计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(5): 1307−1329, 2025

OpenPlanner: 一个开源的时间敏感网络流量规划器

姜旭艳[1] 全 巍[1] 付文文[1] 张小亮[2,3] 孙志刚[1]
[1](国防科技大学计算机学院 长沙 410003)
[2](南京航空航天大学计算机科学与技术学院 南京 211106)
[3](湖南汽车工程职业大学信息工程学院 湖南株洲 412001)
(jiangxuyan@nudt.edu.cn)

OpenPlanner: An Open-Source Traffic Planning for Time-Sensitive Networking

Jiang Xuyan[1], Quan Wei[1], Fu Wenwen[1], Zhang Xiaoliang[2,3], and Sun Zhigang[1]
[1] (*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha* 410003)
[2] (*College of Computer Science and Technology*, *Nanjing University of Aeronautics and Astronautics*, *Nanjing* 211106)
[3] (*College of Information Engineering*, *Hunan Automotive Engineering Vocational University*, *Zhuzhou*, *Hunan* 412001)

Abstract Time-sensitive networking (TSN) has emerged as a primary choice for communication in distributed real-time systems such as industrial automation, avionics, and automotive applications. TSN traffic planning aims to allocate conflict-free transmission times for time-sensitive frames while managing constraints related to network topology, resources, device capabilities, and stream requirements. The traffic planning problem is NP-complete. There is a need of quick development of open-source traffic planning software for both academia and industry. We introduce LOCAP, an architecture for TSN planning with interfaces named minimum collection of planning and general table of planning. LOCAP separates planning algorithms and tools, as well as planning software and hardware details. Based on LOCAP, we implement an open-source TSN planner called OpenPlanner. OpenPlanner integrates multiple algorithms that leverage satisfiability modulo theories and heuristics to solve planning problems. We evaluate the runtime and solution quality of various algorithms using OpenPlanner, highlighting the need for diverse planning algorithms in different TSN applications. To the best of our knowledge, OpenPlanner is the first open-source TSN planner. Its planning results have been deployed on multiple hardware platforms, including OpenTSN, Yinhe Hengxin TSN chip, and XZ-TTE. It has been applied in various systems such as satellites, unmanned vehicles, and artillery.

Key words time-sensitive networking (TSN); traffic planning; open-source; satisfiability modulo theories; time-aware shaper

摘 要 时间敏感网络(time-sensitive networking, TSN)在工业控制、航空电子和车载网络中具有广泛的应用前景.TSN流量规划是在拓扑结构、网络资源、设备能力和业务需求等多维约束下,为TSN交换机计算关键帧的无冲突发送时刻的过程,规划问题是一个NP完全问题.目前不论是学术界的TSN规划算法研究,还是工业界的TSN部署应用都急需一个开源的规划器软件.提出一种构件化、松耦合的TSN规划器软件架构LOCAP(loose-coupled component-based architecture of planner),通过规划参数最小集和规划结果通用表等接口规范设计,实现规划算法与规划工具、规划器软件与交换硬件实现的松耦合.OpenPlanner

收稿日期: 2023-09-28; **修回日期**: 2024-12-05
基金项目: 全国重点实验室基金项目(2023-KJWPDL-14); 湖南省研究生科研创新项目(CX20220013)
This work was supported by the Foundation of National Key Laboratory (2023-KJWPDL-14) and Postgraduate Scientific Research Innovation Project of Hunan Province (CX20220013).
通信作者: 全巍(w.quan@nudt.edu.cn)

是基于 LOCAP 架构使用 Python 语言编写的开源 TSN 规划器,其内嵌自研和第三方贡献的多个可满足性模理论规划算法和启发式规划算法. 基于 OpenPlanner 对不同算法的运行时间开销以及解的质量进行了评估,指出多样化的 TSN 应用场景需要不同的规划算法. 据调研,OpenPlanner 是目前唯一的开源 TSN 规划器,规划结果已部署到 OpenTSN 开源网络、银河衡芯 TSN 芯片以及芯准 TTE 等多个硬件平台,在卫星、无人车和火炮等多个系统中得到应用.

关键词 时间敏感网络;流量规划器;开源;可满足性模理论;时间感知整形器

中图法分类号 TP391

DOI: 10.7544/issn1000-1239.202330776 **CSTR:** 32373.14.issn1000-1239.202330776

时间敏感网络(time-sensitive networking, TSN)由一系列开放的 IEEE 802.1Q 标准组成[1].TSN 是对现有以太网交换技术的增强[2-3],旨在满足实时控制系统中高带宽、高确定性实时通信的需求[4].TSN 面向的实时系统包括车载电子系统[5]、航空电子系统[6-7]和工业制造系统[8-9]等. 近年来,国外的博通(Broadcom)、时触电子技术(TTTech)、美满(Marvell)、恩智浦(NXP)等单位,以及国内的国防科技大学和北京东土科技股份有限公司等单位,相继推出 TSN 芯片[10]. 国防科技大学牵头发起 OpenTSN 开源网络项目,为推动面向不同应用场景的 TSN 芯片定制提出开源解决方案.

TSN 保证通信确定性的基本思路首先是基于时间同步协议[11]实现网络中所有交换设备和端设备的时间同步. 接着,在全局时间基准基础上,每个交换机通过时间感知整形(time-aware shaper, TAS)实现对数据帧发送时刻的控制[12].TSN 利用时间同步和 TAS 为关键流量提供低延时、低抖动的确定性传输服务.

TAS 是 TSN 的核心机制[12]. TAS 的基本思想是为网卡和交换机每个输出接口上的每个输出队列设置一个由全局时间控制的“门控”状态. 只有门控状态为“开”时,输出队列中的数据帧才能调度输出. 数据帧在端到端交换路径上每个设备输出接口的输出队列中缓存,并在一个预先规划好的“开门”时刻调度输出,因此其在端到端路径上每个交换机上的交换延时都能被精确地控制,从而实现端到端延时的确定性.

TAS 的门控需要进行全局规划,全局规划由 TSN 规划算法完成[13]. 当 TSN 网络中存在多条具有不同交换路径和不同延时需求的流时,这些流的数据帧会在不同交换机、不同输出接口以及不同的输出队列中缓存. 为了同时保证这些流的延时确定性,需要对每个交换机、每个输出接口、每个输出队列的门控状态进行全局规划.TAS 的全局规划由规划算法完成. 规划算法在拓扑结构、网络资源、设备能力和业务需求等多维约束下,为关键数据帧分配无冲突的发送时刻,是个 NP 完全问题[14-19]. 学术界针对 TSN 规划问题提出了多种规划算法,包括基于整数线性规划(integer linear programming, ILP)理论的算法[16,20]、可满足性模理论(satisfiability modulo theories, SMT)的算法[15,19,21-22]、启发式算法[17,23-24]以及机器学习算法[18]等.

尽管目前存在丰富的研究工作关注 TSN 规划算法[13-14],但少有工作关注 TSN 规划器. 规划算法与规划器的不同之处在于,规划算法仅研究使用何种方法求解调度整形的配置,如 SMT, ILP 等. 而 TSN 规划器则是一个集场景构建、规划求解、结果编译等功能于一体的功能更加丰富的软件,TSN 规划算法只是 TSN 规划器的一个子功能模块.TSN 规划器是 TSN 应用部署工具链的重要环节,在 TSN 生态中发挥至关重要的作用[25].

然而,由于缺乏对 TSN 规划算法输入输出规范的定义,目前 TSN 规划器面临 3 个问题:一是灵活性不足. 现有规划器仅继承单一规划算法,难以满足不同应用场景的差异化需求. 例如 TTTech 公司开发的 Slate XNS 规划器[26]集成本公司开发的 2 个规划算法[15,21],仅适用于交换机存储资源丰富或流量传输要求较为宽松的场景. 二是转换成本高. 规划器被具体的硬件设备锁定,用户替换当前组网设备的成本是替换整套专用软硬件栈的成本. 三是创新迭代慢. 规划器代码不开源,学术界的最新成果难以快速集成到规划器为工业界所用.

借鉴开源软件定义网络控制器[27]和开源交换机操作系统[28]的经验,将开源作为手段设计 TSN 规划器是一个良好的解决方案. 设计开源 TSN 规划器主要面临 2 个挑战:一是规划算法与规划工具难以解耦. 规划工具是指应用测试集、结果验证和结果编译等工具,这些工具用于输入预处理和输出后处理;二

是规划器与交换硬件的实现难以解耦. 造成上述挑战的主要原因是规划算法的输入输出接口缺乏统一规范.

本文提出松耦合的构件化的开源规划器架构 LOCAP(loose-coupled component-based architecture of planner), 并设计规划算法的输入输出接口规范规划参数最小集(minimum collection of planning, min-col)和规划结果通用表(general table of planning, gen-tbl). 2021 年, 在 OpenTSN 规划器[29]研制的基础上, 提出了开源的 TSN 规划器 OpenPlanner[30]. 经过 2 年多的发展, OpenPlanner 基于 LOCAP 架构进行重构, 已集成多个自研和第三方贡献的规划算法, 规划结果部署到包括 OpenTSN 开源网络、银河衡芯 TSN 芯片和芯准 TTE 等多个硬件平台, 已在卫星、无人车和火炮等多个系统中得到应用, 为推动 TSN 的部署应用发挥重要作用.

1 TSN 规划的基础知识

1.1 TSN 的时间感知整形

时间感知整形(time-aware shaper, TAS)是 TSN 核心的流量整形机制, 为时间敏感(time-sensitive, TS)流提供确定性传输服务[12]. TS 流量①是指在 TSN 中具有最高优先级的流量, 它具有严格的周期性. 严格周期性是指源端产生数据帧的时间间隔是固定的. TS 流量要求 TSN 提供确定性传输服务, 也就是为流量保证有界的延迟和极低的抖动.

图 1 用一个 TSN 交换机展示 TAS 的原理. TAS 位于交换机的整形调度模块. 从输入端口进入交换机的流量, 经转发决策模块转发到整形调度模块, 经整形调度模块整形后从输出端口输出. 每个输出端口都包含一个整形调度模块. 图 1 中的整形调度模块采用 TAS 机制. TAS 基于时分复用的思想, 通过输出队列后端的门控控制数据帧的通行[12], 以保证数据帧能够在特定时间段内独占物理介质.

TAS 更详细的工作流程如下: 数据帧进入整形调度模块后, 映射模块首先根据映射表, 将不同的数据帧定向到不同的输出队列(①). TSN 交换机的每个输出端口一般包含 8 个输出队列, 与标准以太网相同. 为了便于表示, 图 1 的交换机只展示 2 个输出队列. TS 流量享有专用的输出队列, 与其他流量形成空间维度的隔离. 每个输出队列后端都有一个门控,

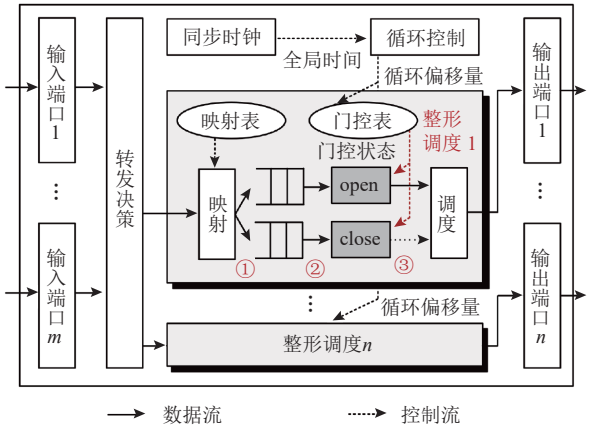


Fig. 1 Principle of time-aware shaper in TSN switches

图 1 TSN 交换机时间感知整形的原理

门控通过“开”和“关”两种状态控制队列内数据帧的通行(②). 门控为“开”时, 输出队列内的数据帧能够被调度至输出端口. 门控为“关”时, 输出队列内的数据帧只能等待(③). 网络中所有节点的时间是同步的, 即所有节点对网络时间有一个共识. 节点以该同步时钟为基准, 控制门控的切换. 换言之, 门控切换以网络同步时间为基准时间. 上述同步时间由 TSN 的时间同步协议维持[11]. TSN 时间同步超出本文的讨论范围, 在此不做详细介绍. 门控状态应当在哪一时刻进行切换, 以及每个状态应当维持多长时间, 由门控列表(gate control list, GCL)控制.

GCL 控制门控在哪一时刻打开或关闭, 以及打开或关闭状态的持续时间. GCL 的表项由门控状态(gateStates)和持续时间(timeInterval)组成. 前者指示门控状态, 后者则指示当前门控状态的持续时间.

GCL 以循环的方式工作, 其循环周期为所有 TS 流量的周期的最小公倍数, 将 GCL 的循环周期称作调度周期. 循环控制模块将全局时间切割成每个调度周期内的循环偏移量, GCL 根据循环偏移量控制门控状态.

综上, TAS 通过空间维度和时间维度的隔离, 保证 TS 流量的无冲突传输. 在空间维度, TAS 为 TS 流量分配专用的传输队列. 在时间维度, TAS 利用 GCL, 使得输出队列后端的门控互斥地打开, 特定时间段内只有某个队列内的流量能够被调度传输. 空间和时间维度的隔离使得 TS 流量能够在特定时间段内独占物理介质访问, TS 流量之间以及 TS 流量与其他流量之间的传输都是无冲突的.

经全局规划的、合理的 GCL 能够使得 TS 流量

① 流由源端和目的端之间的一系列数据帧组成.

在其转发路径上的每一个输出端口上都不会与其他流量发生冲突, 从而保证 TS 流量端到端的延时和抖动是确定的. GCL 的规划, 即 TAS 门控切换的规划由 TSN 流量规划算法完成, TSN 流量规划问题及规划算法将在 1.2 节进行介绍.

1.2 TSN 流量规划与规划算法

本节讨论的 TSN 流量规划是指为 TAS 规划门控的过程. 当 TSN 网络中存在多条具有不同交换路径和服务质量需求的流量时, 这些流量的数据帧在不同交换机、不同的输出端口和不同的优先级队列中进行缓存, 为了使得所有数据帧的传输无冲突从而保证延时确定性, 就需要对每个数据帧的发送时刻进行全局规划.

