负载均衡;然后,应尽可能减少跨物理主机的网络资源消耗,即尽可能降低跨主机网络链路的使用.针对目标一,仿真系统保证每台物理主机上分配的负载个数为 $\lfloor M/N \rfloor$ 或 $\lfloor M/N \rfloor+1$.针对目标二,考虑到位于相同轨道并且前后相邻的两颗卫星之间必然存在星间链路,仿真系统应尽可能将这些同轨道相邻卫星分配到相同的物理主机上,使得这些相邻卫星间的星间链路成为主机内链路而非跨主机链路.当然,除了上述基于经验的做法,也可以严格采用图分割算法求得割边最小化的卫星星座分割方案.

4.4.3 实物节点与仿真节点的映射(半实物仿真)

仿真系统支持半实物仿真,这使得即将发射上天的卫星路由器能够被接入到仿真系统中,作为整个低轨卫星星座的一个节点参与到星座组网和协议验证中来.在目前的设计中,卫星实物节点通过 VXLAN 协议接入到仿真系统中.与分布式部署环境下的载体主机不同,仿真系统无法对实物节点进行干预和控制,那么如何对实物节点的相关链路通断进行模拟呢?在这里,我们可以将仿真节点与实物节点进行映射,通过对仿真节点链路通断的控制,间接做到实物节点的链路通断模拟.类似链路的跨主机模型,仿真系统基于 Vxlan 设备实现实物节点与仿真节点的映射.

卫星实物节点和虚拟仿真节点的映射方案见图 8.主机 A 是卫星实物节点,主机 B 承载了 7 个卫星仿真节点(其中 G 为实物节点对应的虚拟仿真节点)和 1 个地面终端仿真节点.主机 A 和节点 G 各自包含 5 个 Vxlan 设备,构成 5 条隧道连接,并通过 Bridge 与 Veth-pair 完成实物节点和卫星节点 A~F 以及终端节点的互连.完成映射后,流量的星上转发由卫星实物节点负责,链路拓扑更新的维护需要虚拟仿真节点 G 的参与.使用类似方法也可以完成地面终端的接入仿真,细节不再赘述.

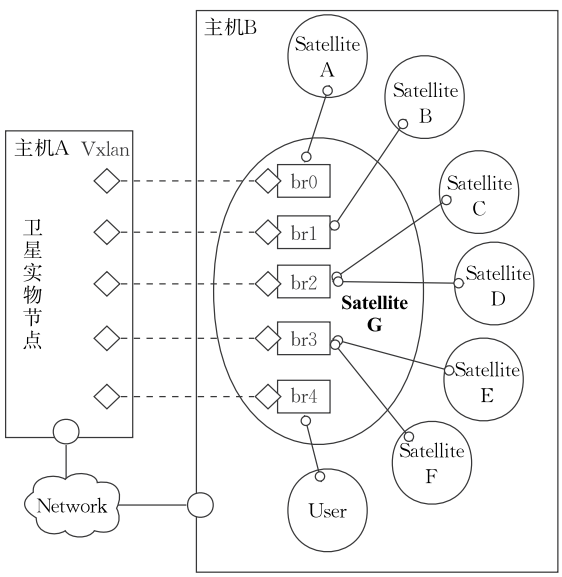

图 8 卫星实物节点(A)和虚拟仿真节点(G)的映射

4.5　基于多线程的仿真事件高效调度

仿真系统中存在大量的仿真事件，依据描述对象的差异，这些事件可分为网络事件、节点事件、链路事件和 Shell 配置四大类. 其中，网络事件包括仿真网络（如星上网络、地面接入网络）的创建、更新和删除等. 节点事件包括仿真节点（如卫星节点、地面终端节点）的创建、删除、位置更新、内部路由更新等. 链路事件包括链路（如星间链路和星地链路）的连通和断开. 常见的 Shell 配置有容器的启动与删除、容器内进程的启动与中止、容器内路由表的配置、Veth-pair 或 Bridge 的创建和配置、VETH 与网络命名空间或虚拟网络设备的绑定等. 依照层级关系，网络事件可以进一步分解为多个节点事件和链路事件，而节点事件或链路事件则可以进一步分解为多条 Shell 配置操作. Shell 配置操作往往是耗时的 I/O 密集型操作，Shell 命令之间存在阻塞，这就导致仿真系统在单线程运行模式下的执行效率是非常低的. 考虑到不同仿真节点及链路的配置、不同 VETH 的配置存在并发调度的可能，我们研究利用多线程提升仿真系统的执行效率. 当然，这里需要考虑调度执行过程中的依赖关系，例如 VETH 的配置通常存在先后关系，需要串行调度. 此外还需要考虑组合调度的情况，如多个仿真节点或多条仿真链路均可以并发创建，但仿真链路必须在相关仿真节点创建完毕后才能创建.

为了方便阐述，任务之间的串行调度关系记作“->”，任务之间的并发调度关系记作“,”. 那么两个任务 a 和 b 之间的串行调度可表示为“a->b”，并发调度可表示为“a,b”. 串行调度可以通过将多个任务交由同一个线程执行来实现，多个任务将按照先后顺序被执行. 并发调度可以通过将多个任务交由多个线程执行来实现，多个线程并发执行并独立的运行各自的任务，且无法保证任务执行完毕时的先后顺序. 对于更加复杂的组合调度，需要多个线程使用同步机制配合执行. 例如对于组合调度“a->(b,c)->d”，可引入两个线程 T1 和 T2. 首先 T1 运行 a；等到 a 完成后 T2 运行 c，同时 T1 运行 b；等到 b 和 c 都完成后由 T1 或 T2 运行 d. 其中，T1 可通过条件变量的 Wait 操作等待 T2 执行完成，T2 在执行完成时通过条件变量的 Signal 操作通知 T1.

