单位代码：10293 密　级：

南京邮电大学

专 业 学 位 硕 士 论 文

论文题目：基于半实物仿真系统的时间敏感网络调度算法研究

学　　号	1220076425
姓　　名	申雷翼
导　　师	张雷
专业学位类别	电子信息硕士
类　　型	全日制
专业（领域）	计算机技术
论文提交日期	2023 年 6 月

Research on Time-Sensitive Network Scheduling Algorithm Based on Semi-physical Simulation System

Thesis Submitted to Nanjing University of Posts and Telecommunications for the Degree of

Master of Electronic Information

By

Leiyi Shen

Supervisor: Prof. Lei Zhang

June 2023

摘要

近年来随着物联网技术的不断发展，自动驾驶、远程医疗等新兴工业应用得到快速的发展。这些新的应用对网络的实时性能提出了更高的要求。由于传统以太网通信技术无法保障时间敏感流的确定性传输，IEEE 802.1 工作组提出了时间敏感网络（Time-Sensitive Networking, TSN）。TSN 通过时间同步、流量调度、路径控制和网络用户配置等机制，实现对时间敏感流的低时延、低抖动和高可靠性传输。

TSN 协议并未制定具体的流量调度方案，合理的流量调度算法是实现时间敏感流确定性传输的关键。同时，新的调度算法在落地实施之前需要在网络仿真平台进行前期功能验证。针对这些问题，本文主要完成以下工作：

（1）设计并实现了 TSN 半实物仿真系统。针对 TSN 半实物仿真问题，对 OMNeT++仿真软件和 NeSTiNg 仿真框架进行分析，并设计实现了实时调度机制和半实物仿真接口。同时针对真实设备和真实程序的半实物仿真，设计开发了两种不同的半实物仿真接口。之后利用对比实验，对本文所设计的半实物仿真接口进行验证分析。最后通过真实物理实验系统、软件仿真系统以及半实物仿真系统的实验结果对比，验证本文所开发的半实物仿真系统性能。实验结果表明，本文设计实现的 TSN 半实物仿真系统可以提供更贴近真实的仿真效果。

（2）提出了一种基于自适应时隙窗口的调度算法。针对 TSN 中动态变化的网络流量，提出了根据时间敏感流时延动态调节对应时隙窗口的调度算法。首先对 TSN 流量调度问题进行分析建模，然后根据时间敏感流时延选择被调节的交换机端口，并根据时延动态调节对应时隙窗口大小。最后，在软件仿真环境和半实物仿真环境下设计对比实验，验证本文所提算法的可行性。实验结果表明，本文所提出的调度算法相比于参考算法更适用于动态流量调度问题，不仅能保障时间敏感流的传输，还有效减小由于流量变化造成的网络抖动。

关键词：时间敏感网络，半实物仿真，时间感知整形，调度算法

Abstract

With the ongoing development of information technology in recent years, new industrial applications like autonomous vehicles and remote medicine have been expanded quickly. These emerging applications proposed higher real-time requirement for data transmission. Traditional Ethernet communication technologies cannot guarantee the deterministic transmission of time-sensitive flows. For this reason, the IEEE 802.1 Working Group has proposed Time-Sensitive Networking (TSN).

Through the combined action of traffic scheduling, network management, and other methods, TSN can ensure the transmission of time-sensitive flows. However, no specific traffic scheduling schemes are defined in TSN protocols. An acceptable traffic scheduling mechanism is crucial for the deterministic transmission of time-sensitive streams. Meanwhile, it is vital to pre-validate the new features of TSN using network simulation because TSN technology is currently in a stage of high-speed development and some protocol functions are still being created. The main work of this thesis are as follows.

(1) TSN semi-physical simulation system is designed and implemented. Based on the analysis of OMNeT++ simulation software and NeSTiNg simulation framework, the real-time scheduling mechanism and semi-physical simulation interface are implemented. Two different semi-physical simulation interfaces are developed for the semi-physical simulation of real devices and real programs. The semi-physical simulation interfaces are verified by comparison experiments. The performance of the developed semi-physical simulation system is verified by comparing the experimental results of the physical experiment system, the software simulation system and the semi-physical simulation system. The experimental results show that the developed TSN semi-physical simulation system can provide a simulation effect closer to the physical system.