TSN 流量规划的具体工作是在拓扑结构、网络资源、设备能力和业务需求等多维约束下, 对每个关键帧在其路由路径的每个输出端口的发送队列及发送时刻进行全局规划. 关键帧的发送队列最终转换为交换机的输出端口的队列映射配置, 关键帧的发送时刻则转换为交换机的输出端口的门控配置. 图 2 是 TSN 流量规划的 1 个示例.

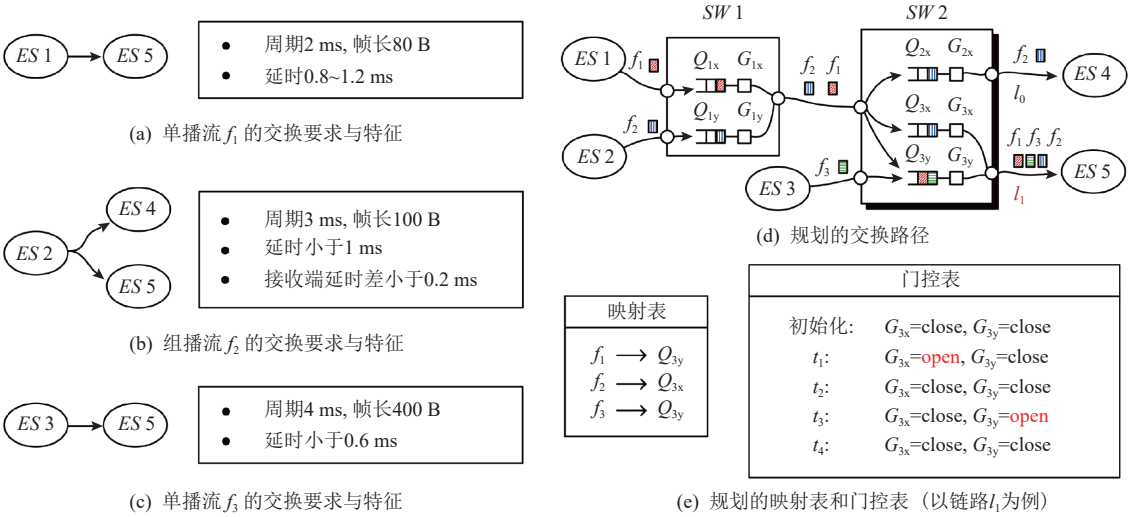


Fig. 2 Example of TSN traffic planning

图 2 TSN 流量规划示例

规划过程可以分为 3 个步骤.

步骤 1. 收集业务需求和场景特征. 业务需求包括流量特征和交换要求, 如图 2(a)(b)(c)所示. 以流量 f_1 为例, 基本的业务需求参数如表 1 所示. 场景特征包括网络拓扑结构、网络资源和设备能力等. 图 2(d)展示该规划示例的场景特征. 以从交换机 $SW2$ 到终端 $ES5$ 的链路 l_1 为例, 基本的场景特征参数如表 2 所示.

步骤 2. 设置约束. 约束是确保数据帧在业务需求和场景特征的限制下达成无冲突传输的一系列规则. 常见的约束包括端到端延迟约束、队列数目约束等. 端到端延迟约束是指数据帧从源端到目的端所经历的延迟不能超过所允许的最大端到端延迟. 队列数目约束是指对于网络中任意交换机的任意一个输出端口, 数据帧占用的队列数目不能超过该输出端口分配给 TS 流量的队列数目. 设置合理的约束是规划算法的工作之一.

Table 1 Parameters of Application Requirements (Taking f_1 as an Example)

表 1 业务需求参数 (以流量 f_1 为例)

参数	值	含义
$f_1.id$	1	流量 f_1 的 id 为 1
$f_1.v_{\text{talker}}$	$ES1$	流量 f_1 的源节点为节点 $ES1$
$f_1.v_{\text{listener}}$	$ES5$	流量 f_1 的目的节点为节点 $ES5$
$f_1.period$/ms	2	流量 f_1 的周期为 2 ms
$f_1.size$/B	80	流量 f_1 的帧长为 80 B
$f_1.delay$/ms	[0.8,1.2]	流量 f_1 的延时为 0.8~1.2 ms

Table 2 Parameters of Scenario Characteristics (Taking Link l_1 as an Example)

表 2 场景特征参数 (以链路 l_1 为例)

类型	参数	值	含义
拓扑结构	$l_1.id$	1	链路 l_1 的 id 为 1
	$l_1.v_{\text{src}}$	$SW2$	链路 l_1 的源节点为 $SW2$
	$l_1.v_{\text{dst}}$	$ES5$	链路 l_1 的目的节点为 $ES5$
网络资源	$l_1.bd$/Mbps	1 000	链路 l_1 的带宽为 1 000 Mbps
	$l_1.que$	2	链路 l_1 缓存 TS 流量的队列数为 2
设备延时参数	$l_1.proc$/μs	1	链路 l_1 的处理延时为 1 μs

步骤 3. 约束求解，生成时空资源分配方案. 约束求解是规划算法的主要工作. 约束求解是一个 NP 完全问题[14-19]，求解方式多种多样. 图 2(e)以映射表和门控表的方式表示交换机$SW2$到终端$ES5$的链路l_1的输出端口的时空资源分配方案. 根据映射表和门控表，3 条流量均能够在某段时间内独占物理介质，不会产生冲突.

在上述规划步骤中，步骤 3 的约束求解由 TSN 规划算法完成. 规划算法的主要功能是在拓扑结构、网络资源、设备能力和业务需求等多维约束下，为每个 TSN 交换机分配关键帧的无冲突发送时刻，生成时空资源分配方案. 流量规划问题的本质是在有限的网络带宽资源中合理安排 TS 流量，可以将其等价为一个装箱问题[14-19]. 装箱问题可以表述为：给定n个尺寸在 (0,1] 内的物品$a_1,a_2,\cdots,a_n$，目标是使用数量尽可能少的单位容量箱子装下所有物品，每个箱子中物品尺寸和都不超过 1. 装箱问题对应的决策问题是 NP 完全的：给定若干物品，判断特定数目的箱子是否能够装下. 流量规划问题中，TS 流量等价为物品，数据帧大小等价为物品尺寸，链路带宽等价为箱子大小. 规划算法需要判断在特定链路带宽下，数据帧是否都能够被无冲突地传输. 无冲突的传输即等价于物品被不重叠地装入箱子. 因此 TSN 流量规划被认为是一个 NP 完全问题.

学术界针对约束求解的方法展开了大量的研究工作. 根据求解方法的不同，规划算法可以分为 2 类：第 1 类是使用通用求解器求解约束的方法，常用的求解器包括 SMT[15,19,21-22]，ILP[16,20]，SMT 和 ILP 均为通用的约束求解工具. 第 2 类是使用专用搜索算法作为求解器求解约束的方法，专用的优化方法包括多种启发式算法[17,23-24]和深度学习算法[18]等.

1.3 TSN 规划器

TSN 规划器是一个集场景构建、规划求解、结果验证等功能于一体的工具软件，TSN 规划算法是 TSN 规划器的一个子功能模块. TSN 规划器在 TSN 生态中承担至关重要的功能.

由于 TSN 主要用于工业制造流水线、汽车、轨道交通工具和航空航天器内部网络，这些网络的共同特点是规模有限、封闭，且组网设备的能力参数、关键流量及确定性需求在流量规划设计阶段可知. 针对上述应用场景，TSN 标准定义了集中配置模型，对 TSN 网络进行管理[31].

图 3 展示集中式配置模型，它能够描述 TSN 生态的主要部分. TSN 规划器在集中式配置模型中发挥关键作用. 配置模型从上至下依次是集中式用户配置(centralized user configuration, CUC)实体、集中式网络配置(centralized network configuration, CNC)实体以及底层的 TSN 设备(终端和交换设备). 一次完整的 TSN 网络配置流程如下(某些步骤并不是严格串行的，为了方便说明，此处以串行的方式介绍)：① CUC 收集所有端设备的接口能力和业务需求，收集完成后根据业务需求为流量分配id，形成流量规范. ② CUC 将端设备接口能力和流量规范发送给 CNC 作为规划器的输入. ③ CNC 收集拓扑信息、交换设备的能力作为规划器的输入，接着 CNC 内的规划器计算时空资源分配方案，生成路由、调度或整形的配置. 研究的规划器目前聚焦于映射表和门控表的生成(如图 2(e)所示). ④ CNC 将交换设备需要配置的各种表项的信息分发给交换设备. ⑤ CNC 将所有的终端配置发送给 CUC，CUC 将配置分发到对应的端设备.

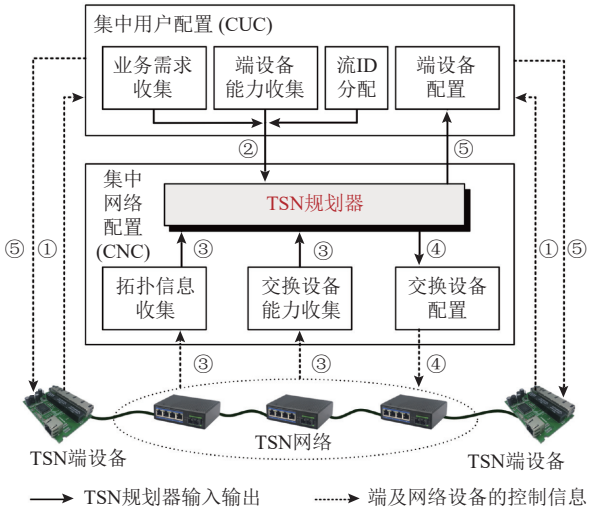


Fig. 3 Location of traffic planning in TSN ecosystem

图 3 流量规划器在 TSN 生态中的位置

可见，TSN 规划器是集中式配置模型的核心软件，在管理配置模型中承担最主要的功能，是 TSN 应用部署的重要工具.

研究的规划器专注于 TS 流量的离线规划. TS 流量是指周期性的数据流，其数据帧的产生间隔固定，并且流量的周期已知. 这种静态特性使得 TS 流量的传输可以被提前规划，这个规划过程通常在系统运行前完成，因此被称为离线规划. 在系统运行前，规划器会离线地规划 TS 流量，生成 GCL. GCL 会被下发到端设备和交换设备. 在系统运行时，端设备和交换设备根据 GCL 切换门控，确保 TS 流量的端到端确定性.

根据功能，规划器大致分为 4 个模块：规划算法、评估程序、验证程序和硬件相关的编译程序.

1)规划算法根据拓扑连接关系、终端和交换设备的接口能力以及流规范等场景需求为流量计算时空资源分配，是一个 NP 完全问题[14-19]；

2)验证程序验证时空资源分配结果的正确性；

3)评估程序评估时空资源分配结果的质量；

4)时空资源分配结果经过编译程序编译后会转换为交换机的路由、调度或整形的表项配置，这些配置与具体的交换机硬件平台紧密相关.

1.4 相关研究

目前，绝大部分研究工作关注的是 TSN 规划算法[13-14]，几乎没有工作关注 TSN 规划器. 据调研，Slate XNS[32] 是仅有的关注 TSN 规划器的研究工作，它支持网络管理人员使用图形化界面对拓扑界面添加或移除流量并生成调度计划. 但 Slate XNS 是一个闭源的商业软件[26]，仅集成内置的 2 个规划算法[15,21]，并且不支持算法开发人员在软件中集成自定义算法.

虽然目前规划算法的研究工作很丰富，但是并不存在一个通用的、满足所有应用场景需求的规划算法. 以 Slate XNS 集成的 2 个经典算法[15,21] 为例. 文献 [15] 的算法求得的解的端到端延时较小，但使用的 GCL 表项较多，仅适用于交换机存储资源丰富的场景. 文献 [21] 的算法能够用较少的 GCL 表项调度更多流量，但端到端延时相对较大，仅适用于延时要求较为宽松的场景.

由于单一规划算法难以适配多种应用场景，因此，集成单一规划算法的闭源规划器软件的应用场景将极其受限. 设计一个能够集成多种规划算法的规划器需要设计规划算法输入输出的接口规范，该规范应当描述 TSN 规划问题的基础抽象，能够兼容不同 TSN 规划算法的输入输出. 据调研，目前尚无研究算法输入输出接口规范的工作.

2 开源 TSN 规划器的研制动机与挑战

2.1 研制的动机

TSN 规划器一般由 TSN 设备产商提供，系统私有且封闭. 具体来说，现有的 TSN 规划器存在 3 个问题：

1)灵活性不足. 规划器仅集成单一规划算法，难以满足不同应用场景的差异化需求.

评价一个算法的指标是多维的，主要包括运行时间和解的质量. 算法的指标不同，其适用的场景就不同. 现有规划器仅集成单一规划算法，这导致其难以满足不同场景的差异化需求. 以 Slate XNS 为例，Slate XNS 仅集成 2 个算法[15,21]. 将文献 [15] 的算法称作帧算法，将文献 [21] 的算法称作窗口算法. 帧算法求得的解的端到端延时较小，但使用的 GCL 表项较多，仅适用于交换机存储资源丰富的场景. 窗口算法能够用较少的 GCL 表项调度更多流量，但端到端延时相对较大，仅适用于延时要求较为宽松的场景. 因此，集成单一算法的规划器使用场景有限，灵活性不足.

2)转换成本高. 规划器被具体的硬件设备锁定，转换成本是替换整套专用软硬件栈.

规划器最终生成的配置需要下发到具体的硬件设备，因此规划器生成的配置与硬件设备紧密相关. 不同产商的规划器与硬件设备之间协同工作的能力较差，规划器被具体的硬件设备锁定. 用户替换当前组网设备的成本将是替换整套专用软硬件栈.

3)创新迭代慢. 规划器代码不开源，学术界的最新成果难以快速集成到规划器为工业界所用.

规划器代码不开源，一方面，这使得学术界的最新研究成果难以快速集成到 TSN 规划器. 另一方面，大多数学术界提出的规划算法只能在各自的仿真平台上进行性能评估，无法在真实的硬件平台上验证效果. 而工业界的 TSN 芯片和设备产商又因为缺少规划器使得设备部署应用滞后. 总之，规划器代码不开源使得学术界与工业界均难以快速创新.