考虑到仿真系统中大量的并发任务以及复杂的任务间依赖关系，完全由程序员手工维护条件变量和互斥锁复杂度非常高，且有可能导致程序代码变得难以维护. 为了降低描述和维护高并发任务间依赖关系的复杂度，本文给出一种“任务树”机制，使用任务间的父子关系描述复杂的组合调度逻辑，对条件变量、互斥锁等底层调度原语进行抽象和封装，实现对高并发线程调度复杂实现细节的屏蔽.

我们将一个任务节点 F 的内部结构定义为 F{pre->(sons):after}，其中，F 为任务节点名称，pre 为 F 的前置过程，after 为 F 的后置过程，sons 为 F 的子任务集合. 任务节点之间通过父子关系形成一棵任务树，称根节点为根任务，其他节点为子孙任务. 任务节点通过互斥锁与条件变量实现等待自身的 Wait 方法和通知外部的 Signal 方法. 这些互斥锁与条件变量的细节被封装在任务节点内部，使得处理高并发线程调度的逻辑细节得以简化. 每个任务节点分配一个执行线程，其执行具体流程为：首先执行前置过程 pre；然后使用多个新线程并发调度执行子任务 sons；之后执行后置过程 after；如果当前任务是根任务则需要递归执行子任务的 Wait；最后更新任务状态为完成并执行自身的 Signal.

在上面的流程中，任务节点里的 pre 和 sons 存在串行调度的关系，sons 和 after 存在并发调度的关系，sons 中的子任务之间也是并发调度关系. 任务节点从根任务开始被递归的在新线程中调度执行，最终由根任务线程递归的等待所有任务节点执行完成. 任务树的执行过程中只会阻塞根任务线程. 另外，可使用线程池技术提前创建好一批线程以降低每次动态新建线程的额外性能开销. 图 9 展示了仿真系统中任务树的执行过程.

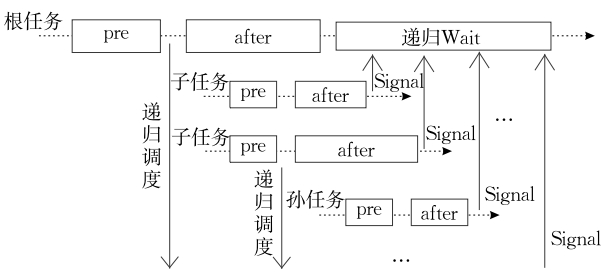

图 9　任务树的执行过程

4.6　一种集中式路由的仿真案例

拓扑动态变化的仿真网络为所承载的用户协议创造了逼真的低轨卫星网络运行环境. 用户协议既可以直接在多个卫星节点上进行分布式部署，也可以引入集中式的协议控制机制. 以路由协议为例，由于容器提供了进程及网络隔离的主机运行环境，可在卫星 Docker 容器中运行开源路由软件 Quagga[25]，实现 RIP、OSPF、BGP 等分布式路由协议的部署. 另

一方面，也可以吸收软件定义网络[26]的思想，在控制平面实现集中式的路由控制模型. 控制平面能够基于规律性拓扑变化的数学模型或收集到的全局网络拓扑信息计算并产生路由更新事件，进而驱动数据平面卫星节点的路由配置，实现集中式的路由控制. 其中，卫星 Docker 容器的路由配置同样可以通过 Shell 调用 Linux iproute2 工具包实现. 此外，如果用户希望开发新的分布式路由协议，也可以为每个 Docker 容器配置独立的控制平面进程，通过编写控制平面进程代码，实现用户自定义的路由协议. 当用户希望定制新的数据平面功能时，既可以修改 Docker 容器中运行的 Linux 内核网络协议栈代码，也可以基于 DPDK 或 P4 等可编程数据平面工具实现 Docker 容器中转发逻辑功能的灵活定制.

下面给出基于本文仿真测试床的一种集中式路由控制机制的仿真方案. 系统的控制平面负责维护星间链路的实时拓扑. 星间链路的通断事件会导致星间链路拓扑的更新，触发控制平面根据当前最新拓扑调用最短路径算法(Dijkstra 算法)重新计算每个卫星的路由表. 然后控制平面通过路由更新事件驱动数据平面上的每个卫星节点进行路由更新，最终以集中式控制的方式实现卫星节点之间的路由收敛. 图 10 展示了集中式路由更新的流程.

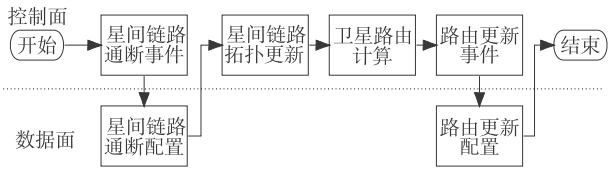

图 10 一次集中式路由更新流程

星间链路的通断事件可分为两类，一类是内部时钟激励下的规律性链路通断(用于仿真正常情况下低轨卫星星座周期性的拓扑变化)，另一类是外部用户交互引发的异常链路通断(用于仿真卫星链路因为故障等原因产生的异常中断). 由于两种通断事件都会触发控制平面星间链路的拓扑更新，仿真系统因此能够支持对规律性拓扑变化和抗毁路由协议的仿真. 需要注意的是，控制平面星间链路的拓扑更新以及集中式路由的计算均应在数据平面完成上一轮的链路通断配置之后进行，使得路由计算的链路拓扑输入能够反映最新的数据平面链路通断状态，从而保证数据平面在进行新的路由更新配置时自身的链路状态已经完全稳定. 另外，不同卫星节点的路由更新配置可以并发执行，依赖前述的任务树机制实现高效的并发任务调度.

