计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2024,33(6):133−142 [doi: 10.15888/j.cnki.csa.009548]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

时间敏感网络业务流动态调度场景中的批量重配置算法①

陈庭威, 王智立

(北京邮电大学 网络与交换技术全国重点实验室, 北京 100876)
通信作者: 王智立, E-mail: zlwang@bupt.edu.cn

摘　要: 工业自动化领域广泛使用时间敏感网络技术. 该领域业务流的调度方式主要包含静态调度和动态调度. 静态调度一次计算所有业务流, 可以最大程度节省链路和时间资源, 但是计算时间长, 无法灵活处理新增业务流. 动态调度以增量的形式计算新增业务流, 计算时间短, 但是资源分配不够合理, 会产生时隙碎片. 全局流重配置机制可以定期对网络中所有业务流进行重新规划, 来优化链路和时间资源的分配, 但该机制只适用于拥有较少业务流的小型网络, 业务流数量的增多会引起计算时间的急剧增长, 影响后续到来的业务流. 本文在现有动态调度算法的基础上, 设计了批量重配置算法. 该算法给出了新的评价指标——网络吞吐率, 并在满足动态调度秒级响应时间的情况下, 定期重配置网络中的部分业务流, 优化网络资源配置. 此外, 算法给出了重配置业务流的选取标准, 并优化了流的路径选择标准和传输开始时间计算方式. 本文针对原算法和增加了批量重配置机制的改进算法进行了仿真实验, 实验结果表明, 改进算法可以在拥有数千条业务流的大型网络运行, 并在网络吞吐率和调度成功的流数量方面有 16.5% 和 5.5% 的提升, 同时保证了算法的秒级计算时间.

关键词: 工业自动化; 时间敏感网络; 业务流调度; 重配置; 路径选择

引用格式: 陈庭威,王智立.时间敏感网络业务流动态调度场景中的批量重配置算法.计算机系统应用,2024,33(6):133–142. http://www.c-s-a.org.cn/1003-3254/9548.html

Batch Reconfiguration Algorithm in Dynamic Flow Scheduling Scenario of Time-sensitive Network

CHEN Ting-Wei, WANG Zhi-Li

(State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Time-sensitive network technologies are widely used in industrial automation. The flow scheduling methods in this field mainly include static and dynamic scheduling. Static scheduling computes all flows at a time, which can save link and time resources to the greatest extent but has the disadvantages of long computation time and lack of flexibility to handle new flows. Dynamic scheduling computes new flows incrementally, with short computation time but insufficient resource allocation, resulting in time slot fragmentation. The global flow reconfiguration mechanism can periodically replan all flows in the network to optimize the allocation of link and time resources. However, this mechanism only applies to small networks with fewer flows, and an increase in the flow number can cause a sharp increase in computation time, affecting subsequent flows. This study designs a batch reconfiguration algorithm based on the existing dynamic scheduling algorithm. This algorithm provides a new evaluation indicator: network throughput. It can regularly reconfigure some flows to optimize network resource allocation while meeting the dynamic scheduling second-level response time requirement. In addition, the algorithm gives reconfigured flow selection standards and optimizes flow path

① 收稿时间: 2023-12-27; 修改时间: 2024-01-29; 采用时间: 2024-02-26; csa 在线出版时间: 2024-05-07
CNKI 网络首发时间: 2024-05-10

selection standards and transmission start time calculation. This study conducts simulation experiments on the original and improved algorithm with the batch reconfiguration mechanism. The experimental results show that the improved algorithm can run in large networks with thousands of flows and have a 16.5% and 5.5% improvement in network throughput and the number of successfully scheduled flows while ensuring the second-level calculation time of the algorithm.

Key words: industrial automation; time-sensitive network; flow scheduling; reconfiguration; path selection

1 引言

物联网、5G、工业互联网等新一代信息通信技术的发展, 加快了传统行业转型升级的步伐. 例如在智能制造中, 实时控制、边缘计算、数字孪生等应用场景对网络低延时、低抖动和高可靠性的要求愈发严格. 传统以太网已经不能满足越来越多的数据和广泛分布的网络需求, 时间敏感网络 (TSN) 技术应运而生. 时间敏感网络以传统以太网为基础, 通过时钟同步、数据调度、网络配置等机制, 提供确定性数据传输能力.

TSN 是 IEEE 802.1 任务组开发的一套数据链路层协议, 用于构建更可靠、低延迟、低抖动的以太网. TSN 技术标准起源于音视频行业, 用于满足广播、直播等网络应用需求. 并且应用到了汽车和工业领域. 在汽车领域中, 主要应用在高级驾驶辅助系统方面. 在工业领域, 则主要应用在智能制造、工业互联网领域[1].

本文主要针对工业 4.0 中的工业自动化领域. 该领域主要研究静态调度和动态调度两种调度方式. 静态调度[2–4]常用在智能工厂中, 集中式网络配置 CNC[5]拥有全局视角, 能够获得将要调度的业务流的全局信息. 计算流的路径后, 使用 Cplex 求解器得到流的传输开始时间 (*SST*), 以最大化可成功调度的流数量, 但是计算时间通常为小时级甚至天级. 动态调度则无法事先知道要调度业务流的全局信息, 每当到来一条流时, 以增量的形式计算这条流的路径和传输开始时间, 该计算过程通常为秒级[6].

本文研究的是动态调度场景. 动态调度存在一些不足: 1) 动态调度场景中流的到达顺序是随机的, 无法更改调度顺序, 因此计算产生的是次优解. 相比于产生最优解的静态调度, 动态调度计算的网络吞吐率低、可调度成功的流数量少; 2) 动态调度的求解目标要求每个流的传输开始时间尽可能小, 这会导致大量的流聚集在超周期的第 1 个时间段进行传输, 其余时间段有着大量的空余时隙.

相关研究提出了在动态调度后进行全局流重新配置的方法[7], 每当动态调度完若干条流后, 就将网络中所有业务流进行重新规划, 来优化链路和时间资源的分配, 为之后到来的流预留调度空间. 然而, 全局流重配置算法的运行时间随流数量的增多而急剧增长, 导致后续到来的流可能无法及时被规划. 因此, 该算法仅适用于几十条或者几百条流的小型网络, 对于典型工业场景, 网络中可能会存在上千条业务流[8], 全局流重配置算法无法在这种大型网络中使用.

本文提出了时间敏感网络业务流动态调度场景中的批量重配置算法 (下文简称为批量重配置算法), 该算法遵循无等待调度机制, 使用整数线性规划 (ILP) 对问题进行建模, 并通过 Cplex 求解器来对问题进行求解. 本文的贡献如下.