长期以来，系统封闭几乎是网络软件面临的共有的难题. 网络软件一般是由硬件供应商提供的专用软件. 随着网络系统越来越开放，开源软件技术成为突破上述难题的良好解决方案. 典型的案例包括开源软件定义网络控制器 OpenDaylight[27] 和开源网络操作系统 SONiC[28]. OpenDaylight 是一个开源的软件定义网络控制器，它为用户提供一套构件化的、可扩展的、支持多种协议的控制器框架，使用户能够快速地连接、配置和管理底层的网络设备.SONiC 是一个开源的网络操作系统，它为用户提供一个开放的、可编程的交换机操作系统，打破交换机硬件产商私有系统所造成的各种限制. 更多的开源网络软件包括 Open VSwitch[33]，Floodlight[34]，FD.io[35] 等，开源软件在网络领域的流行是一种明显的趋势.

受 SONiC 和 OpenDaylight 等开源项目的启发，将开源作为手段设计 TSN 规划器是一个良好的解决方案. 开源 TSN 规划器应当支持集成各种规划算法以满足场景的差异化需求，并且打破硬件厂商的锁定.

2.2 研制的挑战

设计开源 TSN 规划器面临以下 2 个挑战.

1)规划算法与规划工具难以解耦. 规划工具包括典型应用测试集、结果验证工具、结果评估工具和编译工具等. 规划算法与典型应用测试集之间的接口为算法的输入接口. 不同算法的输入接口不同, 这导致开发人员通常需要按照需求为不同算法构造特定的输入, 使得算法与典型应用测试集紧耦合. 另外, 规划算法与评估工具、验证工具及第三方编译工具之间的接口为算法输出接口. 不同算法的输出接口中的未知参数含义完全不同, 这导致规划算法与上述 3 种工具紧耦合.

2)规划器与交换硬件的实现难以解耦. 由于规划器最终生成的整型或调度配置需要下发到具体的硬件设备, 而编译工具通过规划算法紧密地耦合在规划器内, 这实际上导致了规划器与硬件设备难以解耦.

造成上述挑战的主要原因是规划算法的输入输出接口缺乏规范. 算法输入接口由描述业务需求和场景特征的参数组成. 算法输出接口由算法求解的未知参数组成.

不同算法的输入和输出接口各不相同. 就输入接口而言, 不同算法的输入接口具有共性但同时存在差异. 就输出接口而言, 不同算法的输出接口中的参数含义完全不同. 以帧算法[15]和窗口算法[21]为例说明上述观点, 2 个算法输入输出接口的详细参数详见附录 A 的表 A1.

在输入接口方面, 帧算法和窗口算法均包含业务需求和场景特征的基本参数(如表 1 和表 2 所示), 这是两者的共性. 2 个算法的差异体现在两者分别包含某些个性化参数. 例如, 帧算法还需要规划时隙(*macrotick*)这一参数, 规划时隙是数据帧发送时间的最小时间单位. 假设帧算法将规划时隙设为 1 μs, 那么数据帧的发送时间只能是 1 μs 的整数倍. 窗口算法则增加了 GCL 表项长度这一参数, 数据帧所使用的 GCL 表项不能超过 GCL 表项长度. 除此之外, 窗口算法还将流量的最坏端到端抖动作为输入接口的参数, 而帧算法则不关注这一指标, 它默认对数据帧进行 0 抖动的调度.

在输出接口方面, 帧算法和窗口算法完全不同. 帧算法输出每个流实例在第几个*macrotick*从哪个输出队列中将数据帧调度输出, 第几个*macrotick*由输出接口中的发送偏移$offset$描述, 哪个输出队列由输出接口中的*prio*描述. 窗口算法输出每条链路每次在哪 2 个时刻之间打开和关闭哪一个队列的门控, 以及本次门控打开的时间段内哪些流实例被调度输出. 一条链路上一系列打开和关闭门控的时刻由*phi*和*tau*这 2 个数组描述, 打开和关闭哪个队列的门控由*kappa*数组描述, 哪些流实例被调度输出则由*omega*描述, *omega*是上述数组的索引.

因此, 形成算法的输入输出接口规范, 并设计一个松耦合的规划器架构, 对开源 TSN 规划器设计具有重要意义.

3 松耦合构件化规划器架构 LOCAP

3.1 总体设计思想

本文提出一个松耦合的构件化的规划器架构 LOCAP(loose-coupled component-based architecture of planner), 如图 4 所示. LOCAP 提出“插座”和“构件库”2 个概念, 实现规划算法与规划工具的松耦合以及规划器软件与硬件实现的松耦合. “插座”与“构件库”的定义如下.

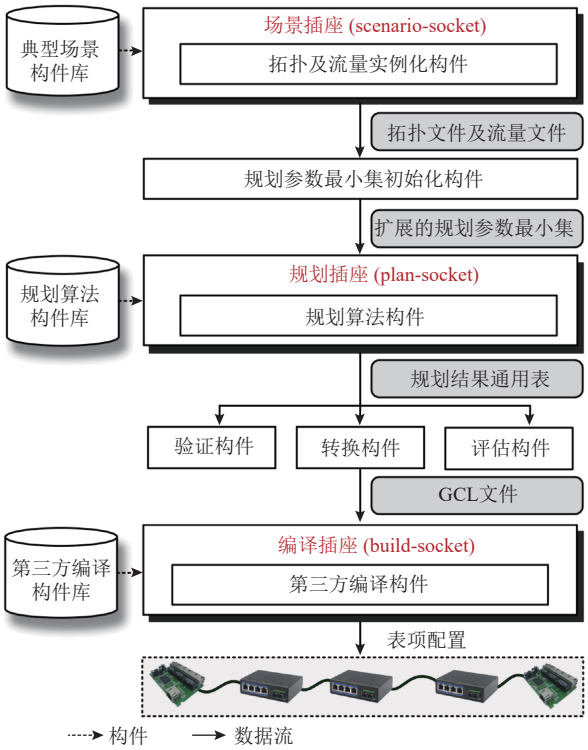


Fig. 4 Architecture of LOCAP

图 4 LOCAP 架构

插座(socket)定义面向特定功能的数据操作方法, 以及该方法的输入输出接口规范. 接口规范是指数据结构定义或文件格式.

构件是对插座定义的操作方法的具体实现. 符合插座所定义的输入输出接口的构件可以作为一个标准“插头”插入该插座. 构件库则是实现某一特定

功能的构件的集合. 构件库中的构件均遵循插座定义的输入输出接口, 区别只在于实现操作方法的具体技术不同.

为解决 2.2 节提到的第 1 个挑战, 即实现规划算法与规划工具的松耦合, LOCAP 提出规划插座(plan-socket). 规划插座定义 min-col 和 gen-tbl 这 2 个接口规范. 符合该接口规范的规划算法可以插入该插座.

min-col 是用于调度规划的基础参数, 是对网络资源和业务需求的高度抽象, 因此被用作规划算法的输入接口规范. gen-tbl 是时间触发通信中最本质的时间和空间资源分配的描述方式, 因此被用作规划算法与验证、转换和评估构件之间的接口.

为解决 2.2 节提到的第 2 个挑战, 即实现规划器软件与硬件实现的松耦合, LOCAP 提出编译插座(build-socket). 编译工具将规划结果编译为硬件可识别、可加载的配置, 不同的交换硬件有配套的编译工具. 因此实现规划器与编译工具的松耦合, 就实现了规划器与特定硬件实现的松耦合.

编译插座定义 GCL 文件为接口规范, 将 GCL 文件作为第三方编译构件的接口规范, 符合该接口规范的编译构件都可以插入该插座.GCL 文件是由 TSN 标准定义的 GCL 表项扩展而来的算法结果描述文件. 选用 GCL 文件作为接口规范的原因在于, 尽管不同产商 TSN 设备具体的硬件表项各不相同, 但 TSN 标准定义的 GCL 表项是这些具体硬件表项的共同抽象, 使用这一共同抽象作为编译工具的输入是合理的. 同时, 将 GCL 文件作为接口规范有利于促成不同 TSN 设备的互联互通. 因为使用不同 TSN 设备组网时, 网络管理人员可以将 GCL 文件作为所有编译工具的输入, 直接生成不同设备的配置, 省略了将 GCL 文件转换成不同编译工具的输入这一中间步骤.

除此之外, 为支持使用实际应用对算法性能进行评估, LOCAP 还提出场景插座(scenario-socket)及典型应用场景构件库. 场景插座定义拓扑及流量文件为接口规范, 典型应用场景构件库中不同场景的场景特征和业务需求都使用这 2 个文件进行描述. 架构图中各个插座、接口规范、库、构件的功能如表 3 所示.

由于设计开源规划器的关键在于提出规划算法的输入输出接口规范, 本节接下来的内容将对规划参数最小集和规划结果通用表 2 个接口规范进行详细描述. 输入输出接口规范的基本目标是兼容各种算法的输入输出, 除此之外输入输出接口规范还需要回答以下 4 个问题:

问题 1. 输入接口规范如何支持设备的各类延时参数的表示, 如何支持不同厂家的设备组网时链路层延时不一致的情况.

问题 2. 输入接口规范如何支持调度时间粒度不

Table 3 Functions of Sockets, Interfaces, Libraries, and Components in LOCAP Architecture

表 3 LOCAP 架构的插座、接口、库和构件的功能

类别	名称	功能
插座	1) 场景插座	定义面向典型应用场景的拓扑和流量实例化操作方法, 将拓扑及流量文件作为输出接口规范
	2) 规划插座	定义流量规划操作方法, 将规划参数最小集和规划结果通用表作为输入输出接口规范
	3) 编译插座	定义面向具体硬件平台的结果编译操作方法, 将规划结果通用表和 GCL 文件作为输入输出接口规范
接口规范	1) 拓扑及流量文件	拓扑文件描述拓扑结构和网络资源, 流量文件描述流量业务需求
	2) 扩展的规划参数最小集	用于调度规划的基础参数, 是对网络资源和业务需求的高度抽象
	3) 规划结果通用表	时间触发通信中最本质的时间和空间资源分配的描述方式
	4) GCL 文件	由 TSN 标准定义的 GCL 表扩展而来的规划结果描述文件
库	1) 典型场景构件库	保存实例化拓扑及流量的构件
	2) 规划算法构件库	保存规划算法构件
	3) 第三方编译构件库	保存第三方编译构件
构件	1) 拓扑及流量实例化构件	实例化指定规模的拓扑和流量, 输出拓扑及流量文件
	2) 规划参数最小集初始化构件	将拓扑文件和流量文件解析为规划参数最小集对应的数据结构
	3) 规划算法构件	以规划参数最小集为输入, 为流量计算无冲突的时空资源分配方案, 输出为规划结果通用表
	4) 验证构件	验证规划算法输出结果的正确性
	5) 转换构件	将规划结果通用表转换为 GCL 文件
	6) 评估构件	评估规划算法输出结果的质量
	7) 第三方编译构件	将 GCL 文件编译为与具体硬件设备实现相关的硬件配置

同的算法.

问题 3. 输出接口规范如何支持算法质量评估.

问题 4. 为何不直接采用 TSN 标准定义的 GCL 格式作为输出接口规范，输出接口规范与 TSN 标准定义的 GCL 是何种关系.

问题 1 和问题 2 将在 3.2 节讨论，问题 3 和问题 4 将在 3.3 节讨论. 针对上述问题，提出规划算法的输入规范 min-col 和规划算法的输出规范 gen-tbl.

3.2 规划参数最小集（min-col）

不同算法输入接口虽然具有共性但也存在差异，使得输入接口难以进行抽象. 共性体现在不同算法的参数存在较大交集，且参数的分类基本相同. 差异则体现在不同算法的某些具体参数有细微区别. 以帧算法和窗口算法输入接口为例说明上述观点.

帧算法和窗口算法输入接口的共性体现在它们的基本参数相同（如表 1 和表 2 所示）. 2 个算法的差异体现在两者分别包含某些个性化参数. 例如，帧算法还需要*macrotick*这一参数. 窗口算法则增加了 GCL 表项长度这一参数. 除此之外，窗口算法还将流量的最坏端到端抖动作为输入接口的参数，而帧算法则不关注这一指标，它默认对数据帧进行 0 抖动的调度.

算法输入接口存在差异，但同时又具有共性. 为了兼容不同算法的已知和未知参数，有 2 条可能的技术路线，如表 4 所示.

Table 4 Possible Roadmap of Input Interface Specification

表 4 输入接口规范可能的技术路线

评价	技术路线	
	参数最大集	参数最小集
优点	如果已知参数在最大集中，无需开发人员扩展	对网络资源和业务需求的高度抽象
缺点	不可能囊括所有未知参数	开发人员需要按照需求扩展最小集
采纳	×	√

注：“×”表示不采纳；“√”表示采纳.

第 1 种技术路线是合并主流规划算法输入接口的参数，提出规划参数最大集. 规划参数最大集保留输入参数中共性的部分，同时合并算法输入参数的差异. 对于已有算法而言，最大集是合理的. 但对于尚未提出的算法，如果算法的输入有新增的参数，那么开发人员仍需要对最大集进行扩展，那么最大集就成为一个伪命题. 另外，规划算法数量众多，如果囊括所有算法的输入参数，最大集将庞大而复杂.

第 2 种技术路线是提取不同算法输入接口的共性部分，提出规划参数最小集. 共性是不同算法对规划问题建模的相似的部分，这些相似的部分描述 TSN 流量规划问题的本质. 对于差异部分，开发人员可按照需求进行扩展. 要描述一个具体的流量规划算法的输入接口，只需要在规划参数最小集的基础上添加更丰富的属性（已知参数或未知参数），即可完成精确建模.

因此，针对算法的输入接口提出规划参数最小集 min-col.

定义 1. 规划参数最小集 min-col. min-col 是指用于调度规划的基础参数，是对拓扑结构、网络资源、设备能力和业务需求的高度抽象，其形式化定义为：

$$C=\{L,F,S\}, \tag{1}$$

其中C表示 min-col，无论算法的求解方式如何，算法关注的对象只有链路L、流量F和流实例S这 3 类.