5 仿真系统性能评测

5.1 实验环境与评测方法

我们基于 7179 行 C++代码实现了上述系统，并评测了系统在单机部署下的性能. 我们使用一台 x86 物理主机作为部署平台. 该主机包含主频为 3.4 GHz、核心数为 4 的 i7-6700 处理器和 16 GB 的 DDR3 DRAM，支持 8 个硬件线程，并运行内核版本号为 4.15.0 的 Linux 操作系统. 仿真节点的容器镜像基于 Ubuntu 打包，安装有 iproute2、iperf、ping 等常用网络工具. 性能评测以系统的内存占用、CPU 占用和仿真耗时为主要指标，其中内存占用与 CPU 占用通过 Linux top 工具测算. 考虑到 CPU 利用率是瞬时值，我们采用根据时间片取平均值的方法，将多次 top 命令采集到的瞬时值取平均值作为 CPU 利用率的测量结果. 实验中观察到，当 CPU 平均利用率接近 70%时，存在多个瞬时值接近 100%的情况. 因此我们把 70%的 CPU 平均利用率定义为仿真系统计算资源饱和的阈值. 卫星星座的实时拓扑由 STK 软件导出，包含了 LEO-48 星座的拓扑变化情况. 链路的通断由 Linux 脚本控制，链路通断时间取决于系统性能和线程机制. 我们使用有线链路的通断来模拟无线链路，因此当前链路上不存在误码等情况. 系统目前主要用于仿真网络层及以上协议，所以对物理层假设较为理想. 仿真链路带宽同样取决于机器性能，目前基于 iperf 在当前硬件配置下可以实现端到端 100 Mbps 速率的打包测试.

5.2 系统内存占用

5.2.1 仿真节点数对内存使用量的影响

仿真节点对内存的占用主要体现在 Docker 容器、Bridge 和 VETH 上. 理论上，仿真节点越多，消耗内存越大，且卫星节点比地面终端节点消耗更多存储空间(卫星节点相比地面节点额外包含 1 个 Bridge). 实验分为卫星节点和终端节点两组，实验结果如图 11 所示. 实验结果显示，100 个卫星节点一共消耗 473 MB 内存；100 个终端节点一共消耗 392 MB内存(请注意，图 11 中曲线显示了整个操作系统和应用程序的内存消耗总和). 由此可知，单个卫星节点和单个终端节点的内存使用量差距刚好是 1 个 Bridge(约为 1 MB). 根据单个仿真节点 5 MB 的内存占用量推算，理论上，16 GB 的内存空间最大能够装载 3276 个仿真节点.

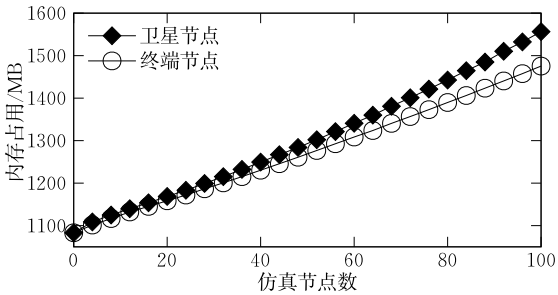

图 11　系统内存占用随仿真节点数目变化的关系

5.2.2　仿真链路数对内存使用量的影响

星间链路的内存使用主要体现在链路两侧的 VETH 上. 星间链路数目越多，内存使用数量越大. 星地链路两侧天线由于依赖仿真节点维护，不再占用额外内存. 图 12 展示了 15 个卫星节点和 100 个终端节点所构成的星间链路和星地链路的内存使用情况. 实验结果表明，内存占用与星间链路数呈正相关，且并没有随着星地链路数的增加而明显增长. 此外，内存使用量并不是严格随着星间链路数目的增长而线性增加(可能与 Veth-pair 实现有关). 100 条星间链路共占用约 200 MB 内存，也就是说每条星间链路约占 2 MB 内存. 由此推算，16 GB 的主机内存理论上最大可以承载 8196 条星间链路.

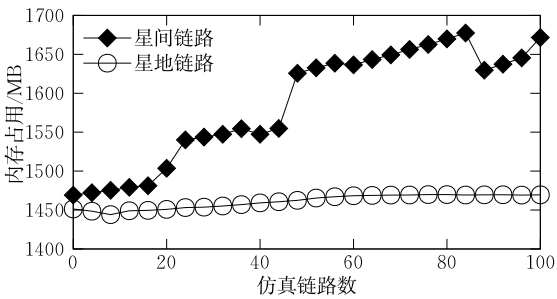

图 12　系统内存占用随仿真链路数目变化的关系

5.3　系统 CPU 占用

5.3.1　链路通断对 CPU 利用率的影响

星间链路的通断通过 VETH 的新建和删除实现，星地链路的通断通过 VETH 与 Bridge 之间的绑定和解绑实现. 我们分别测量了星间链路连通、星间链路断开、星地链路连通和星地链路断开对 CPU 利用率的影响. 图 13 所示实验结果表明，星间链路通断数的增大导致 CPU 利用率的上升，40 条星间链路同时通断使计算资源达到饱和状态. 星地链路的通断大约消耗 30%的 CPU 利用率，CPU 利用率不会随着通断数增大发生明显变化，且星地链路断开的 CPU 利用率通常小于连通时的利用率. 由此可见，相比 VETH 与 Bridge 之间的绑定和解绑操作，VETH 的创建和删除操作消耗更多的 CPU. 此外，相比 VETH 与 Bridge 之间的解绑操作，VETH 与 Bridge 之间的绑定操作消耗更多的 CPU.