(1) 提出了一种批量重配置算法, 在动态调度场景中, 每处理若干条流后, 选择一部分已调度的流进行重新规划, 来优化网络资源的配置.

(2) 提出了新的评价指标——网络吞吐率, 该指标综合考虑了流数量、流周期和流大小带来的影响.

(3) 给出了负载最高流和相关流的选取标准, 从而选出重配置的流集合.

(4) 优化路径选择标准和传输开始时间计算, 为网络中后续到来的流预留出更多可用资源.

(5) 通过实验, 我们验证了本算法可以在数千条流的大型网络中运行, 相比于动态调度, 本算法在网络吞吐率和调度成功的流数量指标上有 16.5% 和 5.5% 的提升, 且算法运行时间满足秒级要求.

(6) 与全局重配置方案相比, 本算法在各项指标上表现良好, 与分组全局重配置方案[9]对比, 本算法拥有短暂且稳定的计算时间, 在网络吞吐率和调度成功的流数量指标上仅减少 1%–2% 左右.

本文的结构如下. 第 2 节介绍相关工作. 第 3 节介绍背景和现存问题. 第 4 节给出批量重配置算法. 第

5 节展示该算法的实验结果. 第 6 节则是总结与展望.

2 相关工作

TSN 业务流调度方面, 目前已有不少专家和学者对其进行了工业应用研究和学术研究. Huang 等人[6]提出了一种基于 TAS 的在线路由和调度机制. 使用可变时隙并最小化每个流的传输开始时间. 设计了一种新的增量式路由调度算法 (IRAS) 来实现流量部署, 并采用预路由算法减少运行时间. Nayak 等人[10]提出了时间敏感软件定义网络 (TSSDN), 为系统中时间触发流的传输提供实时保证. 文献[10]提出了各种 ILP 公式, 解决了时间触发流的路由和调度组合问题. 随后, Nayak 等人[11]又提出了在 TSSDN 中增量添加时间触发流的算法. 该算法可以在亚秒级内计算时间触发流的增量调度, 平均相对最优性为 68%. Dürr 等人[4]引入无等待分组调度机制来模拟时间敏感网络中的业务流调度问题, 并将其映射为运筹学领域中的无等待车间作业调度问题. 文献[4]提出了一种用于高效计算调度的 Tabu 搜索算法和一种用于减少调度中保护带数目的调度压缩技术.

对于重新配置方面而言, Bülbül 等人[7]提出并评估了支持 SDN (software defined network, 软件定义网络) 的时间敏感网络动态路径配置策略, 提出了 TSOR-P, TSOR-T, TSOR-U 这 3 种机制. 该策略根据当前网络环境中的资源利用率, 对流路径和时间进行重新配置, 来最大化可接受流的数量. Li 等人[12]指出 SDN 为网络更新提供了更加高效的方法—将控制平面与数据平面分离, 数据平面只需转发数据即可, 所有的调度和计算都交给控制平面来进行. 控制平面配置新的流规则, 通过两阶段更新机制来更新网络中流的转发行为. Raagaard 等人[13]提出了一种在运行时进行流动态重新配置的方案. 如果网络不能容纳新到来的流, 则重新调度所有业务流, 来尝试容纳新流.

本文在现有的动态调度算法和全局重配置算法的基础上, 综合考虑了重配置的流数量和运行时间, 提出一种改进的批量重配置算法. 该算法每次仅选出网络中的部分业务流进行重新配置, 能减少算法运行时间, 同时为后续到来的流预留出更多资源.

3 背景和现存问题

3.1 背景

2018 年发布的 IEEE 802.1Qcc 中定义了 TSN 的 3 种网络配置模型, 分别是全分布式模型、集中式网络/分布式用户模型和全集中模型[1]. 本文对应的是全集中模型, 该模型借鉴了 SDN 的思想, 将网络的控制逻辑与底层的硬件进行分离, 可以更好地对网络进行管理[5]. 在全集中模型中, 底层的网络设备 (终端、交换机等) 负责数据的转发, 并将网络和设备自身的状态信息上传至集中网络配置 CNC, 并且接收 CNC 下发的指令, 根据指令来调整设备相关状态, 以适应网络的变化. CNC 向上层的集中用户配置 CUC 提供开放的网络资源和能力, 并向下层的网络设备下发指令, 来实现设备控制、流量调度、网络监控等功能. CUC 则根据需要向用户提供自适应服务, 来实现全网拓扑发现、网络状态监测和业务连接与调度等功能[14]. 全集中模型为业务流的动态调度和重新配置提供了基础.

TSN 业务流调度的核心在于计算出业务流的全局调度时间表以及合成门控队列 GCL, 从而让业务流在整个网络范围内同步地、周期性地进行传输. 动态调度适用于需要一定灵活性的网络, 例如在工业自动化场景中, 操作者需要往生产线中添加“即插即用”设备, 此类业务流的到来是不可预知的, 网络需要在秒级时间内计算出该设备是否可以部署的结果[15,16], 并更新 TSN 交换机的配置[10]. 相关研究[6]提出了增量路由与调度算法 (IRAS) 来应对 TSN 业务流动态调度的场景, 该算法已知网络的超周期, 并使用了无等待调度机制, 给出了 3 个约束: 传输开始时间约束、无冲突时隙约束和相邻边约束, 并以最小化每个流的传输开始时间作为计算目标.

业务流动态调度会引起资源浪费的问题, 不时地对网络进行重新配置可以改善现有业务流的资源分配, 并提高网络所能容纳的最大业务流数量[7]. 业务流的重新配置包括被动重新配置和主动重新配置, 被动重新配置是如果网络不能容纳该业务流, 则重新调度网络中的所有业务流, 来尝试容纳新业务流[13]. 主动重新配置则是定期进行全局重新配置, 每新增若干条流, 就对网络中的所有业务流进行重新配置[7].

3.2 现存问题

传统动态调度和全局重新配置均存在一定不足.

(1) 评价指标过于片面. 当前的动态调度算法的主要目标是提升网络中调度成功的流数量[6]. 该评价指标未考虑流周期和流大小带来的影响, 难以对业务流的负载大小进行区分, 也无法根据该指标来比较网络的

拥塞程度. 因此, 批量重配置算法提出了新的评价指标——网络吞吐率, 来综合比较业务流的数量、大小和周期.