L是链路的集合，$l_i\in L$（$i\in\{0,1,\cdots,N-1\}$，N是总的链路数目），描述拓扑结构、网络资源和设备能力. 链路l_i可以用式（2）的五元组表示.

$$l_i=\{id,v_{\text{src}},v_{\text{dst}},bd,que\}, \tag{2}$$

其中$i\in\{0,1,\cdots,N-1\}$，$v_{\text{src}},v_{\text{dst}}\in V$.$l_i$的属性分为身份属性和资源属性 2 类. 第 1 类是身份属性，即$l_i.id$，$l_i.v_{\text{src}}$，$l_i.v_{\text{dst}}$. 这里$l_i.id$指链路id，id一般是指l_i下标i的值.l_i是一条从$l_i.v_{\text{src}}$指向$l_i.v_{\text{dst}}$的链路.V是网络内交换节点和终端节点的集合.$l_i.v_{\text{src}}$指该链路的源节点，$l_i.v_{\text{dst}}$指该链路的目的节点. 身份属性唯一地标识一条链路. 第 2 类是资源属性，即$l_i.bd$和$l_i.que$.这里$l_i.bd$指链路带宽，$l_i.que$指链路分配给 TS 流量的队列数目.

F是 TS 流量的集合，$f_i\in F$（$i\in\{0,1,\cdots,M-1\}$，M是总的 TS 流量的数目）. 流量f_i可以用式（3）的六元组表示.

$$f_i=\{id,v_{\text{talker}},v_{\text{listener}},route,period,size\}, \tag{3}$$

其中f_i的属性分为 3 类，分别是身份属性、基本属性和特征属性. 第 1 类是身份属性，即$f_i.id$.这里$f_i.id$指流量id，id一般是f_i下标i的值. 第 2 类是基本属性，即流量的源节点$f_i.v_{\text{talker}}$、目的节点$f_i.v_{\text{listener}}$以及该流量的路由路径$f_i.route$. 路由路径是一系列链路的有序集合，$f_i.route=\{l_j,l_{j+1},\cdots,l_k\}$. 第 3 类是特征属性，即$f_i.period$和$f_i.size$.这里$f_i.period$指源节点产生流量的最小时间间隔，在 TSN 中，TS 流量均为周期性流量，因此$f_i.period$就是流量的周期. 所有流量周期的最小公倍数被称作调度周期，用HP表示. 流量f_i在HP内会重复出现$HP/f_i.period$次.$f_i.size$是指该流量每个周

期内的数据帧大小.

S是所有TS流量的流实例的集合，流实例用$s_{i,r_j,k}$表示.$s_{i,r_j,k}$是TS流量f_i在链路l_k上在调度周期内的第r_j个数据帧. 流实例$s_{i,r_j,k}$可以用式(4)的三元组表示：

$$s_{i,r_j,k}=\{stream_{id},instance_{id},link_{id}\}, \tag{4}$$

三元组的元素均为流实例的身份属性. $s_{i,r_j,k}.stream_{id}$指该流实例属于哪条流，即下标i对应的值. 其次$s_{i,r_j,k}.instance_{id}$指这是该流在调度周期内的第几个数据帧，其为下标$r_j$对应的值. 然后$s_{i,r_j,k}.link_{id}$是指这是哪条链路上的流实例，即下标$k$对应的值.

关系M定义流量、链路和数据帧序号与流实例的映射关系，$M:F,L,r_j\to S$，如式(5)所示.

$$M\left(f_i,l_k,r_j\right)=\begin{cases}s_{i,r_j,k},l_k\in f_i.route\wedge r_j\in\{0,1,\cdots,HP/f_i.period\},\\ \text{null},l_k\notin f_i.route\vee r_j\notin\{0,2,\cdots,HP/f_i.period\}.\end{cases} \tag{5}$$

针对问题1和问题2，无论是算法调度的时间粒度，还是链路层的延时，实际上都是与链路相关的参数. 算法在链路l_i原有五元组的基础上扩展时间粒度，例如帧算法的$macrotick$，即可按照不同时间粒度进行规划(问题1). 规划算法关心的延时参数大致包括时间同步精度、链路层延时等. 时间同步精度可以直接在链路l_i五元组的基础上进行扩展. 当底层网络采用不同厂家的设备时，测得相邻设备的链路层延时，将链路延时参数在链路l_i原有五元组的基础上进行扩展即可支持不同设备组网的情况(问题2).

3.3 规划结果通用表（gen-tbl）

不同算法输出接口的参数含义完全不同，使得输出接口难以进行抽象. 输出接口的规范有2个要求，一是算法无关，二是设备无关.

规划算法输出接口的参数是算法设置的未知参数，其参数含义各不相同. 即便规划结果相同，算法的输出也完全不同. 以帧算法和窗口算法为例，帧算法输出每个流实例在第几个$macrotick$从哪个输出队列中将报文调度输出，第几个$macrotick$由输出接口中的发送偏移$offset$描述，哪个输出队列由输出接口中的$prio$描述. 窗口算法输出每条链路每次在哪2个时刻之间打开和关闭哪一个队列的门控，以及本次门控打开的时间段内哪些流实例被调度输出. 一条链路上一系列打开和关闭门控的时刻由phi和tau这2个数组描述，打开和关闭哪个队列的门控由$kappa$数组描述，哪些流实例被调度输出则由$omega$描述，$omega$是上述数组的索引.

无论规划算法的输出如何，算法最终的求解目标均为“每个流实例何时从何队列调度输出”，这是由时间触发网络时分复用的基本原理决定的.TSN属于时间触发网络技术，TAS利用门控的“开”和“关”控制队列中的数据帧在指定的时刻调度输出. 因此，“流实例何时从何队列调度输出”是描述算法输出的一种通用的、普适的方式. 基于此基本抽象提出gen-tbl.

定义2. 规划结果通用表gen-tbl. gen-tbl是时间触发通信中最本质的时间空间资源分配描述方式. gen-tbl由规划算法求解得到，$solve(C)\to O$，其形式化定义如式(6)所示.

$$O=\left\{\left\{o_{i,r_j,k}\right\}\middle|\ \begin{array}{c}f_i\in F,\\ r_j\in\{0,1,\cdots,HP/f_i.period-1\},\\ l_k\in f_i.route\end{array}\right\}, \tag{6}$$

其中$o_{i,r_j,k}$是每个流实例的解. $o_{i,r_j,k}$的表达式如式(7)的六元组所示.

$$o_{i,r_j,k}=\{stream_{id},instance_{id},link_{id},t_{\text{start}},t_{\text{end}},q_{id}\}. \tag{7}$$

流实例的解由六元组表示.$stream_{id}$，$instance_{id}$，$link_{id}$唯一地标识一个流实例. t_{start}表示为该流实例分配的传输窗口的开始时刻，t_{end}表示为该流实例分配的传输窗口的结束时刻，q_{id}表示该流实例从何TS队列调度输出.TS队列是指用于缓存TS流量的数据帧的队列数目.

gen-tbl是时间触发通信中最本质的时间和空间资源的描述方式，任意一个规划算法的输出接口都可以转换成gen-tbl，gen-tbl也可以转化成任意规划算法的输出.

针对问题3和问题4，由于gen-tbl的表项包含每个流实例的时间和空间资源分配，因此可以从gen-tbl获取每条流的端到端延时、流量在特定周期在网络内的驻留时间、流量在特定周期在源端的驻留时间、流量到达接收端的抖动等指标.gen-tbl能够方便地支持算法质量评估(问题3).

虽然所有规划算法的结果均要转换成对门控列表GCL的配置，但是直接采用GCL作为输出接口的规范会导致算法质量无法直接进行对比(问题4). 因为GCL定义的是“门控何时切换至何状态”，GCL的每个表项由gateStates和timeInterval构成，gateStates指示输出队列尾部的门控状态，timeInterval则指示当前门控状态的持续时间. 首先，GCL的表项与流量并无直接的对应关系，针对每条流量评估算法质量时，还需要做额外的映射，即GCL表项与流量之间的映

射关系. 其次, timeInterval 定义门控状态持续的相对时间, 使用绝对时间更符合直观经验. 因此相较于 GCL, 直接采用 gen-tbl 作为输出接口规范更为方便. 值得一提的是, gen-tbl 能够简单直接地转换为 GCL, t_{start}和t_{end}之差即门控持续时间, q_{id}则对应门控状态. 转换工作在转换构件中完成.

4 基于 LOCAP 的 OpenPlanner 设计与实现

基于 LOCAP 实现开源 TSN 规划器 OpenPlanner. OpenPlanner 已在代码托管平台开源. 本节首先描述接口、插座、构件和构件库的设计, 接着以帧算法为例介绍基于 OpenPlanner 开发规划器的流程, 最后给出 OpenPlanner 的实现细节.

4.1 接口规范设计

算法输入输出的规范由 *MinCol* 类和 *GenTbl* 类这 2 个数据结构实现. *MinCol* 类由链路(*Link*)、流量(*Stream*)和流实例(*Instance*)这 3 类成员变量组成. *GenTbl* 类由流实例的解(*SolutionEachLink*)成员变量组成.*MinCol* 类和 *GenTbl* 类中的成员变量采用数组进行组织. 使用数组的原因是规划算法对 *MinCol* 类和 *GenTbl* 类的操作只有查找元素或查找并修改元素的操作, 几乎没有增加或删除元素的操作. 考虑到 *MinCol* 类和 *GenTbl* 类中的元素都包含身份属性, 同一类对象(如 *Link*)的身份属性一般是连续的非负自然数, 使用身份属性作为数组索引即可快速查找到对应的元素. 同时, 由于没有增加或删除元素的操作, 数组不便于增加或删除元素的缺点被完全屏蔽. 其他接口规范(拓扑文件、流量文件以及 GCL 文件)均采用 json 格式描述, 简单直观且可读性强.

4.2 Socket 设计

3 类插座 scenario-socket, plan-socket, build-socket 分别由虚类 *ScenarioSocket*, *PlanSocket*, *BuildSocket* 实现.

以 *PlanSocket* 虚类为例说明插座的设计. *PlanSocket* 虚类定义虚方法 *plan*(). *plan*() 将 *MinCol* 类的对象作为输入, 并返回 *GenTbl* 类的对象. 虚方法在主进程中被调用. 不同算法继承 *PlanSocket* 类并实现重写虚方法 *plan*(), 主进程动态地加载算法类, 并调用算法类实现的 *plan*() 方法, 实现不同算法插入插座的效果.

4.3 功能构件设计

OpenPlanner 目前仅能实现转换构件, 评估和验证构件的实现是 OpenPlanner 的下一步工作. 转换构件由 *SolutionTransformingComponent* 类实现. 在该类中, 定义了一个名为 *transform_gen_tbl_into_gcl*() 的方法, 用于将算法输出的 *GenTbl* 类结果转换为 GCL 文件. 具体来说, 该方法将流实例的解 $o_{i,r_j,k}$ 中的 t_{start} 和 t_{end} 转换为门控持续时间, 并将 q_{id} 转换为相应的门控状态.q_{id} 队列后端的门控状态设置为打开, 而其他队列后端的门控状态设置为关闭.

另外, 为了支持开发人员灵活地插入自定义功能, 在场景插座和规划参数最小集初始化构件内插入 2 个 HOOK 函数 .*ScenarioSocket* 类内部的 HOOK 函数允许开发人员定制拓扑和流量. 规划参数最小集初始化构件内部的 HOOK 函数允许开发人员基于 *MinCol* 类定制算法输入.

4.4 库设计

OpenPlanner 典型场景构件库中包含多种典型场景, 基础场景包括线型拓扑、环型拓扑和雪花型拓扑, 真实的应用场景包括汽车拓扑[5]、Miura 微型发射器拓扑[6]、Orion 载人航天器拓扑[7] 和列车拓扑[36]. 与基础拓扑相比, 真实应用的拓扑是各种基础拓扑的组合, 比基础拓扑更复杂. 基础场景支持按照自定义需求实例化为不同规模的拓扑. 基于上述拓扑能够按照自定义需求实例化具有不同特征的业务流量.

OpenPlanner 算法构件库目前包含 4 种典型算法, 分别为帧算法[15]、窗口算法[21]、ITP[17] 和 XZ-Plan[37]. 帧算法和窗口算法为第三方算法, ITP 和 XZ-Plan 为自研算法.OpenPlanner 第三方编译构件库目前包含 OpenTSN4.0 的编译构件 OpenTSNBuilder.

更详细的 OpenPlanner 设计详见开源仓库[38].

4.5 规划器开发流程

基于 OpenPlanner 开发规划器的流程最多包含 6 个步骤.

步骤 1. 根据规划算法的输入接口, 按需继承并扩展 *MinCol* 类中的链路类、流量类 *Stream* 和流实例类 *Instance*.

步骤 2. 按需实现 HOOK 函数, 包括场景插座和规划参数最小集初始化构件内的 2 个 HOOK 函数.

步骤 3. 若使用场景构件库以外的场景构件, 则继承 *ScenarioSocket* 类, 并实现其定义的操作方法 *construct_scenario*().

步骤 4. 若使用算法构件库以外的算法构件, 则继承 *PlanSocket* 类, 并实现其定义的操作方法 *plan*().

步骤 5. 若使用编译构件库以外的编译构件, 则继承 *BuildSocket* 类, 并实现其定义的操作方法 *build*().

步骤 6. 编写配置文件, 通过配置文件选取并加载场景构件、算法构件和编译构件, 并注册 HOOK 函数.

至此, 开发完成. 更详细的开发流程见 OpenPlanner 开源仓库中的开发指南.

帧算法[15] 是 TSN 流量规划的经典算法. 基于 OpenPlanner 开发帧算法的伪代码如算法 1 所示.

算法 1. 帧算法的伪代码.