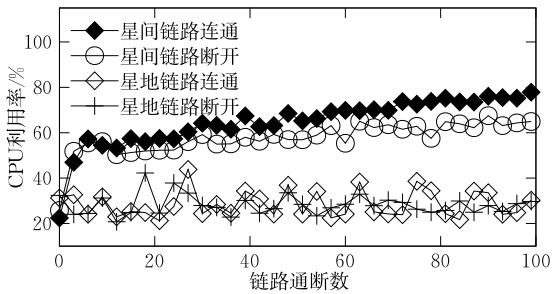

图 13　瞬时链路通断数对 CPU 负载的影响

5.3.2　流量吞吐率对 CPU 利用率的影响

当链路上承载更高速率的网络流量时，CPU 利用率将会随之提高. 我们在实验中部署了一定数量的发包终端，并通过增加发包终端数量的方式实现网络中流量总吞吐率的增大. 我们按照终端发包速率不同将实验分为四组，每组发包终端数从 0 增加到 100，实验结果见图 14. 实验结果表明前三组的 CPU 利用率随着流量速率增加而不断上升. 例如 10 Mbps 组的 CPU 利用率从 0%递增至 70%，达到计算资源的饱和阈值. 第四组实验以发包终端数为 16 的点为分界点，CPU 利用率先增后减. 因为 100 Mbps 时宿主机的 I/O 代替计算资源成为了仿真系统新的性能瓶颈，同时也限制了流量处理吞吐率进一步的提高. 实验结果表明，当前使用的主机能够承载的最大网络流量总吞吐率约为 1.6 Gbps.

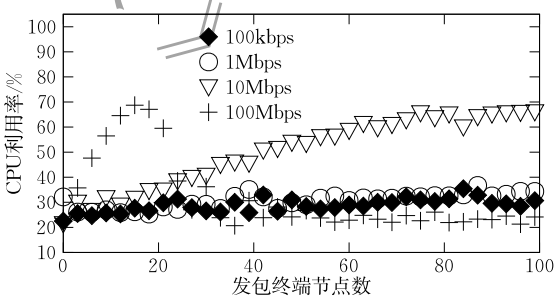

图 14　网络吞吐率的变化对 CPU 负载的影响

5.3.3　多线程对 CPU 利用率的影响

仿真系统通过多线程调度挖掘多核 CPU 的计算潜力. 我们分别在创建卫星节点、删除卫星节点、创建卫星链路和删除卫星链路四种操作下，评估多线程对 CPU 利用率的影响. 前两组实验创建和删除了 15 个卫星节点；后两组实验创建和删除了 100 条星间链路. 仿真系统基于多线程完成上述任务，对应的 CPU 利用率见图 15. 实验结果显示，CPU 利用率在工作线程数为 8 时最高且达到计算资源的饱和

阈值，相比单线程增加了 45%. 当工作线程数比宿主机硬件线程数小时，由于多线程提高了 I/O 任务的处理并发度，CPU 利用率随着工作线程数的增加而不断增长. 当工作线程数达到并超过宿主机的硬件线程数之后，工作线程数的继续增加对 CPU 利用率的影响并不明显，甚至出现了不升反降的现象. 这是因为 I/O 成为此时系统的真正瓶颈，且过多的线程切换也会带来额外的性能开销.

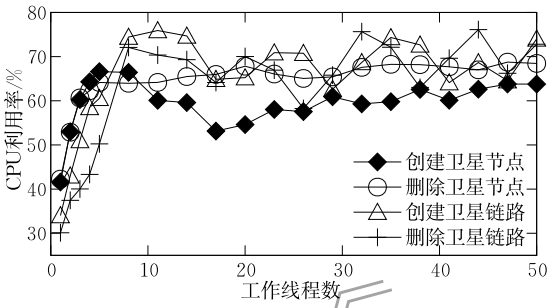

图 15 工作线程数的变化对 CPU 负载的影响

5.4 系统仿真耗时

5.4.1 仿真节点和链路的创建及删除耗时

我们分别测量了创建/删除卫星节点、创建/删除终端节点、创建/删除卫星链路所花费的时间，实验结果如图 16 所示. 实验结果显示虚拟化节点及链路的创建与删除时间随仿真实例数量的增多而逐渐增长，且删除时间远低于创建时间，其中卫星节点和卫星链路的创建时间最大. 从数值上看，创建 100 条卫星链路耗时 80.6 s. 创建和删除 1 个卫星节点的耗时分别是 1.8 s 和 0.9 s，而创建和删除 100 个卫星节点的耗时分别是 78.5 s 和 37.4 s(并非 180 s 和 90 s). 由此可见，基于多线程的仿真事件并发处理机制使创建和删除卫星节点的时间分别缩短了 56%和 58%.