(2) 不可更改的调度顺序. 当前的动态调度算法只能按照业务流的到达顺序进行调度, 容易产生时隙碎片, 如图 1 所示. 动态调度算法先调度 f_3, 后调度 f_4, 会产生 2 μs 的时隙浪费. 批量重配置算法则可以自由地对部分流的调度顺序进行排列组合, 先调度 f_4, 后调度 f_3, 可以更充分地利用时隙资源, 减少时隙碎片.

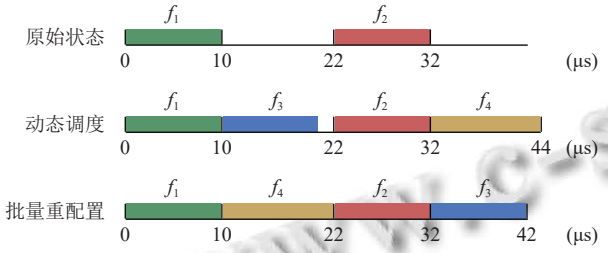


图 1 不同调度顺序引起的时隙浪费

(3) 路径选择标准只考虑流数量. 当前的动态调度路径选择标准仅考虑了流路径上已存在的业务流数量[6], 但是如前所述, 只考虑业务流数量而不考虑流周期和流大小是片面的, 无法很好起到负载均衡的效果. 因此, 批量重配置算法将网络链路利用率方差纳入路径选择标准, 可以更客观地反映网络链路的负载情况.

(4) 动态调度的求解目标导致时隙资源浪费. 当前的动态调度算法的求解目标是最小化业务流的传输开始时间[6]. 这会使得业务流的调度集中在第 1 个基周期[11]所在的传输时间范围中, 而其余传输时间范围中的流数量较少, 从而导致业务流的调度时间不均匀, 造成时隙资源的浪费. 如图 2 所示, 该图展示了在超周期为 4 ms 的网络中, 某个交换机端口的传输时隙分布. 业务流 f_1 和 f_3 的周期为 4 ms, f_2 和 f_4 的周期为 2 ms, f_5 的周期为 1 ms, 所有流的传输时间均为 10 μs, 且流的到来顺序为 f_1, f_2, f_3, f_4, f_5. 若采用动态调度算法, 且求解目标为最小化业务流的传输开始时间, 会产生图中动态调度所示的调度结果. 由于低频流 f_1 和 f_3 各占据了第 1 个传输时间范围中的一段传输时隙, 会导致网络中周期为 1 ms 和 2 ms 的高频流大大减少, 空余的时隙资源也很难被充分使用, 造成大量时隙资源浪费. 若增加批量重配置算法, 则可以分散业务流的传输时间, 避免业务流产生聚集. 如图 2 中批量重配置方法所示的调度结果, 业务流经过调整之后, f_1 和 f_3 可以分配到不同传输时间范围的相同时隙段进行传输, f_2 和 f_4 同理, f_5 的传输开始时间提前了 20 μs, 可以为后续到来的流节省时隙资源.

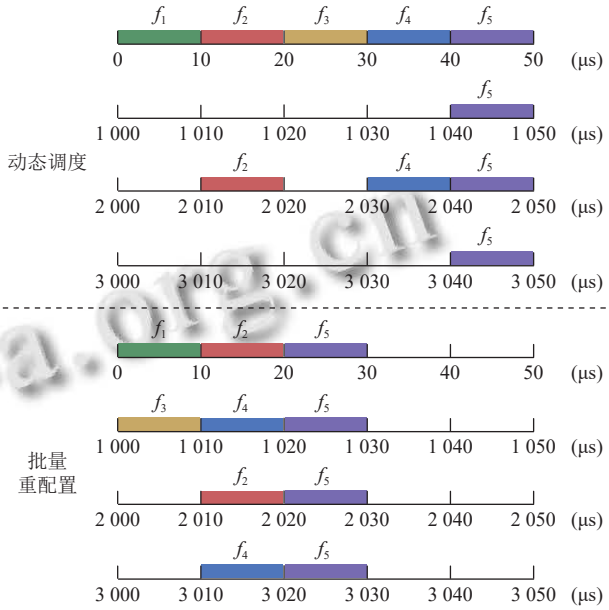


图 2 不同求解目标产生的传输时间分布

(5) 全局重配置计算时间过长. 全局重配置需要对网络中的所有业务流进行重新规划, 其时间复杂度最坏情况下为 $\mathrm{O}\left(n!\times n^3N^3\right)$[4,6], 其中 n 为要计算的流数量, N 为任意流中的总操作 (包括处理和传输). 当网络中业务流数量增多时, 全局重配置很难在合理的时间内给出计算结果. 批量重配置算法则对问题规模进行了缩减, 每次选取网络中的一部分业务流进行重新配置, 可以减少重配置算法的运行时间.

4 算法设计

4.1 关键指标

为了更好地衡量算法在 TSN 业务流动态调度场景下的性能, 本节给出 4 个关键的性能指标: 网络吞吐率、调度成功的流数量、一轮重配置平均计算时间和传输时间范围方差. 其中网络吞吐率为主要指标, 其余 3 个为辅助指标. 本文涉及的符号如表 1 所示.

(1) 网络吞吐率. 如前所述, 现有评价标准, 即网络中调度成功的流数量[6], 无法客观和准确地反映业务流的负载情况. 因此, 本文提出了新的评价指标网络吞吐率. 网络吞吐率综合考虑了流大小和流周期带来的影响, 其含义是单位时间内网络成功传输的字节数量. 网

络吞吐率的定义如式 (1) 所示:

$$网络吞吐率=\left(\sum_{i=1}^{n} f_i.size\times(sp/f_i.p)\right)\Big/sp \quad (1)$$

(2) 调度成功的流数量. 本文保留了原始动态调度的评价指标, 只是不再将其作为主要指标, 而是作为辅助指标进行参考. 将其与主要指标网络吞吐率共同进行研究, 可以更客观、更合理地分析网络当前的负载情况. 在大部分情况下, 调度成功的流数量和网络吞吐率是正相关的.

(3) 一轮重配置平均计算时间. 该指标的含义是运行一次重配置算法所要花费的平均计算时间, 动态调度场景要求该时间为秒级. 网络吞吐率和一轮重配置平均计算时间呈负相关, 极致地追求网络吞吐率将导致不可容忍的计算时间, 会对后续到来的业务流产生影响.