①/*步骤 1: 扩展 *MinCol* 类*/
② class *FrameLink*(*Link*):
③ int *macrotick*;
④ class *FrameStream*(*Stream*):
⑤ int *latency_requirement*;
⑥ class *FrameInstance*(*Instance*):
⑦ Int *offset*; /*SMT 变量*/
⑧ Int *prio*; /*SMT 变量*/
⑨ int *period_in_macrotick*;
⑩ int *latency_in_macrotick*;
⑪ /*步骤 2: 实现钩子函数*/
⑫ function *min_col_reinit_hook*();
⑬ /*步骤 4: 继承 *PlanSocket* 类, 实现算法*/
⑭ class *FrameAlgo*(*PlanSocket*):
⑮ function *plan*(*col*):
⑯ *cons*=*gen_frame_constraints*(*col*);
⑰ *tbl*=*solve_frame_constraints*(*cons*);
⑱ return *tbl*.

步骤 1 是扩展 *MinCol* 类. 在链路方面, 帧算法引入 *macrotick* 的概念. *macrotick* 是等长的时间片, 是规划流实例的最小时间粒度. 每一条链路都有自己的 *macrotick*, 不同链路的 *macrotick* 不同. 因此帧算法继承 *Link* 类实现 *FrameLink*(行②), 在 *FrameLink* 类中增加 *macrotick* 这一成员变量(行③). 在流量方面, 帧算法使用最坏的端到端延时来描述流量的服务质量需求, 因此它继承 *Stream* 类实现 *FrameStream* 类(行④), 在 *FrameStream* 类中添加 *latency_requirement* 这一成员变量(行⑤). 在流实例方面, 帧算法将“流实例的调度时间”和“流实例从哪一个队列调度输出”作为需要求解的未知量, 因此帧算法继承 *Instance* 类实现 *FrameInstance* 子类(行⑥), 在 *FrameInstance* 子类中添加 2 个成员变量: *offset* 和 *prio*(行⑦⑧), 这 2 个成员变量是 SMT 中的整型未知变量, 并不是普通的整型变量, 它们的类型为 Int 而非 int. SMT 求解器是一个通用的约束求解器, 帧算法使用 SMT 求解器求解. *offset* 的含义是流实例被调度输出的时间, 这个时间以 *macrotick* 为单位. *prio* 的含义是流实例从哪一个队列被调度输出. 另外, 帧算法将 *macrotick* 作为最小的时间粒度, 它将流实例的周期和延迟都转化成以 *macrotick* 为单位的数值, 因此帧算法还在 *FrameInstance* 类中添加 *period_in_macrotick* 和 *latency_in_macrotick* 这 2 个成员变量(行⑨⑩).

步骤 2 是编写 HOOK 函数. 帧算法有定制算法输入的需求, 因此编写规划参数最小集初始化构件内的 HOOK 函数, 即根据 *FrameLink* 中的 *macrotick* 与 *FrameStream* 中的 *period* 和 *latency_requirement* 初始化 *FrameInstance* 中的 *period_in_macrotick* 和 *latency_in_macrotick*(行⑪).

无需使用场景构件库之外的构件, 因此跳过步骤 3.

步骤 4 是实现算法. 继承 *PlanSocket* 类, 重写其中的虚方法 *plan*(), 在虚方法中实现流量求解的过程(行⑬~⑱).*plan*() 的输入是扩展后的基于 *MinCol* 类的数据结构(行⑮), 输出是基于 *GenTbl* 类的数据结构(行⑱). 帧算法 *plan*() 方法的实现请参考附录 B 第 1 节.

无需使用编译构件库之外的构件, 因此跳过步骤 5.

步骤 6 是编写配置文件. 通过配置文件从构件库中选取并注册功能构件以及注册 HOOK 函数 *min_col_reinit_hook*().

4.6 实 现

OpenPlanner 使用 Python 语言(版本为 3.8.1)编程, 基础功能代码约 1 080 行.

OpenPlanner 中嵌入 4 种典型算法, 算法及其源代码行数分别为帧算法[15](411 行)、窗口算法[21](422 行)、ITP[17](783 行)和 XZ-Plan[37](700 行). 帧算法和窗口算法由 TTech 公司等单位研发, ITP 和 XZ-Plan 属于自研算法. 其中帧算法和窗口算法使用 SMT 求解器搜索 1 个可行解. OpenPlanner 采用的 SMT 求解器是 z3py(版本为 4.8.15).z3py 是由微软研究院开发的高性能通用求解器, 在约束求解、软件/硬件验证与测试、混合系统分析等多个领域都得到了广泛的应用. ITP 和 XZ-Plan 使用专用搜索算法进行求解. ITP 使用禁忌搜索算法(tabu search), 在领域相关知识的指导下搜索可行解. XZ-Plan 则使用了有指导策略的贪心算法. 由于流量规划算法是规划器中的一个重要模块, 因此在附录 B 中描述了 4 个算法的原理、流程以及它们在 OpenPlanner 中的实现.

图 5 展示算法代码占基础功能代码的比例. OpenPlanner 所构建的统一算法输入输出: *MinCol* 类和 *GenTbl* 类, 使不同算法能够复用同一套数据结构, 研究人员只需要关注算法核心部分的开发, 这使得

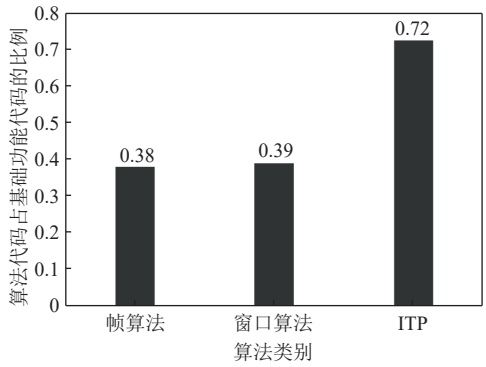


Fig. 5 Proportion of code

图 5 代码占比

算法的代码开发的工作量大大降低. OpenPlanner 能够有效提升算法研究人员的开发效率.

5 实验评估

实验评估主要回答第 1 个挑战提出的问题：OpenPlanner 是否支持规划算法与规划工具的松耦合，即 OpenPlanner 是否支持集成不同算法. 关于第 2 个挑战提出的问题，即 OpenPlanner 是否支持规划器与硬件实现的松耦合，在第 6 节进行验证.

所有实验均在配有 Intel core i7-8550U 64 位 1.80 GHz CPU, 16 GB RAM 的机器上运行. 测试案例使用的拓扑包括基本拓扑和真实场景中的拓扑，基本拓扑包括环型拓扑、线型拓扑和雪花型拓扑，真实场景下的拓扑包括汽车拓扑[5]、Orion 载人飞船拓扑[7]以及列车拓扑[36]. 6 个拓扑的图片在附录 C 中展示. 流量的周期为 5 ms 或 10 ms，路径长度不超过 7 跳. 数据帧大小均为 1518 B. 帧算法的时间粒度(参数 *macrotick*)设为 1 μs. 窗口算法的门控列表长度(参数 *gcl*)设为 4. ITP 的时间粒度(参数 *slot_cycle*)设为 125 μs，TS 队列深度设为 10，概率系数设为 0.7. 概率系数的作用可参考附录 B 第 1 节.

为证明 OpenPlanner 支持集成不同规划算法，将算法作为研究案例，在 OpenPlanner 上实现 4 个算法，4 个算法的代码均可通过 OpenPlanner 开源仓库访问. 为方便描述，将文献 [15] 的算法称作帧算法，将文献 [21] 的算法称作窗口算法，将文献 [17] 的算法称作 ITP，将文献 [37] 的算法称作 XZ-Plan. 对比帧算法、窗口算法和 ITP 这 3 个算法的解的质量和运行时间. 解的质量评估包括端到端延时评估、网络内驻留时间评估等. 3 种算法的综合评价如表 5 所示.

Table 5 Algorithms Integrated in OpenPlanner and Their Characteristics

表 5 OpenPlanner 集成的算法及其特征

算法名称	运行时间	解的质量	适用场景
帧算法[15]	较长	1) 流量的端到端延时小 2) 数据帧在网络内和源端的驻留时间均较短 3) 能够实现 0 抖动 4) GCL 表项占用多	适用于对延时和抖动要求十分严苛的场景，例如：运动控制
窗口算法[21]	长	1) 流量端到端延时较大 2) 数据帧在网络内和源端的驻留时间均较长 3) 有界抖动 4) GCL 表项占用少	适用于对延时和抖动要求相对宽松的场景，例如：工业控制
ITP[17]	较短	1) 流量的端到端延时中等 2) 流量在网络内的驻留时间短，在源端的驻留时间长 3) 有界抖动 4) GCL 表项占用少	适用于网络资源受限但终端资源较为丰富的场景

1)端到端延时评估. 流量的端到端延时是指从流量在源端产生直至被目的端接收这一过程的延时. 本实验假设流量均在周期的起始时刻产生. 某些对实时性要求严苛的应用希望该延时能够尽可能地小. 统计流量的平均端到端延时，实验结果如图 6 所示. 帧算法的端到端延时在环型、线型和雪花型 3 种简单拓扑并且流量周期单一(周期为 5 ms)的情况下基本保持在 200 μs 以下，但在汽车、Orion 和列车 3 种复杂拓扑下并且流量周期多样(周期为 5 ms 或 10 ms)的情况下，帧算法的端到端延时急剧增加. 窗口算法的趋势与帧算法基本相同，但是平均端到端延时的数值大于帧算法. ITP 的端到端延时无论是在简单拓扑下还是在复杂拓扑下没有太大变化. 其原因在于，ITP 中流量的端到端延时仅与路由跳数和时间槽大小(参数 *slot_cycle*)有关. 假设流量跳数为 h，某条流量的最大端到端延时为 $(h+1)\times slot_cycle$，最小端到端延时为 $(h-1)\times slot_cycle$. 不同拓扑下 ITP 设置的 *slot_cycle* 均相同，因此流量的端到端延时实际上仅与跳数有关. 无论在何种拓扑下，流量的跳数均在 7 跳以内. 因此，在不同拓扑下，ITP 计算的流量端到端延时相差不大.

2)网络内驻留时间评估. 流量在网络内的驻留

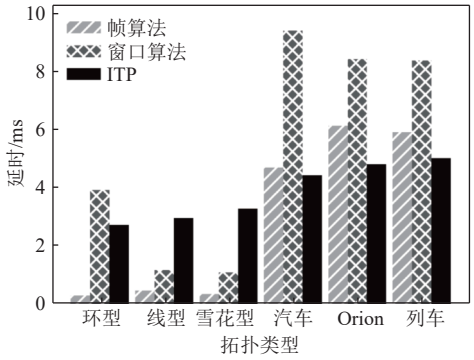


Fig. 6 Average end-to-end delay

图 6 平均端到端延时

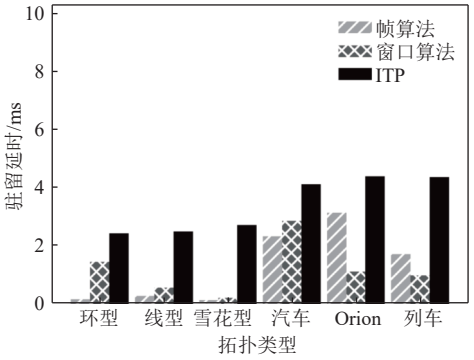


Fig. 8 Residence time at the source

图 8 源端的驻留延时

时间是指源端发出流量直至目的端接收到该流量这一过程的延时. 在网络内的驻留时间越长, 对交换机内存的压力越大. 在某些交换资源受限的应用场景下, 网络内的驻留延时应当尽可能地小. 统计流量在网络内每一跳的平均驻留延时, 实验结果如图 7 所示. ITP 在网络内的驻留时间最短, 并且驻留时间并不随着拓扑结构和流量特征的变化有剧烈变化. 这是因为 ITP 规定流量在每一跳仅停留 1 个时间槽. 而帧算法和窗口算法在网络内的驻留时间都比较长, 对交换机的内存资源要求相对较高.

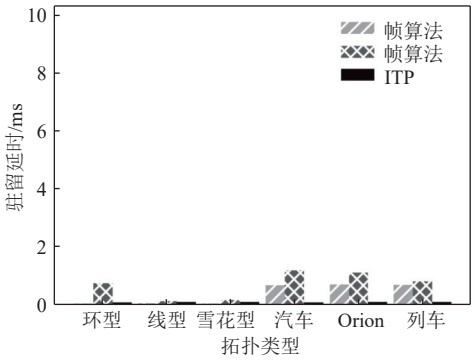


Fig. 7 Average in-network residence time per hop

图 7 网络内每跳的平均驻留延时

3)源端驻留时间评估. 流量在源端的驻留时间是指流量从产生到源端发送这一过程的延时. 在源端的驻留时间越长, 端系统就需要分配更多存储资源为流量设置缓冲区. 如果端系统在网卡上缓存这些尚未发送的流量, 对网卡的存储资源要求就会较高. 统计流量在源端的平均驻留延时, 实验结果如图 8 所示. 与网络内驻留延时的结果相反, 帧算法和窗口算法在源端的驻留时间相对较短, 而 ITP 无论在何种拓扑下, 在源端的驻留时间都超过 2 ms. 由于 ITP 导致 TS 流量在源端的驻留时间较长, 因此为了适配 ITP 算法, 源端需要配备足够多的缓冲区以缓存驻留在端系统中的流量. 因此, ITP 对源端的存储资源有较高的需求. 然而, ITP 导致流量在网络内的驻留时间相对较短, 因此对于交换机的资源需求相对较低.

4)目的端抖动评估. 流量在目的端的抖动是指目的端接收到该流量不同周期的数据帧的时间间隔之间的变化范围. 某些应用要求实现 0 抖动. 统计流量在接收端的平均抖动, 结果如图 9 所示. 帧算法可以达到 0 抖动的效果, 窗口算法和 ITP 的抖动则分别在 15~35 μs 和 12 μs 左右波动. 帧算法达到 0 抖动的效果是通过限制任意时刻一个输出队列内只能存在 1 个关键数据帧达到的. 窗口算法和 ITP 则允许 1 个输出队列内同时存在多个关键数据帧, 由于关键数据帧在队列内的顺序并不完全确定, 输出队列又以先进先出的方式传输数据帧, 因此关键数据帧会存在一定程度有界的抖动.