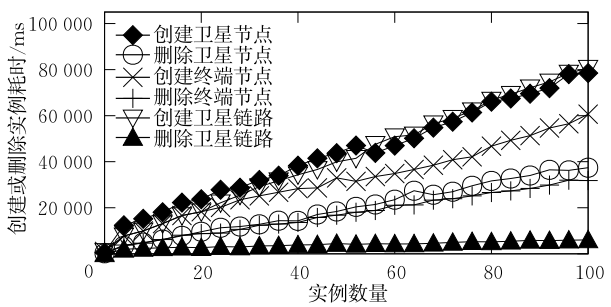

图 16 仿真节点和链路的创建及删除耗时

5.4.2 仿真链路的连通及断开耗时

我们分别测量了仿真系统中星间链路连通、星间链路断开、星地链路连通和星地链路断开四个场景所花费的时间. 每个场景中的链路数目从 0 开始递增到 100，实验结果见图 17. 从实验结果中，我们可以发现，星间链路的连通耗时与断开耗时大致接近，而星地链路的连通耗时比断开耗时小，且星间链路的通断时间远远大于星地链路的通断时间. 从数值上看，1 条星间链路的通断时间大约是 0.2 s，而 100 条星间链路的通断时间总共为 12.7 s，该时间远远小于创建或删除 100 条星间链路所耗费的时间. 此外，仿真系统的多线程调度机制也将星间链路通断所消耗的时间缩短了 37%.

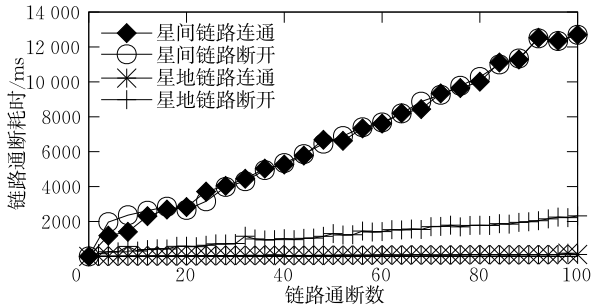

图 17 仿真链路的连通及断开耗时

5.5 动态拓扑下的网络性能

5.5.1 动态拓扑下的端到端时延和跳数

图 18 显示了动态拓扑下端到端时延和跳数的变化情况. 我们选取了 s8(第一轨道第八颗星)和 s28(第四轨道第四颗星)两颗卫星作为路径的两端. 其中时延基于 iperf 发包工具测量得到，路由的跳数基于 traceroute 工具测量得到. 考虑到真实链路存在传输时延以及卫星节点存在转发时延，我们使用 Linux tc 工具在路由器的出端口处模拟了 10 ms 和 100 ms 两种逐跳时延的情况. 可以看到，随着卫星星座的拓扑变化和路由的重新计算，两颗距离较远的卫星之间的路由跳数也呈现周期性的变化，跳数的差别也直接反映在了端到端时延的动态变化上.

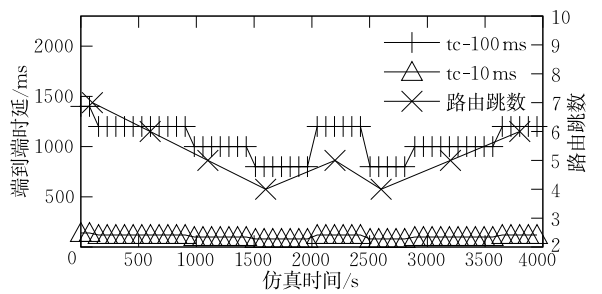

图 18 动态拓扑下端到端时延(tc-100 ms/10 ms)和跳数

5.5.2 动态拓扑下的丢包率和吞吐率

图 19 显示了低速率发包(30 Mbps)情况下，基于 iperf 发包工具测量得到的动态网络拓扑下的丢包率和端到端吞吐率. 我们可以看到，无论星座拓扑如何变化，端到端的吞吐率总体较为稳定(即为 iperf 设定的发包速率). 但在拓扑变化的瞬间，由于路由计算过程存在一定的时延，产生了一定的丢包情况.

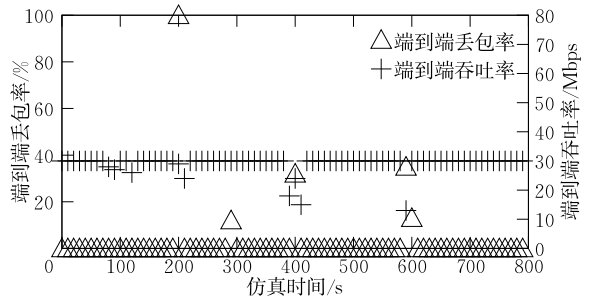

图 19　动态拓扑下的丢包率和吞吐率(低速率 30 Mbps)

与之对应的是,端到端吞吐率也会产生相应的抖动.

图 20 显示了高速率发包(100 Mbps)情况下,动态网络拓扑下的丢包率和端到端吞吐率. 当发包速率较高时,由于触及仿真机器性能的上限,iperf 测量到的吞吐率始终处于较大的抖动状态,并且小于 iperf 设定的发包速率. 相应的,系统中的丢包事件也变得非常频繁. 由于仿真机器计算资源紧张,甚至在路由没有发生改变的情况下,系统都出现了较大的丢包概率.