(2) For the dynamically changing traffics in TSN, a scheduling algorithm is proposed that dynamically adjusts the time slot window according to the feedback of the transmission delay. The TSN traffic scheduling problem is modeled, and then a scheduling algorithm is proposed. In the proposed algorithm, a switch port to be adjusted are firstly selected, and the corresponding time slot window size is adjusted according to the real-time delay. Finally, comparison experiments are designed in software simulation environment and semi-physical simulation environment. The experimental results show that the proposed scheduling algorithm can adapt to the dynamic traffic

than the reference algorithm. It can not only guarantee the transmission of time-sensitive flows, but also effectively reduce the network jitter caused by traffic changes.

Key words： Time-sensitive networking, Semi-physical simulation, Time aware shaper, Scheduling algorithms

目　录

专用术语注释表

符号说明：

ES	终端节点
SW	交换机
G	网络拓扑
v	网络中所有设备的集合
E	网络中数据链路的集合
p_j	编号为j的交换机出端口
F	网络中所有流的集合
f_i	编号为i的流
U_i	编号为i的流经过的交换机出端口集合
L_i	编号为i的流的数据帧长度
D_i	编号为i的流的允许的最大时延
ES_i^{src}	编号为i的流的源地址
ES_i^{dst}	编号为i的流的目的地址
T_{GCL}	GCL 周期
T_{CT}	CT 时隙窗口大小
T_{BE}	BE 时隙窗口大小
d_i	编号为i的流的当前时延
$\alpha_i(t)$	编号为i的流的到达曲线
$\beta(t)$	服务曲线
S_i	编号为i的流的时隙窗口长度
B	数据传输速率
D^{upper}	触发时隙调节的上限
Q	触发时隙调节上限的流集合
w_j	编号为j的端口权重系数
O_j	最后一帧 CT 流经过交换机出端口的偏移时间
$Step$	调节步长
$Step_{max}$	单次调节步长上限
Exp	时隙调节比例因子
T_{CT}^{low}	CT 时隙窗口下边界

缩略词说明：

ASW	Adaptive Slotted Window	自适应时隙窗口
AVB	Audio Video Bridge	音视频桥接流
BE	Best Effort	尽力而为流
CNC	Centralized Network Configuration	集中式网络配置

CT	Control-Data Traffic	控制数据流
FES	Future Event Set	未来事件队列
GCL	Gate Control List	门控列表
PCP	Priority Code Point	优先级代码点
QoS	Quality of Service	服务质量
RTT	Round-Trip Time	往返时间
TAS	Time- Aware Shaping	时间感知整形
TSN	Time-Sensitive Networking	时间敏感网络
VLAN	Virtual Bridged Local Area Networks	虚拟局域网

第一章 绪论

1.1 研究背景与意义

近年来，随着工业物联网技术的不断发展，自动驾驶、远程医疗等新兴领域对网络通信的实时性和可靠性提出了更高的要求[1]。这些领域涉及到大量数据传输和即时反馈，因此需要高带宽和低时延的通信网络来保障数据的准确传输和实时响应。在传统以太网中，数据传输过程中会面临竞争，无法保障特定数据的实时性传输。因此在不同应用领域出现了基于以太网的各种通信技术，如工业领域的 PROFINET[2]和 EtherCAT[3]、多媒体领域的(Audio Video Bridge, AVB)[4]以及汽车领域的 FlexRay[5]等。这些通信技术不仅满足了各自领域对实时性通信的需求，而且保持了在相关领域其他应用上的可扩展性。然而，市场上存在不同供应商提供的采用不同技术协议的设备，导致设备之间的兼容性不佳，增加了集成和部署设备的成本。为了进一步提高以太网通信的可靠性、实时性以及不同通信技术之间的兼容性，IEEE 802.1 任务组将 AVB 工作组升级为时间敏感网络（Time-Sensitive Networking, TSN）[6]工作组。TSN 工作组在修订和改进 AVB 协议的基础上，提出了一系列增强机制和流量策略的规范和标准，旨在现有的以太网技术基础上，增加对实时数据传输和可靠性的支持，以满足实时应用对实时性和确定性的要求[7]。同时，TSN 向下兼容标准以太网，可以兼容非实时数据的传输。