(4) 传输时间范围方差. 超周期可以划分为若干个时间范围 (基周期), 传输时间范围方差指的是在这些传输时间范围中整个网络被占用的传输时隙数量的方差. 其定义如式 (2) 所示:

$$传输时间范围方差=\frac{1}{n}\sum_{i=1}^{n}(time\_slot\_num_i-time\_slot\_num\_avg)^2 \quad (2)$$

其中, n 为超周期和基周期的比值, 变量$time_slot_num_i$代表某个传输时间范围内, 网络中业务流所占据的传输时隙数量之和. 其定义如式 (3) 所示. 变量$time_slot_num_avg$是所有$time_slot_num_i$的平均值.

$$time\_slot\_num_i=\sum_{t=(i-1)\times bp}^{i\times bp}\sum_{k=1}^{flow\_num} FT[f_k][t] \quad (3)$$

传输时间范围方差可以用来衡量流的调度时间在不同时间范围中的分散程度. 该指标越低, 说明流的调度时间越均匀, 网络吞吐率提升的可能性就越大.

表 1 本文涉及的符号

符号	符号说明	符号	符号说明
sp	超周期, 所有流周期的最小公倍数	$f_i.size$	流的大小, 单位为字节
$f_i.p$	流的周期, 单位为ms	bp	基周期, 所有流周期的最大公约数
$FT[f][t]$	流和时隙的映射表, 取值为0或1. 代表流f是否在t时刻进行传输	$u[f][e]$	路径表, 取值为0或1. 表示流f是否占据边e进行传输
$t[f][e]$	时间表, 代表流f在边e处的传输开始时间	K	重配置周期, 每到来K个流进行重配置
M	重配置的流集合大小, 每次重配置M个流	$f.t$	流的一跳传输时间, 单位为μs

4.2 指标平衡

批量重配置算法的 3 个辅助指标与主要指标之间存在着相辅相成、相互制约的关系, 因此需要对 4 个关键指标进行平衡. 结合第 3.2 节中存在的问题, 以及第 4.1 节中各关键指标之间的关系, 批量重配置算法通过以下改进来平衡各关键指标.

(1) 对业务流进行重新配置. 不可更改的业务流调度顺序决定了时隙浪费是动态调度不可避免的问题, 如果要减少时隙浪费, 那么就必须更新网络资源的配置, 优化其中不合理的地方. 反映在关键指标上, 重配置可以提升网络吞吐率和调度成功的流数量, 但同时会引入额外的计算时间.

(2) 优化路径选择标准. 相比于流数量, 链路利用率方差可以更合理、更客观地体现网络链路负载的均衡情况, 更适合作为路径选择的标准. 选取路径时应优先考虑那些能使得网络链路利用率方差最低的路径, 有利于平衡网络中各链路的利用率, 从而减少瓶颈链路的出现. 反映在关键指标上, 优化路径选择标准可以提高网络吞吐率和调度成功的流数量.

(3) 优化业务流的传输时间计算. 动态调度的求解目标会产生传输时间集中的问题. 批量重配置算法可以通过增加约束条件和优化求解目标来避免流的聚集. 反映在关键指标上, 优化传输时间计算可以降低传输时间范围方差以及降低一轮重配置平均计算时间.

(4) 缩减重配置的业务流规模. 由第 3.2 节中给出的时间复杂度可知, 问题规模直接决定了重配置算法的计算时间. 要想满足动态调度场景的秒级时间要求, 就需要对问题规模进行缩减, 因此需要进行“批量重配置”.

4.3 算法流程

批量重配置算法在业务流动态调度之后, 添加了一个尾处理过程, 因此本算法的系统模型及使用的符号和变量与 IRAS 类似[6]. 批量重配置算法综合考虑了动态调度和全局重配置机制的优点, 可以定期优化网络配置. 该算法的流程如算法 1 所示, 其中涉及的具体细节在算法后给出.

算法 1. 批量重配置算法

输入: 拓扑 G, $u[f][e]$, $t[f][e]$, 流集合 $list$, 重配置周期 K, 重配置流集合大小 M, 闭集 $close$.

输出: 调度结果 $Result$, $u[f][e]$, $t[f][e]$.

1. if 调度的流数量不为 K 的倍数 then
2. $Result$ =“不进行重配置”;
3. 返回 $Result$, $u[f][e]$, $t[f][e]$;
4. end if
5. 计算出网络中负载最高的流 $f_{\max ROI}$;
6. 计算出与负载最高流 $f_{\max ROI}$ 相关的 M–1 个流, 组成集合 FL;
7. 释放流集合 FL 中所有流占据的链路和时间资源;
8. 根据新的路径选择标准, 为流集合 FL 中的每个流重新计算路径 R;
9. 通过求解器计算 FL 中所有流的传输开始时间集合 $list\_sst$;
10. if $list\_sst$ 不为空 then
11. 得到每个流的 SST, 将新的路径和时间记录在 $u[f][e]$ 和 $t[f][e]$ 中;
12. $Result$ = “重配置成功”;
13. else
14. 恢复流集合 FL 中每个流的最初路径和传输时间, 恢复 $u[f][e]$ 和 $t[f][e]$;
15. $Result$ = “重配置失败”;
16. end if
17. 将流集合 FL 中的每个流加入闭集 $close$;
18. 返回 $Result$, $u[f][e]$, $t[f][e]$;

批量重配置算法在动态调度算法之后运行, 先判断流数量是否为 K 的倍数, 如果不是算法终止. 如果是, 则先计算出网络中负载最高的流, 然后计算出 M–1 个相关流, 组成流集合 FL. 随后释放 FL 中每个流的链路和时间资源. 根据新的路径选择标准 $WT\_Variance$, 为每个业务流重新计算路径 R. 根据路径和流集合 FL, 利用 Cplex 求解器计算出 FL 中每个流的传输开始时间. 如果计算成功, 则更新流的传输开始时间 SST、$u[f][e]$ 和 $t[f][e]$, 标记重配置成功. 如果计算失败, 则恢复流的路径和传输开始时间, 标记重配置失败. 将 FL 中的流加入闭集 $close$, 下次重配置时不会再选取到这些流, 随后返回重配置结果.

(1) 计算重配置流集合. 本文提出了资源占用指数 ROI 来衡量网络中某个流占据链路和时间资源的多少. ROI 的定义如式 (4) 所示, 其中, $Length$ 为流路径跳数, $Transmission$ 为流在交换机端口的一跳传输时间, $Period$ 为流周期.

$$ROI = \frac{Length \times Transmission}{Period} \tag{4}$$

本文定义了关联度 $Relevancy$ 来选取与负载最高流相关的其他流, 如式 (5) 所示. 其中, OL 和 OT 为流与负载最高流重叠的路径跳数和传输时间长度.

$$Relevancy = OL \times OT \tag{5}$$

(2) 优化路径选择标准. 批量重配置算法改进了动态调度路径选择标准 WT[6], 引入了当前网络中链路利用率的方差, 提出了新的路径选择标准 $WT\_Variance$, 如式 (6) 所示. 其中, $new\_variance$ 是流 f 的路径加入网络之后, 网络的链路利用率方差. $max\_variance$ 是网络中出现过的最大链路利用率方差.

$$WT\_Variance = \frac{HC}{2(1+|\varepsilon|)} + \frac{new\_variance}{(1+\max\_variance)} \tag{6}$$

(3) 优化业务流的传输时间计算. 除了无等待调度机制的基本约束[6]之外, 还需要额外添加两个约束: FL 无冲突时隙约束和分散 SST 约束.