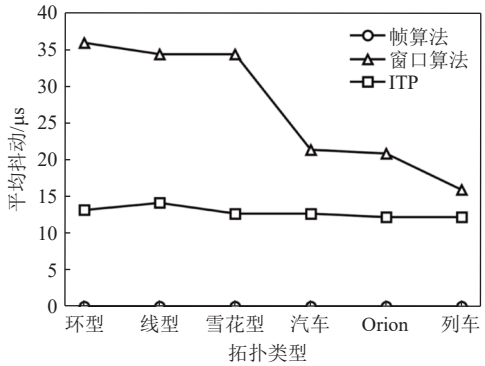


Fig. 9 Average jitter at the destination

图 9 目的端的平均抖动

5)GCL 表项数目评估. 本实验评估算法产生的解在交换机输出端口需要使用的 GCL 表项的数量. 统计流量在网络内所有输出端口中占用的最长的 GCL 表项数目, 结果如图 10 所示. 无论在何种拓扑下, 窗口算法的解最多只需要 9 条 GCL 表项, 因为窗口算法将 GCL 表项数目作为算法输入, 流量所使用

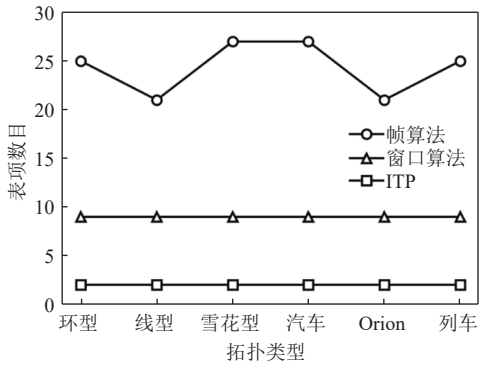


Fig. 10 Maximum number of used GCL entries

图 10 最多使用的 GCL 表项数目

的表项数目不能超过该数目. ITP 的解使用的 GCL 表项数目则始终为 2, 这与 ITP 底层的调度机制有关. 帧算法需要的 GCL 表项数目最多.

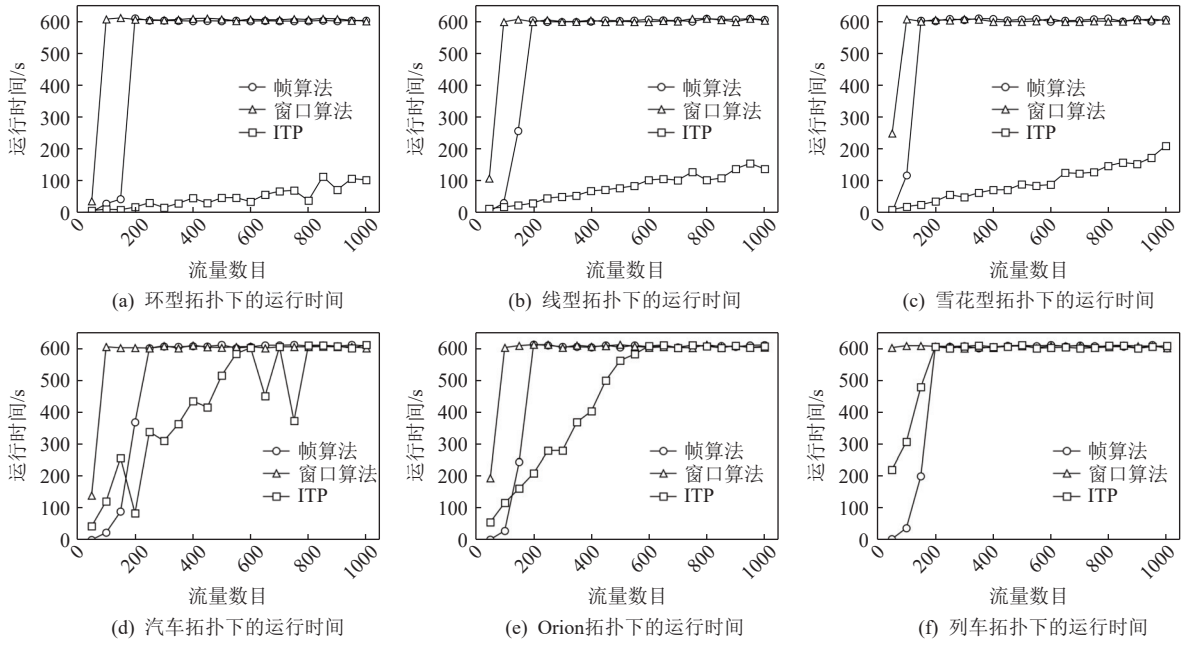
6)运行时间. 运行时间是指算法求解所花费的时间. 在环型、线型、雪花型、汽车、Orion 载人飞船和列车 6 个拓扑下, 流量数目从 50 逐渐增加到 1 000, 对比 3 个算法的运行时间. 将 3 个算法的超时时间设为 10 min, 如果超过 10 min 没有求出解则终止程序.3 个算法在 6 个拓扑上的运行时间如图 11 所示.

无论在何种拓扑下, 随着流量数目增多及问题规模变大, 3 种算法的运行时间均有增长的趋势. 除了超时的情况, 算法均求出可行解. 其中窗口算法的求解速度最慢, 运行时间在 3 种算法中最长, ITP 的求解速度最快, 运行时间在 3 种算法中最短. 在汽车、Orion 载人飞船和列车这 3 种真实的复杂拓扑下, 帧算法的运行时间在流量数目达到 200~250 时就已经超过 600 s.窗口算法的运行时间在流量数目超过 100 时已超过 600 s. 在列车拓扑下, 窗口算法求解 50 条流量的时间已经接近 600 s. 无论在何种拓扑下, ITP 的运行时间则远远小于帧算法和窗口算法. 在 Orion 载人飞船拓扑中, 流量数目达到 550 时, 其运行时间才接近 600 s. 这是由于帧算法和窗口算法采用通用求解器 SMT 求解, 随着问题规模增长, 其运行时间会爆炸式增长. 而 ITP 使用启发式方法求解, 在搜索解的过程中有领域相关知识的指导, 求解更有指向性, 因此求解速度更快.

值得注意的是, 在列车拓扑下, 当流量数目超过 200 时, ITP 的运行时间就会超过 600 s, 与帧算法相当. 在列车拓扑下, ITP 未能充分发挥其求解更具指向性的优势. 原因可能在于, 在相同的流量数目下, 与其他拓扑相比, 列车拓扑的网络资源更为紧缺, 如附录 C 的图 C1(f)所示. 其中, 8 个交换机构成线型的骨干网. 每个骨干交换机连接着对应每节列车车厢的子网, 子网由 3 个交换机构成环型拓扑. 骨干交换机的网络资源较为紧缺. 因此, 问题的解空间变小. 在这种情况下, 即便有领域下相关知识的指导, ITP 仍然需要花费更多时间搜索解, 因此运行时间变长.

6 OpenPlanner 的应用

在 OpenPlanner 中实现的多种 TSN 规划算法[15,17,21,37]



(a) 环型拓扑下的运行时间 (b) 线型拓扑下的运行时间 (c) 雪花型拓扑下的运行时间

(d) 汽车拓扑下的运行时间 (e) Orion拓扑下的运行时间 (f) 列车拓扑下的运行时间

Fig. 11 Runtime of algorithms under different topologies

图 11 算法在不同拓扑下的运行时间

已经在由 OpenTSN 开源网络(4.0 版)[39-40]构成的系统中得到应用,应用系统包括卫星、无人车和火炮等.

还在 OpenPlanner 中实现了基于宏时隙的 TTE 规划算法 XZ-Plan[37],该算法还可以实现 TTE 网络的规划,已经在湖南华芯通网络科技有限公司的 TTE 网络环境中得到应用.

上述应用部署案例表明,OpenPlanner 开源规划器具备支持多平台的能力,实现规划器与交换硬件的松耦合.

7 总结和下一步工作

现有 TSN 规划器存在灵活性不足、转换成本高和创新迭代慢 3 个问题. 设计开源 TSN 规划器面临算法与规划工具难以解耦以及规划器与硬件平台难以解耦 2 个挑战. 为此提出一种松耦合的构件化开源规划器架构 LOCAP,并设计规划算法的输入输出接口规范 min-col 和 gen-tbl.基于 LOCAP 实现 OpenPlanner 开源规划器.OpenPlanner 集成 3 个经典的 TSN 流量规划算法,第三方基于 OpenPlanner 实现自研规划算法.OpenPlanner 已在 3 个硬件平台得到应用部署. 这说明 OpenPlanner 支持算法与规划工具松耦合以及规划器与硬件平台松耦合.

OpenPlanner 的下一步工作将从 3 个方面展开:

1)验证构件的设计. 验证构件的功能是验证算法结果是否满足基础约束,验证构件对于使用启发式方法的规划算法具有重要意义. OpenPlanner 计划在验证构件中提供 2 种验证方式,即基于 SMT 的结果验证和状态递推的结果验证. 基于 SMT 的结果验证只对 TSN 中最基本的传输约束进行验证;状态递推的验证则以规划算法的最小时间粒度为单位,推演整个网络在每个时间粒度的传输状态以进行冲突检测. 实现验证构件是 OpenPlanner 的下一步工作.

2)对 AVB(audio and video bridging)流量的支持. 目前 OpenPlanner 集成的 4 个规划算法均针对 TS 流量规划时间和空间资源的分配,不考虑 AVB 流量的空间资源分配. AVB 流量通常为高带宽的流量,对端到端延时同样具有实时性的要求. 例如,车载自动驾驶系统中高清摄像头产生的流量就属于 AVB 流量. 如何在保证 TS 流量严格实时性的情况下,合理地为 AVB 流量分配空间资源/带宽资源,同样是一个重要的问题. OpenPlanner 目前讨论的输入输出接口主要针对 TS 流量规划算法的输入输出,这与其他类型算法(例如 AVB 流量的规划算法)的输入输出接口规范并不矛盾. 当讨论范围扩展到其他算法时,只需要对这一类算法的输入输出进行抽象,就能够使对应的算法集成到规划器. 对 AVB 流量规划算法的输入输出规范进行抽象是 OpenPlanner 的下一步工作.

3)输入输出接口规范的迭代. 由于 OpenPlanner 源码开放,可以引入广泛的用户评估,未来随着新算法的提出和新功能的加入(比如集成 AVB 流量的规划算法),输入输出接口能够不断迭代,以适配新功能,进一步完善 TSN 生态.

致谢:感谢国防科技大学计算机学院吴茂文、彭锦涛、黄容、刘雅兰工程师在 OpenPlanner 到 OpenTSN4.0 网络硬件适配过程中给予的帮助. 感谢北京交通大学李宗辉副教授、天津大学滕立平博士生、同济大学王子通博士生、中船 716 所冯振乾博士和湖南华芯通网络科技有限公司唐路博士对本文提供的帮助.

作者贡献声明:姜旭艳负责论文撰写和代码实现;全巍负责研究点讨论与论文修改;付文文负责实验设计;张小亮负责部分实验实现;孙志刚提出指导意见并修改论文.

参 考 文 献

[1] IEEE Standards Association. IEEE 802.1Q−2022: IEEE Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks[S]. Los Alamitos, CA: IEEE Computer Society, 2022

[2] Messenger J L. Time-sensitive networking: An introduction[J]. IEEE Communications Standards Magazine, 2018, 2(2): 29−33

[3] Nasrallah A, Thyagaturu A, Alharbi Z, et al. Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research[J]. IEEE Communications Surveys & Tutorials, 2019, 21(1): 88−145

[4] Huang Tao, Wang Shuo, Huang Yudong, et al. Survey of the deterministic network[J]. Journal on Communications, 2019, 40(6): 160−176 (in Chinese)
(黄韬, 汪硕, 黄玉栋, 等. 确定性网络研究综述[J]. 通信学报, 2019, 40(6): 160−176)

[5] Patti G, Bello L L, Leonardi L. Deadline-aware online scheduling of TSN flows for automotive applications[J]. IEEE Transactions on Industrial Informatics, 2023, 19(4): 5774−5784

[6] Sanchez-Garrido J, Aparicio B, Ramirez J G, et al. Implementation of a time-sensitive networking (TSN) Ethernet bus for microlaunchers [J]. IEEE Transactions on Aerospace and Electronic Systems, 2021, 57(5): 2743−2758

[7] Obermaisser R. Time-Triggered Communication[M]. Boca Raton, FL: CRC Press, 2012: 121−152

[8] Csaba S, Markosz M, Miklos M. Design aspects of low-latency

services with time-sensitive networking[J]. IEEE Communications Standards Magazine, 2018, 2(2): 48−54

[9] IEEE TSN Group. Use cases IEC/IEEE 60802[EB/OL]. [2023-09-13]https://grouper.ieee.org/groups/802/1/files/public/docs2018/60802-industrial-use-cases-0918-v13.pdf

[10] Quan Wei, Fu Wenwen, Sun Zhigang, et al. HX-DS09: A customized low power time sensitive networking chip for high-end equipment[J]. Journal of Computer Research and Development, 2021, 58(6): 1242−1245 (in Chinese)
(全巍, 付文文, 孙志刚, 等. 枫林一号: 一款面向高端装备定制的低功耗时间敏感网络芯片[J]. 计算机研究与发展, 2021, 58(6): 1242−1245)

[11] IEEE Standards Association. IEEE Std 802.1AS−2020: IEEE Standard for Local and Metropolitan Area Networks — Timing and Synchronization for Time-Sensitive Applications[S]. Los Alamitos, CA: IEEE Computer Society, 2020

[12] IEEE Standards Association. IEEE Std 802.1Qbv-2015: IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic[S]. Los Alamitos, CA: IEEE Computer Society, 2016

[13] Zhang Tong, Feng Jiaqi, Ma Yanying, et al. Survey on traffic scheduling in time-sensitive networking[J]. Journal of Computer Research and Development, 2022, 59(4): 747−764 (in Chinese)
(张彤, 冯佳琦, 马延滢, 等. 时间敏感网络流量调度综述[J]. 计算机研究与发展, 2022, 59(4): 747−764)

[14] Li Zonghui, Yang Siqi, Yu Jinghai, et al. State-of-the-art survey on deterministic transmission technologies in time-sensitive networking[J]. Journal of Software, 2022, 33(11): 4334−4355 (in Chinese)
(李宗辉, 杨思琪, 喻敬海, 等. 时间敏感网络中确定性传输技术综述[J]. 软件学报, 2022, 33(11): 4334−4355)