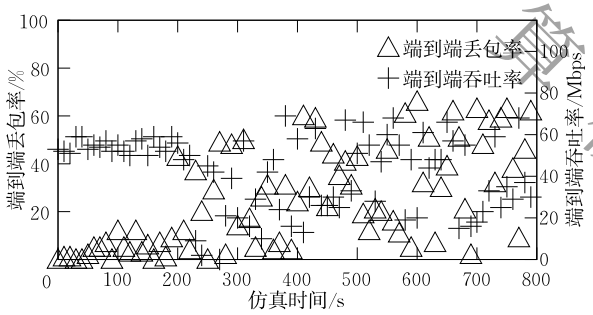

图 20　动态拓扑下的丢包率和吞吐率(高速率 100 Mbps)

5.5.3　动态拓扑下的路由协议计算开销

图 21 显示了动态拓扑下的路由协议计算开销,使用 CPU 利用率进行描述. 当发生拓扑变化时,系统需要计算新的路由,并将增量变化的路由通过 Linux route 命令下发到各个卫星节点的内核路由表中,因此产生了较大的计算开销. 随着星座的规律运动,拓扑变化周期性出现. 图 21 反映了上述周期性的路由协议计算开销的变化过程.

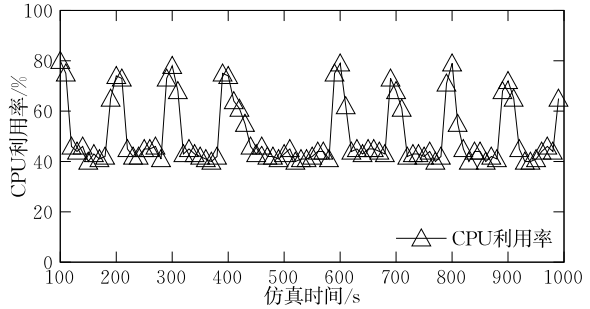

图 21　动态拓扑下的路由协议计算开销(CPU 利用率)

5.5.4　动态拓扑下的路由算法收敛时间比较

图 22 展示了在卫星拓扑规律动态变化时,使用仿真平台验证经典分布式 OSPF 路由协议和基于轨道预测的改进 OSPF 路由协议(即 OPSPF 协议[27])时的路由收敛时间对比情况. 不同于经典的 OSPF 协议,OPSPF 协议中的每个卫星节点提前掌握卫星星座运动规律,能够在任意时刻预判出其他卫星节点的位置,从而可以计算得到实时的星座拓扑情况,并快速计算得到最优路由. 可以看到,经典的 OSPF 路由协议因为并不提前知晓低轨卫星网络的通断情况,需要通过 hello 包和 time-out 机制学习到拓扑的实时变化并向全网进行路由洪泛,从而导致了较高的收敛时间. 而基于轨道预测的 OPSPF 协议因为在各个卫星节点上掌握了实时的星座拓扑,能够在较短时间内直接计算更新自身的路由表,从而消除了原本做分布式路由学习的时间.

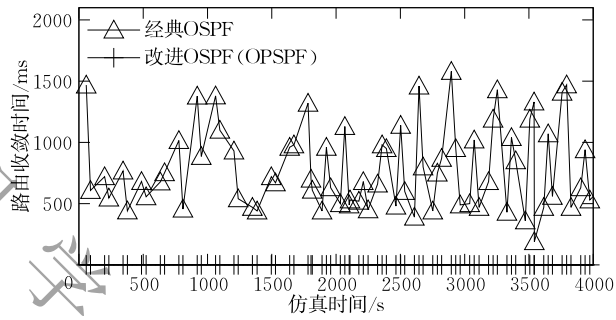

图 22　经典 OSPF 协议和改进 OSPF 协议(OPSPF 协议)在仿真平台上的路由收敛时间对比

5.6　半实物仿真

在全仿真环境中,两个卫星节点之间的通信基于简单的 IP 报文. 而在半实物仿真环境中,半实物节点所在的物理主机与其余仿真网络卫星节点所在的物理主机之间的通信基于 VXLAN 协议构建的隧道. 也就是说,当报文穿梭在半实物主机与仿真主机之间时,需要进行 VXLAN 报文头的封装和解封装操作,从而产生额外的处理开销. 此外,半实物仿真的端到端转发跳数也会因为引入虚拟映射节点(见 4.4.3 节)而有所增多,进而也会产生额外的端到端时延. 图 23 显示了半实物仿真和全仿真环境下的端到端时延比较. 受限于硬件资源,目前我们的实验使用两台虚拟机作为半实物节点主机和仿真节点主机. 我们可以看到,无论是半实物仿真还是全仿真环境,由于周期性的最优路由计算,端到端的时延均因为网络拓扑变化而周期性的改变. 半实物仿真因为 VXLAN 报文头的封装和解封装操作以及端到

端转发跳数的增多，比全仿真环境增加了约 0.2 ms 的端到端时延. 如果测量处于两台不同虚拟机的两个半实物节点之间的端到端时延，那么增量将会翻倍达到 0.4 ms. 当前的实验结果没有考虑额外的链路和排队时延. 如果使用 Linux tc 引入实际的链路和排队时延，半实物仿真所增加的额外时延相对来说可以忽略不计.