FL 无冲突时隙约束如式 (7) 所示. 其规定了对于 FL 中任意两个不相同的流 f_n 和 f_c, 使得它们在传输路径中每个交换机端口处的传输时间不冲突.

$$\begin{cases} \forall f_n \in FL, \forall f_c \in FL, f_n \neq f_c, e \in \varepsilon: \\ \forall x \in X, \forall y \in Y: \\ \text{if } (u[f_n][e] + u[f_c][e] \geqslant 2) \text{ then} \\ \quad (t[f_n][e] + x \cdot f_n.p \geqslant t[f_c][e] + y \cdot f_c.p + f_c.t \text{ or} \\ \quad t[f_c][e] + y \cdot f_c.p \geqslant t[f_n][e] + x \cdot f_n.p + f_n.t) \text{ with} \\ \quad X = \{x \in N: 0 \leqslant x \leqslant sp/f_n.p\} \\ \quad Y = \{y \in N: 0 \leqslant y \leqslant sp/f_c.p\} \end{cases} \tag{7}$$

分散 SST 约束如式 (8) 所示. 其目的是把流的传输开始时间 SST 分散到各个传输时间范围中. 一方面可以避免不同的流在某一时间段产生聚集, 另一方面缩小 SST 的计算范围可以加快求解器的计算速度.

$$\begin{cases} \forall f_n \in FL, x = [0,0,\cdots,0], x.length = sp/bp: \\ x[f.p] \cdot bp \leqslant f.SST \leqslant (x[f.p]+1) \cdot bp \text{ with} \\ x[f.p] = x[f.p] + 1,\ x[f.p] = x[f.p] \mod (sp/f.p) \end{cases} \tag{8}$$

其次, 不同于动态调度[6], 批量重配置算法定义了新的求解目标, 如式 (9) 所示:

$$\forall f \in FL, \min(sum((sp/f.p) \times f.t \times (f.SST \bmod bp))) \tag{9}$$

新的求解目标综合考虑了流的周期和传输时间, 不同周期和传输时间的流被赋予不同的权重, 使得那些周期较小、传输时间较长的流的传输开始时间 SST 尽可能小. 新的求解目标让流的调度更加紧密, 有效减少了时隙碎片的产生.

5 仿真实验

本节对批量重配置算法进行仿真实验, 首先给出实验设置, 然后给出实验结果与分析.

5.1 实验设置

本文进行了 4 组实验. 实验①: 设计了基于传统动态调度算法 (IRAS) 和批量重配置算法 (BR) 的对比实验. 目的是比较动态调度和不同 K 和 M 下批量重配置算法的运行结果. 实验②: 设计了 IRAS 和 BR 在不同流周期和流大小组合下的对比实验, 并对每种组合下 IRAS、BR-TIME (只优化时间计算)、BR-PATH (只优化路径选择) 和 BR (同时优化时间计算和路径选择) 这 4 种调度算法进行了消融实验. 目的是验证 BR 算法在不同流周期和流大小组合下的运行情况及可靠性, 以及探寻不同的优化指标对于算法提升效果的影响. 实验③: 设计了基于全局重配置方案 (GR) 和批量重配置方案的对比实验. 目的是探寻全局重配置方案和批量重配置方案在 4 个评价指标上的差异. 实验④: 设计了分组全局重配置方案 (GRBG)[9]和批量重配置方案的对比实验. 目的是在实验③未能充分展现出的网络吞吐率和流数量指标上进行扩展实验. 其中, BR 的 K 和 M 均为 10, GR 和 GRBG 的 K 为 10, GRBG 的分组大小为 10, 15 和 20. 每个实验均进行了 5 次重复实验, 取其平均值作为结果进行展示. 实验条件如下.

(1) 网络拓扑. 对于实验①和实验②, 使用添加链路的 OCEV 拓扑[6], 该拓扑共有 44 个节点, 53 条边. 对于实验③和实验④, 使用工业中常见的雪花拓扑[17], 该拓扑共有 37 个节点, 36 条边.

(2) 流量特征. 对于实验①, 使用周期为{1 ms, 2 ms, 4 ms, 8 ms}, 大小为 50–1 500 字节的业务流. 对于实验②, 使用如表 2 所示的不同周期和大小的流组合. 对于实验③和实验④, 使用周期为{0.5 ms, 1 ms, 2 ms, 4 ms}, 大小为 1500 字节的业务流. 所有业务流都是在 IEC/IEEE 60802 标准[18]的指导下生成的.

(3) 机器配置. 本文所有仿真实验都是在服务器上运行的. 该服务器的 CPU 为 Intel(R) Xeon(R) CPU E5-2660, RAM 为 125 GB, Linux 版本为 4.15.0.