[15] Craciunas S S, Oliver R S, Chmelík M, et al. Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks[C] // Proc of the 24th Int Conf on Real-Time Networks and Systems. New York: ACM, 2016: 183−192

[16] Dürr F, Nayak N G. No-wait packet scheduling for IEEE time-sensitive networks (TSN)[C] // Proc of the 24th Int Conf on Real-Time Networks and Systems. New York: ACM, 2016: 203−212

[17] Yan Jinli, Quan Wei, Jiang Xuyan, et al. Injection time planning: Making CQF practical in time-sensitive networking[C] // Proc of IEEE Int Conf on Computer Communications. Piscataway, NJ: IEEE, 2020: 616−625

[18] He Xiaowu, Zhuge Xiangwen, Dang Fan, et al. DeepScheduler: Enabling flow-aware scheduling in time-sensitive networking[C/OL] // Proc of IEEE Int Conf on Computer Communications. Piscataway, NJ: IEEE, 2023 [2024-08-19]. https://ieeexplore.ieee.org/document/10228875

[19] Steiner W. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks[C] // Proc of the 31st IEEE Real-Time Systems Symp. Piscataway, NJ: IEEE, 2010: 375−384

[20] Smirnovt F , Glaß M , Reimann F , et al. Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks[C/OL]// Proc of the 54th Annual Design Automation Conf. New York: ACM, 2017 [2024-08-19]. https://ieeexplore.ieee.org/document/8060332

[21] Oliver R S, Craciunas S S, Steiner W. IEEE 802.1Qbv gate control list synthesis using array theory encoding[C] // Proc of the 24th IEEE Real-Time and Embedded Technology and Applications Symp. Piscataway, NJ: IEEE, 2018: 13−24

[22] Wilfried S, Craciunas S S, Serna O R. Traffic planning for time-sensitive communication[J]. IEEE Communications Standards Magazine, 2018, 2(2): 42−47

[23] Pahlevan M, Obermaisser R. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks[C] // Proc of the 23rd IEEE Int Conf on Emerging Technologies and Factory Automation (ETFA). Piscataway, NJ: IEEE, 2018: 337−344

[24] Quan Wei, Yan Jinli, Jiang Xuyan, et al. On-line traffic scheduling optimization in IEEE 802.1Qch based time-sensitive networks[C] // Proc of IEEE 22nd Int Conf on High Performance Computing and Communications & IEEE 18th Int Conf on Smart City & IEEE 6th Int Conf on Data Science and Systems. Piscataway, NJ: IEEE, 2020: 369–376

[25] Fu Wenwen, Quan Wei, Yan Jinli, et al. Fenglin-I: An open-source time-sensitive networking chip enabling agile customization[J]. IEEE Trans on Computers, 2023, 72(1): 140−153

[26] TTTech Industrial. Slate XNS[EB/OL]. [2023-09-06]. https://www.tttech-industrial.com/products/slate/slate-xns/

[27] The Linux Foundation Projects. OpenDaylight platform overview [EB/OL]. [2023-09-06]. https://www.opendaylight.org/about/platform-overview

[28] The Linux Foundation Projects. SONiC – Software for open networking in the cloud[EB/OL]. [2023-09-06]. https://sonicfoundation.dev/

[29] OpenTSN. OpenPlanner1.0[EB/OL]. [2024-07-07]. https://gitee.com/opentsn/open-planner

[30] OpenTSN. OpenPlanner2.0[EB/OL]. [2024-07-07]. https://gitee.com/xyjiang_1216/open-planner2.0

[31] IEEE Standards Association. IEEE Std 802.1Qcc-2018: IEEE Standard for Local and Metropolitan Area Networks − Bridges and Bridged Networks - Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements[S]. Los Alamitos, CA: IEEE Computer Society, 2018

[32] Craciunas S S, Oliver R S, Steiner W. Demo abstract: Slate XNS——An online management tool for deterministic TSN networks [C] //Proc of the 24th IEEE Real-Time and Embedded Technology and Applications Symp. Piscataway, NJ: IEEE, 2018: 103−104

[33] Open VSwitch. Production quality, multilayer open virtual switch[EB/OL]. [2023-09-06]. http://www.openvswitch.org/

[34] Project Floodlight. Floodlight OpenFlow controller (OSS)[EB/OL]. [2023-09-06]. https://github.com/floodlight/floodlight

[35] FD. io. The world's secure networking data plane[EB/OL]. [2023-09-07]. https://fd.io

[36] Li Zonghui, Wan Hai, Pang Zaiyu, et al. An enhanced reconfiguration for deterministic transmission in time-triggered networks[J]. IEEE/ACM Transactions on Networking, 2019, 27(3): 1124−1137

[37] Hunan Huaxintong Network Technology Co., Ltd.. Open-Source TTE planning software (XZ-Plan) white paper[EB/OL]. [2023-09-06]. http://www.c2comm.cn/resource_centre/XZ-Plan.pdf (in Chinese)

(湖南华芯通网络科技有限公司. 开源 TTE 规划软件(XZ-Plan)技术白皮书[EB/OL]. [2023-09-06]. http://www.c2comm.cn/resource_centre/XZ-Plan.pdf)

[38] OpenTSN.OpenTSNbuilder[EB/OL].[2024-07-07].https://gitee.com/xyjiang_1216/OpenTSN_planner_builder

[39] OpenTSN. OpenTSN4.0 [EB/OL]. [2023-09-13]. https://gitee.com/opentsn/open-tsn4.0

[40] Quan Wei, Fu Wenwen, Yan Jinli, et al. OpenTSN: An open-source project for time-sensitive networking system development[J]. CCF Transactions on Networking, 2020, 3: 51−65



Jiang Xuyan, born in 1998. PhD candidate. Her main research interests include time-sensitive networking and FPGA design.

姜旭艳, 1998 年生. 博士研究生. 主要研究方向为时间敏感网络、FPGA 设计.



Quan Wei, born in 1987. PhD, associate professor. His main research interests include time-sensitive networking, software-defined networking and FPGA design.

全　巍, 1987 年生. 博士, 副研究员. 主要研究方向为时间敏感网络、软件定义网络、FPGA 设计.



Fu Wenwen, born in 1994. PhD, assistant professor. His main research interests include programmable network and time-sensitive networking.

付文文, 1994 年生. 博士, 助理研究员. 主要研究方向为可编程网络、时间敏感网络.



Zhang Xiaoliang, born in 1991. Master, lecturer. His main research interests include software-defined networking and time-sensitive networking.

张小亮, 1991 年生. 硕士, 讲师. 主要研究方向为软件定义网络、时间敏感网络.



Sun Zhigang, born in 1974. PhD, professor. His main research interests include software-defined networking, time-sensitive networking, network architecture, and FPGA design and network security.

孙志刚, 1974 年生. 博士, 研究员. 主要研究方向为软件定义网络、时间敏感网络、网络体系结构、FPGA 设计以及网络安全.

附录 A.

表 A1 描述帧算法[15]和窗口算法[21]的输入输出参数及参数含义. 表 A1 以图 2 中的流量 f_1 及链路 l_1 为例, 对算法输入和输出进行描述. 其中算法输入仅展示帧算法与窗口算法不同的参数.

在算法输入方面, 除了基本参数外, 帧算法还将规划时隙(*macrotick*)这一参数作为输入, 规划时隙是数据帧发送时刻的最小时间单位. 如表 A1 所示, 帧算法将规划时隙设为 1 μs, 那么数据帧的发送时间只能是 1 μs 的整数倍.

窗口算法增加了 GCL 表项长度(*gcl*)和流量的最坏端到端抖动(*jitter*)作为算法输入. 如表 A1 所示, 链路 l_1 的最长 GCL 表项长度为 256, 窗口算法使用的 GCL 表项不能超过 256 项. 同时, 窗口算法还关注流量的最坏端到端抖动, f_1 的端到端延时要求在 0.8~1.2 ms, 因此其最坏端到端抖动不能超过 400 μs. 值得注意的是, 帧算法天然地将所有流量的抖动限制为 0, 因此无需将最坏端到端抖动这一参数作为输入.

在算法输出方面, 帧算法输出每个流实例在第几个*macrotick*从哪个输出队列中将数据帧调度输出, 第几个*macrotick*由输出接口中的发送偏移*offset*描述, 哪个输出队列由输出接口中的*prio*描述. 如表 A1 所示, 流量 f_1 在第 300 个*macrotick*从队列 Q_{3y} 被调度输出.

窗口算法输出每条链路每次在哪 2 个时刻之间打开和关闭哪一个队列的门控, 以及本次门控打开的时间段内哪些流实例被调度输出. 一条链路上一系列打开和关闭门控的时刻由*phi*和*tau*这 2 个数组描述, 打开和关闭哪个队列的门控由*kappa*数组描述, 哪些流实例被调度输出则由*omega*描述, *omega*是上述数组的索引. 如表 A1 所示, 流量 f_1 在 3 个数组内的索引为 1, 这表示 f_1 在 300~304 μs 从队列 Q_{3y} 被调度输出.

Table A1 Parts of Input Parameters and Output Parameters of the Frame and the Window Algorithms

表 A1 帧算法和窗口算法的输入参数（部分）及输出参数

算法		帧算法[15]		窗口算法[21]	
		参数	含义	参数	含义
算法输入	场景特征	$l_1.macrotick=1$	链路 l_1 的最小规划粒度为 1 μs	$l_1.gcl=256$	链路 l_1 的 GCL 表项长度为 256
	业务需求			$f_1.jitter=400$	流量 f_1 允许的最坏端到端抖动为 400 μs
算法输出		$f_1^{l_1}.offset=300$	流量 f_1 在链路 l_1 上开始被调度的时刻为第 300 个*macrotick*	$l_1.phi=[200,300]$	链路 l_1 打开传输窗口的时刻分别为第 200 μs 和第 300 μs
		$f_1^{l_1}.prio=Q_{3y}$	流量 f_1 在链路 l_1 上从 TS 队列 Q_{3y} 被调度输出	$l_1.tau=[200.64,304]$	链路 l_1 关闭传输窗口的时刻分别为第 200.64 μs 和第 304 μs
				$l_1.kappa=[Q_{3x},Q_{3y}]$	链路 l_1 在上述 2 个传输窗口内分别从 TS 队列 Q_{3x} 和 Q_{3y} 调度报文输出
				$f_1^{l_1}.omega=1$	流量 f_1 在链路 l_1 的*phi*，*tau*，*kappa*数组内对应的索引为 1

附录 B.

由于流量规划算法是规划器中的一个重要模块, 因此在本附录中描述了 4 个算法的原理、流程以及它们在 OpenPlanner 中的实现.

帧算法和窗口算法所使用的求解工具均为可满足性模理论 SMT. 为介绍帧算法和窗口算法的原理, 首先对 SMT 进行介绍. SMT 是一种形式化技术, 被用于检验针对某一特定背景理论的一阶逻辑公式(first-order logic formula)的可满足性或不可满足性. 如果公式可满足, 则给出一组使该公式成立的解. SMT 要求将被检验的一阶逻辑公式用合取范式表达.

1 帧算法的原理、流程和实现

帧算法所基于的背景理论是线性整数算数(linear integer arithmetic) $\mathcal{LA}(\mathbb{Z})$, 它将所要求解的变量定义为整型变量, 用基于线性整数算数的逻辑表达式表达规划问题的约束条件.

帧算法求解的变量可分为 2 类. 第 1 类是发送时间, 即流实例在哪一个 *macrotick* 被调度输出. 第 2 类是发送队列, 即流实例使用哪一个 TS 队列. 2 类变量均为整型变量. 其中, *macrotick* 是帧算法的规划粒度. 帧算法将连续的时间划分成等长的时间间隔, 一个时间间隔被称为一个 *macrotick*. 这样, 连续的时间被

划分成离散的时刻，以降低规划的复杂性.

帧算法需要求解出所有流实例的这2类变量. 帧算法为上述变量定义相应的约束，以确保生成的GCL能够保证TS流的有界延时和零抖动. 帧算法定义5类约束，其中除了4类时间触发调度的常见约束外，还包括一类TAS特定的约束.TAS特定的约束被称作帧隔离约束. 帧隔离约束要求任意2个经过同一链路l_k的流实例$s_{i,r_a,k}$和$s_{j,r_b,k}$的发送时间和发送队列只能是以下2种情况之一. 第1种情况，流实例$s_{i,r_a,k}$离开该链路后，流实例$s_{j,r_b,k}$才能从上游链路发出，或是相反，即流实例$s_{j,r_b,k}$离开该链路后，$s_{i,r_a,k}$才能从上游链路发出. 第2种情况，2个流实例进入不同的TS队列. 帧隔离约束能够保证任意时刻一个输出队列内至多只有1个流实例，这使得流实例的传输是0抖动的.

定义上述约束后，帧算法将约束输入到SMT，使用SMT检验是否有解. 如果有解，则给出一组使所有约束成立的解，即所有流实例的发送时间和发送队列.

OpenPlanner中开发帧算法的工作主要集中在扩展*MinCol*类和实现*plan*()方法2部分，如算法1所示. 4.5节已经介绍了帧算法对*MinCol*类的扩展，在此只对*plan*()方法的实现进行介绍(行⑮~⑱). *plan*()由2个关键函数组成. 第1个函数为构建约束的函数*gen_frame_constraints*(). 该函数以扩展后的*MinCol*类的对象作为输入，遵循z3py求解器的语法表达窗口算法的约束，并以列表的形式返回所有约束(行⑯). 第2个函数为求解函数*solve_frame_constraints*(). 该函数以约束列表为输入，将约束列表中的所有约束构成一个合取范式输入到z3py求解器. 具体的求解过程由求解器完成. 如果存在一个解使得所有约束都成立，求解器会输出该解. 函数*solve_frame_constraints*()解析求解器返回的解，并以*GenTbl*类的对象的形式输出(行⑱).