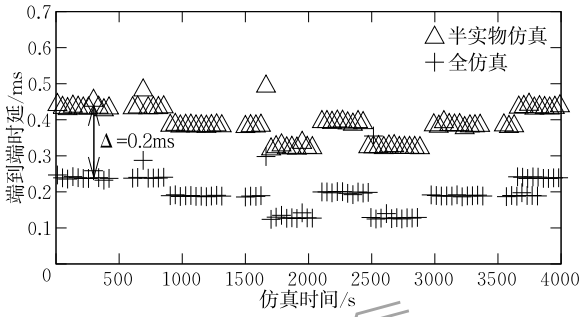

图 23 半实物仿真和全仿真的端到端时延比较

6 总结与展望

本文提出一种基于容器的低轨卫星网络协议仿真测试床的设计和实现方案. 测试床采用控制平面和数据平面分离的架构设计，在控制平面基于离散事件模拟技术建立卫星网络实时拓扑变化的数学模型，在数据平面基于虚拟化技术建立仿真网络的实体，通过控制平面对数据平面的动态配置实现低轨卫星网络拓扑变化的精准复现. 本文阐述了系统的分布式部署方案，使系统具备横向扩容能力，满足超大规模组网节点的仿真需求；同时给出了一种基于节点映射的半实物仿真方案，使系统具备接入外部业务流量的能力. 为了提升系统的仿真效率，本文提出了一种基于多线程的任务调度机制以充分挖掘多核处理器的计算潜力. 此外，本文还给出了一种集中式路由协议的仿真案例. 性能评测结果表明，仿真测试床在多核处理器主机上能够承载大量网络节点的真实流量处理任务. 相比单线程处理，基于多线程的高效任务调度机制通过提升 CPU 利用率大大缩短了仿真耗时. 在后续的工作中，一方面，我们将对除卫星网络链路物理通断以外的其他链路特性（如误码率、排队时延、带宽约束等）进行抽象和描述，从而进一步提升低轨卫星星座组网仿真的逼真程度；另一方面，我们将尝试把网络协议测试床云化，并通过云服务的方式为外部多个租户并发提供低轨卫星网络协议仿真的能力.

致 谢 审稿专家对论文提出了宝贵的意见，编辑对作者给予了耐心帮助，在此表示衷心感谢！

参 考 文 献

[1] Chen Shan-Zhi. Analysis of LEO satellite communication and suggestions for its development strategy in China. Telecommunications Science, 2020, 36(6): 1-13(in Chinese)
(陈山枝. 关于低轨卫星通信的分析及我国的发展建议. 电信科学, 2020, 36(6): 1-13)

[2] Li He-Wu, Wu Qian, Xu Ke, et al. Progress and tendency of space and earth integrated network. Science & Technology Review, 2016, 34(14): 95-106(in Chinese)
(李贺武, 吴茜, 徐恪等. 天地一体化网络研究进展与趋势. 科技导报, 2016, 34(14): 95-106)

[3] Bhattacherjee D, Kassing S, Licciardello M, et al. In-orbit computing: An outlandish thought experiment?//Proceedings of the 19th ACM Workshop on Hot Topics in Networks. Chicago, USA, 2020: 197-204

[4] Del Portillo I, Cameron B G, Crawley E F. A technical comparison of three low earth orbit satellite constellation systems to provide global broadband. Acta Astronautica, 2019, 159: 123-135

[5] Kassing S, Bhattacherjee D, Águas A B, et al. Exploring the "internet from space" with Hypatia//Proceedings of the ACM Internet Measurement Conference. Pittsburgh, USA, 2020: 214-229

[6] Wang J, Li L, Zhou M. Topological dynamics characterization for LEO satellite networks. Computer Networks, 2007, 51(1): 43-53

[7] Tsunoda H, Ohta K, Kato N, et al. Supporting IP/LEO satellite networks by handover-independent IP mobility management. IEEE Journal on Selected Areas in Communications, 2004, 22(2): 300-307

[8] Zhang Xue-Bei. A Design and Implementation of Mobility Management in LEO Satellite Networks[M. S. dissertation]. Beijing University of Posts and Telecommunications, Beijing, 2017(in Chinese)
(张雪贝. 一种基于 LEO 卫星的移动性管理方案的设计与实现[硕士学位论文]. 北京邮电大学, 北京, 2017)

[9] Su Si-Rui. Research on Low-Orbit Satellite Mobile Handoff Protocol Based on Terminal Geographical Location [M. S. dissertation]. Beijing University of Posts and Telecommunications, Beijing, 2019(in Chinese)
(苏思睿. 基于终端地理位置的低轨卫星移动切换协议研究[硕士学位论文]. 北京邮电大学, 北京, 2019)

[10] Lim J, Klein R, Thatcher J. Good technology, bad management: A case study of the satellite phone industry. Journal of Information Technology Management, 2005, 16(2): 48-55

[11] Chertov R, Fahmy S, Shroff N B. Fidelity of network simulation and emulation: A case study of TCP-targeted denial of service attacks. ACM Transactions on Modeling and Computer Simulation (TOMACS), 2009, 19(1): 1-29

[12] Guo L, Lee J Y B. On TCP simulation fidelity in ns-2//Proceedings of the 14th ACM International Symposium on QoS and Security for Wireless and Mobile Networks. Montreal, Canada, 2018: 55-62

[13] Merkel D. Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014, 2014(239): 2

[14] Qi J, Li Z, Liu G. Research on coverage and link of multi-layer Satellite Network based on STK//Proceedings of the 10th International Conference on Communications and Networking in China (ChinaCom). Shanghai, China, 2015: 410-415

[15] Claassen J, Koning R, Grosso P. Linux containers networking: Performance and scalability of kernel modules//Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium. Istanbul, Turkey, 2016: 713-717