5.2 实验结果与分析

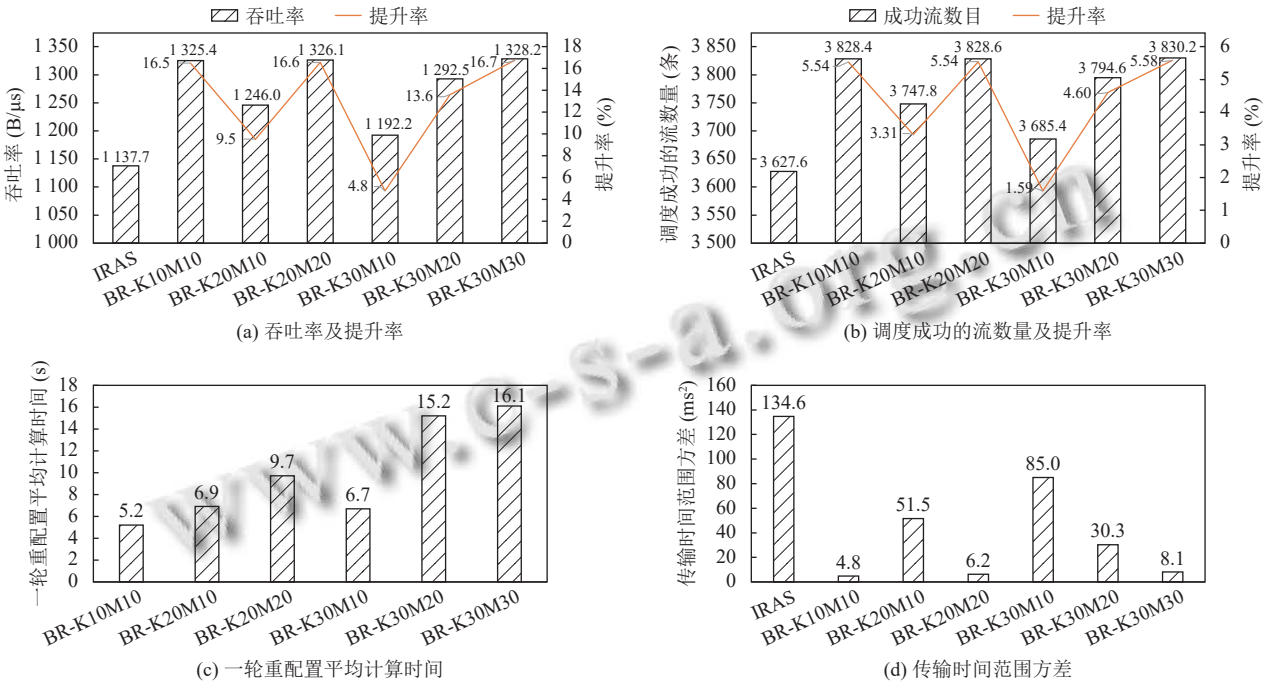
实验①的结果如图 3 所示, 1) 增加了 BR 算法之后, 网络吞吐率和调度成功的流数量均优于原始的动态调度算法. 2) 在不同 K 和 M 的组合下, BR 算法在网络吞吐率方面提升 4.8%–16.7%, 在调度成功的流数量方面提升 1.59%–5.58%. 3) 在大多数组合下, 一轮重配置平均计算时间可以满足动态调度的秒级时间要求. 4) 在传输时间范围方差方面, BR 算法可以不同程度地改善该指标, 避免流的调度时间过于集中. 5) 当 K 不变时, 随着 M 的增大, BR 算法的吞吐率和调度成功的流数量随之提升, 传输时间范围方差逐渐降低, 但与此同时, 一轮重配置的平均计算时间也在提升. 6) 当 M 不变时, 随着 K 的减小, BR 算法的吞吐率和调度成功的流数量逐渐增大, 传输时间范围方差逐渐降低, 一轮重配置平均计算时间则没有明显变化.

表 2 实验②使用的流周期和流大小组合

组合类型	流周期集合 (ms)	流大小范围 (B)
组合1	{1, 2}	1 000–1 500
组合2	{1, 2}	500–1 500
组合3	{1, 2}	50–1 500
组合4	{1, 2, 4}	1 000–1 500
组合5	{1, 2, 4}	500–1 500
组合6	{1, 2, 4}	50–1 500
组合7	{1, 2, 4, 8}	1 000–1 500
组合8	{1, 2, 4, 8}	500–1 500
组合9	{1, 2, 4, 8}	50–1 500

实验②的结果如图 4 所示. 1) 网络吞吐率方面, 在 9 种不同的流组合下, BR 算法在网络吞吐率方面均有一定的提升, 提升范围为 1.2%–16.4%; 随着流大小和流周期范围的增大, 网络吞吐率的提升效果也在增强. 2) 调度成功的流数量方面, 在组合 6、组合 7、组合 8 和组合 9 下, BR 算法在调度成功的流数量方面有一定的提升, 提升范围在 1.5%–5.5%; 而在其余的流组合情况下, 由于 BR 方案节省出的链路和时间资源被调度频繁、字节数更大的流所占据, 因此在流数量方面有一定下降, 通过分析网络吞吐率和流数量的关系可以得出这一结论. 且随着流大小和流周期范围的增大, 调度成功的流数量的提升效果也在增强. 3) 一轮重配置的平均计算时间方面, BR-TIME 和 BR 因为限定了流的传输时间范围, 加快了求解速度, 因此在所有 9 种流组合下, 一轮重配置的平均计算时间均在 10 s 以内, 满足动态调度的秒级时间要求; 而 BR-PATH 没有优化时间分配, 所以在流周期和流大小组合范围较大时, 计算复杂度会增加, 因此重配置的计算时间也会较长, 在组合 7、组合 8 和组合 9 的情况下无法满足动态调度的秒级时间要求. 但随着流周期和流大小的组合范围逐渐缩小, BR-PATH 的计算时间也在减小, 因此在其他流组合的情况下, BR-PATH 逐渐能够满足秒

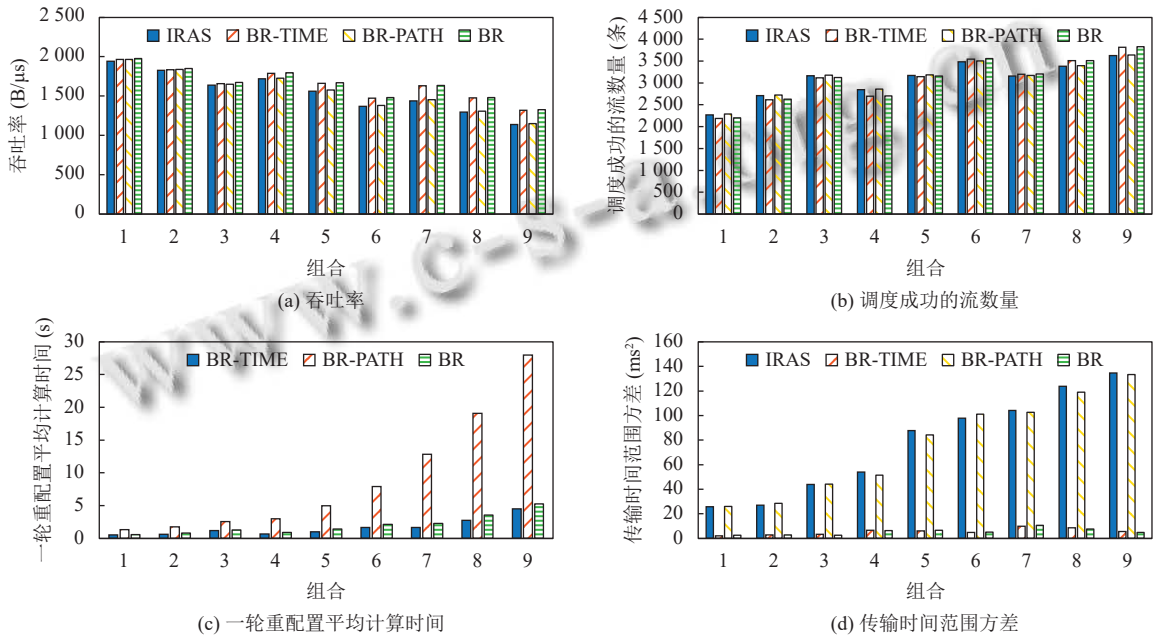
级计算时间的要求. 4) 在传输时间范围方差方面, 由于 BR-TIME 和 BR 将流的传输开始时间均匀分布到各个时间范围中, 因此 BR-TIME 和 BR 显著降低了传输时间范围方差, 使得流的分布更加均匀. BR-PATH 并未优化时间计算, 因此该指标与 IRAS 处于同一个数量级. 5) 在大部分组合下, BR 算法在网络吞吐率和调度成功的流数量上的提升效果超过了 BR-TIME 和 BR-PATH, 对二者的提升效果进行了叠加, 获得了更高的提升率.