2 窗口算法的原理、流程和实现

窗口算法所基于的背景理论是数组一阶理论(first-order theory of integer-indexed arrays)$\mathcal{T}_A^{\{\mathbb{Z}\}}$，它将所要求解的变量定义为整型数组，用基于一阶数组理论的语法和运算符表达规划问题的约束条件.

窗口算法求解的变量可分为3类. 第1类是每条链路的传输时间窗口打开和关闭的时刻，窗口开闭时刻按序在窗口打开时刻数组和窗口关闭时刻数组中排列. 第2类是每条链路每个传输时间窗口所对应的处于开门状态的TS队列号. 队列号按照与窗口一一对应的关系在TS队列数组中排列. 第3类是流实例在传输窗口数组和TS队列数组中对应的索引值，即流实例到传输时间窗口的映射.

窗口算法需要求解每条链路的窗口开闭时刻数组与TS队列数组，以及每个流实例在上述3个数组内对应的索引值. 窗口算法为上述变量定义约束，以确保生成的GCL能够保证TS流的有界延时和低抖动. 除时间触发调度的基本约束之外，窗口算法还定义了流隔离约束. 与帧算法的帧隔离约束有所不同，流隔离约束要求任意2个经过同一链路l_k的流实例$s_{i,r_a,k}$和$s_{j,r_b,k}$只能是以下3种情况之一. 第1种情况，流实例$s_{i,r_a,k}$离开该链路后，$s_{j,r_b,k}$才能从上游链路发出，或是相反. 第2种情况，2个流实例进入不同的TS队列. 第3种情况，2个流实例位于同一个传输时间窗口，即它们在窗口开闭时刻数组和TS队列数组内的索引值相同. 另外，窗口算法可以通过限制窗口开闭时刻数组的长度，使得求出的解能够满足TSN交换机的GCL长度限制.

类似于帧算法，窗口算法将约束输入SMT求解器，由SMT求解器判断是否有解，若有，则输出一组使所有约束成立的解.

在OpenPlanner中开发窗口算法的工作主要集中在2个部分：其一是基于*MinCol*类扩展窗口算法的输入；其二是继承并实现*PlanSocket*类的虚方法*plan*()在其中实现求解的功能. 算法的伪代码如算法B1所示.

算法 B1. 窗口算法的伪代码.

```
①/*扩展 MinCol 类*/
② class WindowLink(Link):
③     int gcl;
④     Array phi_array;
⑤     Array tau_array;
⑥     Array kappa_array;
⑦ class WindowStream(Stream):
⑧     int latency_requirement;
⑨     int jitter_requirement;
⑩ class WindowInstance(Instance):
⑪     Int omega;
⑫ /*继承 PlanSocket 类，实现算法*/
⑬ class WindowAlgo(PlanSocket):
⑭     function plan(col):
⑮         cons=gen_win_constraints(col);
⑯         tbl=solve_win_constraints(cons);
⑰         return tbl.
```

首先需要根据窗口算法的输入扩展*MinCol*类(行①~⑪). 在链路方面，窗口算法引入*gcl*这一参数. *gcl*是该链路所允许的门控列表长度. 窗口算法继承

*Link*类实现*WindowLink*(行②)，在*WindowLink*类中增加*gcl*这一成员变量(行③)．同时，窗口打开时刻数组、窗口关闭时刻数组以及TS队列数组也应当作为成员变量加入*WindowLink*(行④~⑥)．3个数组的类型是z3py中的数组型未知变量，用Array表示．在流量方面，窗口算法使用最坏的端到端延时和最坏的端到端抖动描述TS流的服务质量需求，因此它继承*Stream*类实现*WindowStream*(行⑦)，并在*WindowStream*类中添加*latency_requirement*和*jitter_requirement*这2个成员变量(行⑧⑨)．在流实例方面，窗口算法将流实例在窗口开闭时刻数组和TS队列数组内所对应的索引值作为求解变量之一，因此窗口算法继承*Instance*类实现*WindowInstance*(行⑩)，在其中添加*omega*这一成员变量，*omega*的类型是z3py中的整型未知变量(行⑪)，用Int而非int表示．

接着继承并实现*PlanSocket*类的虚方法*plan*()，在*plan*()中实现算法的主体(行⑫~⑰)．*plan*()由2个关键函数组成．第1个函数为构建约束的函数*gen_win_constraints*()．该函数以扩展后的*MinCol*类的对象作为输入，遵循z3py求解器的语法表达窗口算法的约束，并以列表的形式返回所有约束(行⑮)．第2个函数为求解函数*solve_win_constraint*()．该函数以约束列表为输入，将约束列表中的所有约束构成一个合取范式输入到求解器．具体的求解过程由求解器完成．如果存在一个解使得所有约束都成立，求解器会输出该解．函数*solve_win_constraints*()解析求解器返回的解，并以*GenTbl*类的对象的形式输出(行⑰)．

3 ITP的原理、流程和实现

与帧算法和窗口算法不同，ITP假设交换机的流量整形机制为循环队列转发(cyclic queueing and forwarding, CQF)模型．CQF是TAS的一种特殊配置，它将交换机输出端口最高优先级的2个队列设置为TS队列，2个队列以乒乓的方式进行切换，交替地传输2个队列中的数据帧[1]．使用CQF整形流量时，数据帧在网络内的传输由它进入网络的时刻决定．因此，ITP规划的对象是数据帧从源端注入网络的时刻．

ITP采用禁忌搜索方法求解规划问题．ITP的目标是最大化规划成功率．它首先使用贪心算法产生一个初始解，从该初始解开始，利用交换模式生成候选解，不断地探索邻域解．交换模式是指从规划成功的流量集合中剔除一部分流量，释放它们所占用的资源，接着从规划失败的流量集合中选取流量进行规划，直到没有任何一条流量可以规划成功．

从规划成功的流量集合移除哪些流量，以及从规划失败的集合优先选取哪些流量重新规划将影响候选解的质量．ITP基于对CQF的观察，提出多级流量排序的策略．多级流量排序是指根据流量的周期、交换路径长度等特征，优先选取更容易被规划成功的流．多级流量排序作为领域相关知识指导ITP搜索解空间．

为防止陷入局部最优解，ITP设置概率系数，随机选择使用多级流量排序策略或随机策略搜索候选解．同时ITP设置禁忌列表，列表中保存历史最优解的候选解产生方式，避免陷入局部最优解．当搜索时间达到阈值，或迭代次数达到阈值时，则停止搜索并返回当前最优解．

在OpenPlanner中开发ITP的工作主要集中在2个部分，其一是基于*MinCol*类扩展ITP的输入．其二是继承并实现*PlanSocket*类的虚方法*plan*()．算法的伪代码如算法B2所示．

算法B2. ITP算法的伪代码．

```
① /*扩展 MinCol 类*/
② class ITPLink(Link):
③     int queue_size;
④ class ITPStream(Stream):
⑤     int latency_requirement;
⑥     int pkt_num;
⑦ /*继承 PlanSocket 类，实现算法*/
⑧ class ITPAlgo(PlanSocket):
⑨     function plan(col):
⑩         init, cur=random_result(col);
⑪         while iteration<max_iter do
⑫             cans=sched(cur, col, prob);
⑬             best, cur, repeat=tabu_update(cans);
⑭             if repeat≥max_repeat then
⑮                 return trans_gen_tbl(best);
⑯             end if
⑰         end while
⑱         return trans_gen_tbl(best).
```

首先根据ITP的输入扩展*MinCol*类(行①~⑥)．在链路方面，ITP关注TS队列的长度，引入*queue_size*这一参数．ITP继承*Link*类实现*ITPLink*类(行②)，并在其中添加成员变量*queue_size*(行③)．在流量方面，ITP将TS流所允许的最大端到端延时作为输入，同时允许一条TS流在一个周期内连续地发送多个报文．因此ITP继承*Stream*类实现*ITPStream*类(行④)，并在其中增加*latency_requirement*和*pkt_num*这2个成员变量(行⑤⑥)．

接着继承并实现*PlanSocket*类的虚方法*plan*()

(行⑧~⑱). 首先随机生成一个初始解(行⑩). 接着基于初始解生成候选解. 在搜索候选解时，以一定的概率系数 *prob* 随机选择多级流量排序策略或随机策略(行⑫). 产生候选解的集合后，按照需要更新历史最优解、当前解和更新禁忌列表(行⑬). 当迭代次数超过设定值(行⑪~⑰)，或最优解的重复出现的次数超过设定值时(行⑭~⑯)，算法终止，并以 *GenTbl* 类的对象的形式返回历史最优解(行⑮⑱).

4 XZ-Plan 的原理、流程和实现

XZ-Plan 采用贪心算法求解规划问题. XZ-Plan 将时间划分为粗粒度的等长时隙，该时隙被称作宏时隙. 宏时隙是最小的规划粒度. XZ-Plan 规划流量的核心思想是，冲突最多的链路是阻碍流量规划成功的瓶颈，因此优先解决冲突最多的链路的时空资源分配问题对于提高规划成功率非常关键. 为此 XZ-Plan 定义冲突度. 冲突度是指一条链路上发送窗口存在重叠的流实例的个数. 冲突度越大说明链路越拥挤，流量越难以规划.

在 OpenPlanner 中开发 XZ-Plan 的工作主要集中在 2 个部分：其一是基于 MinCol 扩展 XZ-Plan 的输入；其二是继承并实现 *PlanSocket* 类的虚方法 *plan*(). 算法的伪代码如算法 B3 所示.

算法 B3. XZ-Plan 算法的伪代码.

① /*扩展 *MinCol* 类*/
② class *HugeSlotStream*(*Stream*):
③ int *latency_requirement*;
④ /*继承 *PlanSocket* 类，实现算法*/
⑤ class *HugeSlotAlgo*(*PlanSocket*):
⑥ function *plan*(*col*):
⑦ *win_set*=*random_window*(*col*);
⑧ *conf_tbl*=*compute_conflict*(*win_set*);
⑨ while *exist conflict* do
⑩ *link*=*max_conflict*(*conf_tbl*);
⑪ *win_set*=*reschedule*(*link*,*win_set*);
⑫ end while
⑬ return *trans_gen_tbl*(*win_set*).

首先根据 ITP 的输入扩展 *MinCol* 类(行①~③). 在流量方面，XZ-Plan 关注 TS 流的最大端到端延时需求. 因此 XZ-Plan 继承 *Stream* 类定义 *HugeSlotStream*(行②)，并在其中添加 *latency_requirement* 这一成员变量(行③).

接着继承并实现 *PlanSocket* 类的虚方法 *plan*()(行④~⑬). 算法首先根据输入随机生成源端的发送窗口和目的端的接收窗口，以缩小算法的搜索空间. 随机得到的窗口包含若干连续的宏时隙(行⑦). 接着根据发送和接收的窗口划定交换路径上每一跳链路的窗口范围. 根据上述窗口范围计算每条链路的冲突度(行⑧). 找到冲突度最大的链路(行⑩)，为该链路上的流实例分配发送的时隙(行⑪). 重复冲突度计算和分配的步骤，直至所有链路都无冲突(行⑨~⑫). 最后算法将求得的解以 *GenTbl* 类的对象的形式输出(行⑬).

附录 C.

6 个被测拓扑如图 C1 所示.

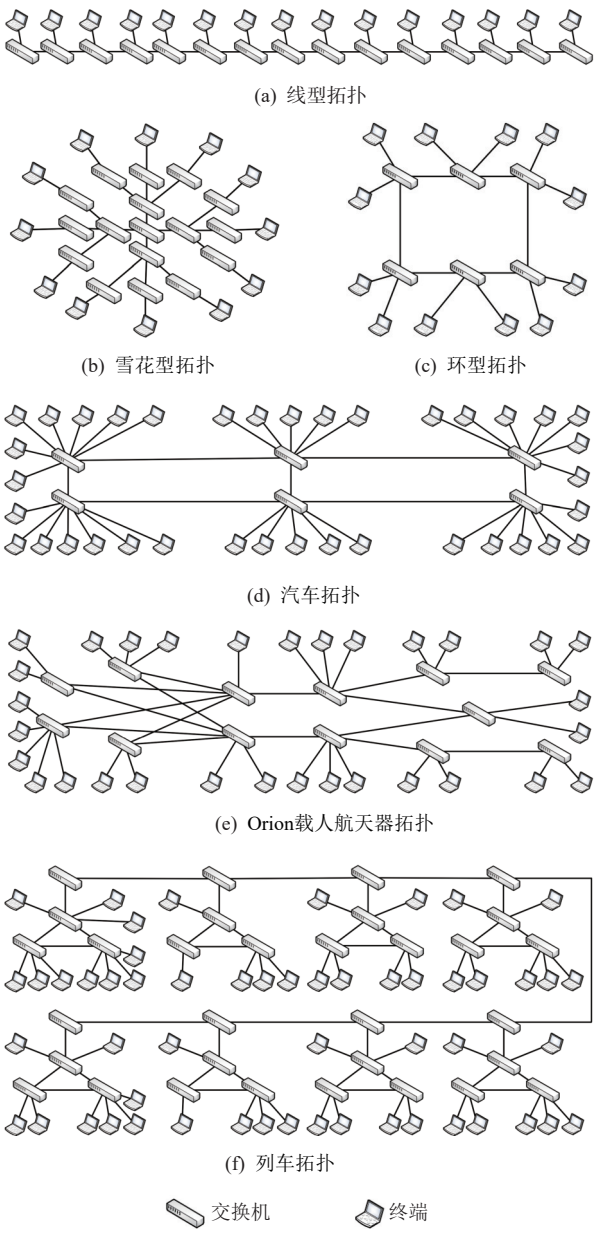


Fig. C1 The tested topology

图 C1 被测拓扑

环型拓扑包含 6 个交换机和 12 个终端, 每个交换机上 2 个终端. 线型拓扑包含 15 个交换机和 15 个终端, 每个交换机上 1 个终端. 雪花型拓扑由 3 层构成, 每层分别包含 1, 4, 12 个交换机, 最后一层的交换机上每个交换机上 1 个终端, 共 17 个交换机和 12 个终端.

汽车拓扑[5] 包含 6 个交换机和 37 个终端. Orion 载人飞船拓扑[7] 包含 13 个交换机和 31 个终端. 列车拓扑[36] 包含 32 个交换机和 54 个终端.