[16] El Saddik A. Digital twins: The convergence of multimedia technologies. IEEE Multimedia, 2018, 25(2): 87-92

[17] Wang C J. Structural properties of a low earth orbit satellite constellation-the walker delta network//Proceedings of the MILCOM'93-IEEE Military Communications Conference. Boston, USA, 1993, 3: 968-972

[18] Henderson T R, Lacage M, Riley G F, et al. Network simulations with the ns-3 simulator. SIGCOMM Demonstration, 2008, 14(14): 527

[19] Puttonen J, Herman B, Rantanen S, et al. Satellite network simulator 3//Proceedings of the Workshop on Simulation for European Space Programmes (SESP). Noordwijk, The Netherlands, 2015, 24: 26

[20] Chen Yu-Jie. A Design and Implementation of Routing Protocol in LEO Satellite Networks [M. S. dissertation]. Beijing University of Posts and Telecommunications, Beijing, 2018(in Chinese)
(陈愈杰. 一种基于 LEO 卫星的路由协议的设计与实现[硕士学位论文]. 北京邮电大学, 北京, 2018)

[21] Zhang Y, Zhang Y, Li X, et al. Simulation platform of LEO satellite communication system based on OPNET//Proceedings of the 4th Seminar on Novel Optoelectronic Detection Technology and Application. Nanjing, China, 2018, 10697: 106975C

[22] Lantz B, Heller B, McKeown N. A network in a laptop: Rapid prototyping for software-defined networks//Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. Monterey, USA, 2010: 1-6

[23] Pfaff B, Pettit J, Koponen T, et al. The design and implementation of Open vSwitch//Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). Oakland, USA, 2015: 117-130

[24] Mahalingam M, Dutt D G, Duda K, et al. Virtual eXtensible Local Area Network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks. RFC 7348, 2014: 1-22

[25] Jakma P, Lamparter D. Introduction to the Quagga routing suite. IEEE Network, 2014, 28(2): 42-48

[26] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74

[27] Pan T, Huang T, Li X, et al. OPSPF: Orbit prediction shortest path first routing for resilient LEO satellite networks//Proceedings of the 2019 IEEE International Conference on Communications (ICC). Shanghai, China, 2019: 1-6

PAN Tian, Ph. D., associate professor. His research interests include satellite network protocols, data center networks, high-speed programmable network systems, etc.

LI Xing-Chen, M. S. candidate. His research interests include LEO satellite networks, network emulation systems, etc.

XUE Wen-Hao, M. S. candidate. His research interests include LEO satellite networks, satellite network security, etc.

BIAN Zi-Zheng, M. S. candidate. His research interests include high-speed programmable network systems, network virtualization, etc.

HUANG Tao, Ph. D., professor. His research interests include future network architecture, software-defined networks, content-centric networks, etc.

LIU Yun-Jie, professor, academician of Chinese Academy of Engineering. His research interests include future network architecture, network operating systems, software-defined networks, etc

Background

As the manufacturing and launching costs of LEO satellites decrease, LEO satellite networks have reemerged as a hot research topic in recent years. For the research purpose, a high-fidelity network testbed needs to be built for LEO satellite networks to provide an environment for evaluating and verifying the LEO satellite network protocols

on the ground before launching these satellites into space. The cost-effective network testbed is a necessary complement to the expensive in-orbit experiments on satellite network protocols. However, the topology of the LEO satellite network is extremely dynamic, the scale of the satellite constellation is enormous, and the network traffic pattern is complex. To this end, the network testbed needs to enable the accurate modeling of the satellite constellation topologies, the emulation of satellite/terminal nodes over distributed hosts, and the loading of real network traffic, all of which produce extreme challenges to the testbed design. Existing solutions either leverage network simulation, which is not that real and limited by the computer performance, or relies on network virtualization, which is however not dedicated to accurately modeling and emulating the dynamic satellite network topologies. Furthermore, LEO satellite constellations contain a huge number of networked nodes including the satellites in the space and the terminal nodes on the ground. Therefore, the network testbed is also expected to be highly performant and scalable to carry the massive number of these networked nodes.

To address the aforementioned requirements, we build an LEO satellite network testbed via integrating the discrete event simulation technology and the virtualization-based emulation technology. First, we establish a mathematical model to abstract the dynamic satellite constellation, which produces discrete events of the satellite link on/off under the drive of an internal clock. Second, we use the Docker containers to emulate the LEO satellites/terminal nodes, and use the Linux virtual network devices to emulate the inter-satellite links and satellite-ground links. Third, we leverage VXLAN tunnels to support distributed virtual node deployment and hardware-in-loop emulation, making the network testbed have good horizontal expansion capabilities to support current and future mega-scale satellite constellations. Finally, a concurrent scheduling mechanism based on multithreading is developed for performant scheduling of discrete events, taking advantage of the modern multi-core CPUs.

We conduct an extensive performance evaluation of the proposed testbed. Evaluation results indicate that the LEO satellite network testbed well satisfies our design requirements. On a multi-core machine, the testbed can carry 3276 emulated nodes and sustain a traffic throughput of 1.6Gbps. Compared to the single-threaded scheduler, the concurrent task scheduler based on multithreading greatly improves the CPU utilization by 45% so that the time for emulation is significantly reduced.

This work is supported by the National Key Research and Development Program of China (No. 2019YFB1802600), the National Natural Science Foundation of China (No. 61702049), and the Fundamental Research Funds for the Central Universities.