(a) 吞吐率及提升率　(b) 调度成功的流数量及提升率

(c) 一轮重配置平均计算时间　(d) 传输时间范围方差

图 3　实验①结果



(a) 吞吐率　(b) 调度成功的流数量

(c) 一轮重配置平均计算时间　(d) 传输时间范围方差

图 4　实验②结果

实验③的结果如图 5 所示, 1) 在 20–40 条流的情况下, BR 和 GR 在网络吞吐率和调度成功的流数量上均相同. 2) 在一轮重配置平均计算时间方面, BR 稳定在 0.2 s 左右, 而 GR 的一轮重配置平均计算时间随着

流数量的提升而急剧增长, 无法满足秒级时间要求. 在 45 条流时, GR 算法在 24 h 之后没有返回计算结果, 因此无法将 GR 扩展到流数量较多的大型网络中. 3) 在传输时间范围方差方面, BR 和 GR 均可有效减小该指标, 使得流的调度时间均匀到各个传输时间范围中.

实验④的结果如图 6 所示, 1) 将流数量扩展到 500 时, 相比于 IRAS 算法, BR 在吞吐率方面大约有 20.3% 的提升, 在流数量方面提升了 8.3%. 相比于 GRBG, BR 在吞吐率上低 1%–2%, 在流数量上低 0.5%–1%. 2) BR 的计算时间在 1 s 左右, GRBG 的计算时间则随着分组数量的增大而急剧增长. 在分组数量为 20 时, 一轮重配置平均计算时间达到了 268 s, 无法满足动态调度的秒级时间要求. 3) BR 和 GRBG 有效降低了传输时间范围方差, 使流的调度时间更加均匀.

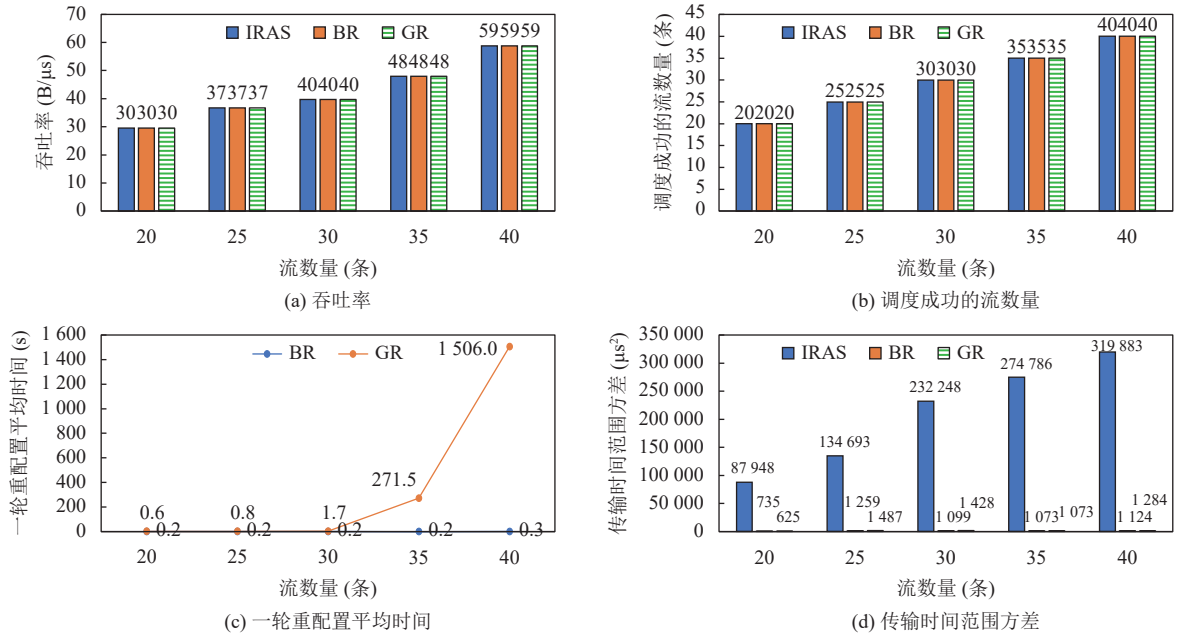


图 5 实验③结果

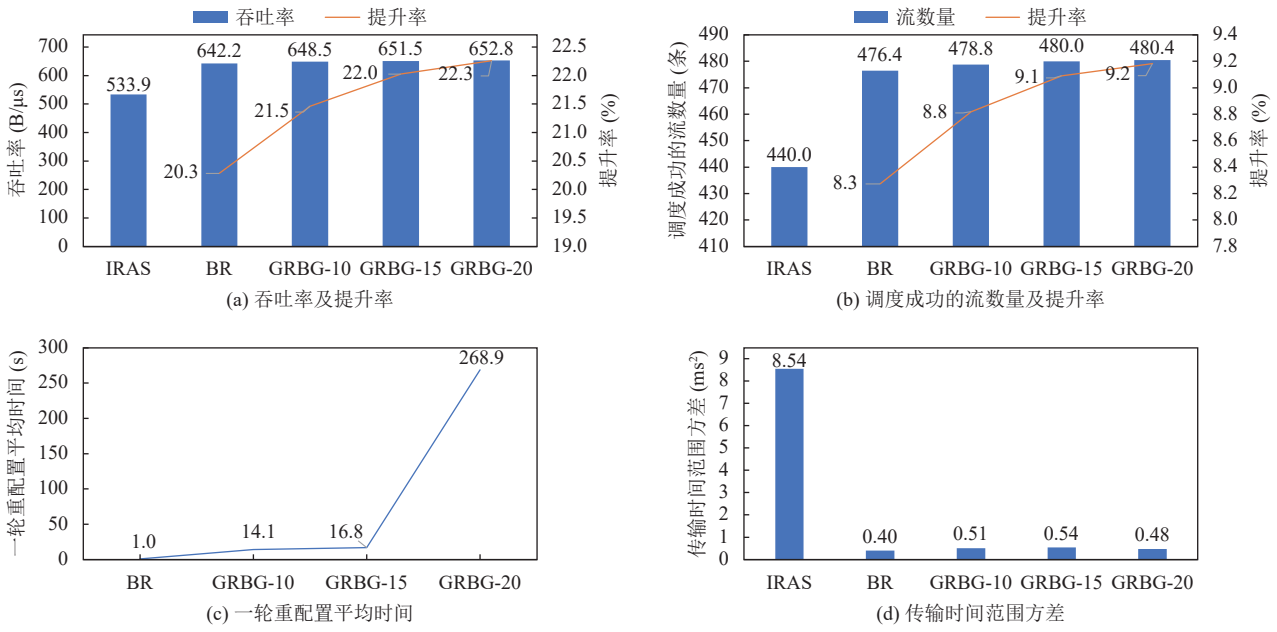


图 6 实验④结果

6 总结与展望

本文设计了时间敏感网络业务流动态调度场景中的批量重配置算法, 该算法在动态调度的基础上, 能够定期对部分业务流进行重新配置, 优化网络资源的分配. 实验结果表明, 该算法能够适用于上千条流的大型网络, 在吞吐率方面提高 16.5%, 在调度成功流数量方面提升 5.5%, 并且计算时间满足秒级要求. 与分组全局重配置算法相比, 该算法在计算时间上拥有巨大优势, 且在吞吐率和流数量上仅减少 2% 和 1% 左右.

在工业场景中, 可能会存在不同传输速率的异构交换机, 并且各交换机之间的线路长短也可能不一样. 因此探讨该算法在复杂网络场景下的适用情况, 是未来需要关注的一个问题.

参考文献

1 中国电子技术标准化研究院. 时间敏感网络白皮书. 北京: 中国电子技术标准化研究院, 2020.

2 Craciunas SS, Oliver RS, Chmelík M, *et al*. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. Proceedings of the 24th International Conference on Real-time Networks and Systems. Brest: ACM, 2016. 183–192. [doi: 10.1145/2997465.2997470]

3 Pozo F, Steiner W, Rodriguez-Navas G, *et al*. A decomposition approach for SMT-based schedule synthesis for time-triggered networks. Proceedings of the 20th Conference on Emerging Technologies & Factory Automation. Luxembourg: IEEE, 2015. 1–8. [doi: 10.1109/ETFA.2015.7301436]

4 Dürr F, Nayak NG. No-wait packet scheduling for IEEE time-sensitive networks (TSN). Proceedings of the 24th International Conference on Real-time Networks and Systems. Brest: ACM, 2016. 203–212. [doi: 10.1145/2997465.2997494]

5 IEEE. 802.1Q-2022 IEEE standard for local and metropolitan area networks-bridges and bridged networks. https://1.ieee802.org/maintenance/802-1q-2022-rev/. (2018-05-07).

6 Huang YD, Wang S, Huang T, *et al*. Online routing and scheduling for time-sensitive networks. Proceedings of the 41st International Conference on Distributed Computing Systems. Washington: IEEE, 2021. 272–281. [doi: 10.1109/ICDCS51616.2021.00034]

7 Bülbül NS, Ergenç D, Fischer M. Towards SDN-based dynamic path reconfiguration for time sensitive networking. Proceedings of the 2022 IEEE/IFIP Network Operations and Management Symposium. Budapest: IEEE, 2022. 1–9. [doi: 10.1109/NOMS54207.2022.9789890]

8 Jin X, Xia CQ, Guan N, *et al*. Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries. IEEE Access, 2020, 8: 6751–6767. [doi: 10.1109/ACCESS.2020.2964690]

9 Mahfouzi R, Aminifar A, Samii S, *et al*. Stability-aware integrated routing and scheduling for control applications in Ethernet networks. Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition. Dresden: IEEE, 2018. 682–687. [doi: 10.23919/DATE.2018.8342096]

10 Nayak NG, Dürr F, Rothermel K. Time-sensitive software-defined network (TSSDN) for real-time applications. Proceedings of the 24th International Conference on Real-time Networks and Systems. Brest: ACM, 2016. 193–202. [doi: 10.1145/2997465.2997487]

11 Nayak NG, Dürr F, Rothermel K. Incremental flow scheduling and routing in time-sensitive software-defined networks. IEEE Transactions on Industrial Informatics, 2018, 14(5): 2066–2075. [doi: 10.1109/TII.2017.2782235]

12 Li D, Wang ST, Zhu KL, *et al*. A survey of network update in SDN. Frontiers of Computer Science, 2017, 11(1): 4–12. [doi: 10.1007/s11704-016-6125-y]

13 Raagaard ML, Pop P, Gutiérrez M, *et al*. Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing. Proceedings of the 2017 IEEE Fog World Congress. Santa Clara: IEEE, 2017. 1–6. [doi: 10.1109/FWC.2017.8368523]

14 赵奕, 王海涛. 时间敏感网络研究现状与关键技术探析. 邮电设计技术, 2022(10): 65–70. [doi: 10.12045/j.issn.1007-3043.2022.10.0013]

15 Said SBH, Truong QH, Boc M. SDN-based configuration solution for IEEE 802.1 time sensitive networking (TSN). ACM SIGBED Review, 2019, 16(1): 27–32. [doi: 10.1145/3314206.3314210]

16 Gutiérrez M, Ademaj A, Steiner W, *et al*. Self-configuration of IEEE 802.1 TSN networks. Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation. Limassol: IEEE, 2017. 1–8. [doi: 10.1109/ETFA.2017.8247597]

17 Li Q, Li D, Jin X, *et al*. A simple and efficient time-sensitive networking traffic scheduling method for industrial scenarios. Electronics, 2020, 9(12): 2131. [doi: 10.3390/electronics9122131]

18 Yan JL, Quan W, Jiang XY, *et al*. Injection time planning: Making CQF practical in time-sensitive networking. Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications. Toronto: IEEE, 2020. 616–625. [doi: 10.1109/INFOCOM41043.2020.9155434]

(校对责编: 孙君艳)