TSN 工作组将网络中的流量划分为三种主要类型：控制数据流（Control-Data Traffic, CT）、AVB 流和尽力而为流（Best Effort, BE）。其中，CT 流也称为计划流（Scheduled Traffic, ST）或时间触发流（Time-Triggered, TT），具有最高的优先级，通常具有固定的发送周期和流量大小，具有最严格的时延和抖动要求，例如工业自动化网络中周期性设备检测信号。AVB 流也称为速率限制流，具有次要优先级，对时延有一定的要求，同时必须保障固定的带宽需求。BE 流具有最低的优先级，对时延和抖动没有特殊要求。为了对网络中的流量进行划分，TSN 工作组最先提出了 IEEE 802.1Q[8]虚拟局域网（Virtual Bridged Local Area Networks, VLAN）协议。该协议通过在传统以太网数据帧中加入 VLAN 标签，实现局域网中的虚拟划分和隔离。可以通过 VLAN 标签中的优先级代码，对不同类型的流量进行标记。

为了保障 CT 流的确定性传输，TSN 工作组提出了一系列协议，包括基于时间感知整形（Time- Aware Shaping, TAS）的 IEEE 802.1Qbv 协议[9]，基于循环排队与转发的 IEEE 802.1Qch 协议[10]以及基于异步时间整形的 IEEE 802.1Qcr 协议[11]等。其中，IEEE 802.1Qbv 协议备受关注，该协议规定不同优先级的流量按照其携带的优先级代码，进入最多 8 个不同优先级的队

列。通过门控列表（Gate Control List, GCL）周期性控制不同队列门的开关，调整队列中流量的传输。通过合理的调度算法，可以将 CT 流与其他流量在时域上进行划分，保障其确定性传输。

与此同时，TSN 作为一种新兴技术，部分 TSN 协议仍处于草案阶段[12]。为了评估和验证新的通信协议、机制和技术在实际网络环境下的适用性和效果，需要进行网络仿真。通过进行仿真实验，能够模拟和评估 CT 流的时延、丢包率以及网络负载等关键指标。这些仿真结果能够帮助深入理解和优化网络拓扑结构、时钟同步机制、流量调度算法等关键方面，从而提高时间敏感网络的性能和可靠性。此外，网络仿真还有助于指导和优化 TSN 网络的部署和配置。通过仿真实验，可以评估不同的网络参数设置、拓扑结构设计和流量调度策略，以找到最佳的配置方案。这种实验性的评估和优化可以减少实际部署过程中的试错成本，并提高时间敏感网络的部署效率和性能。与软件仿真相比，半实物仿真可以将真实设备与仿真环境相结合，提供更贴近真实的仿真结果[13]。

在本文中，将对 TSN 半实物仿真系统以及基于 TAS 机制的流量调度算法展开研究。

1.2 研究现状

1.2.1 TSN 仿真研究现状

TSN 通过一系列协议，在现有以太网的基础上提供了高可靠性和低时延的确定性通信技术。随着近几年的快速发展，TSN 引起了工业自动化和车载以太网领域的广泛关注[14][15]。为了快速评估 TSN 网络性能，优化网络配置方案，需要在 TSN 网络部署前进行仿真验证。目前，学术界和产业界提出了多种仿真验证方案，按照验证方式的不同，可以分为真实物理实验验证、软件仿真验证以及半实物仿真验证。

2009 年，英特尔、博通等公司共同组成了 Avnu 联盟[16]，旨在促进和推动 AVB 和 TSN 技术的发展。Avnu 联盟提供了一套工业场景下的 TSN 真实物理实验系统搭建方案，该方案不仅包含了 TSN 相关标准的实现方式和详细系统架构，还包含了包括过程自动化在内的多个工业用例要求。随后，英特尔公司推出了支持 TSN 功能的 I210 网卡，该网卡在硬件层面支持硬件时间戳，支持高精度时间同步。与此同时，英特尔公司提供了利用该网卡搭建 TSN 真实物理实验系统的方案[17]。随着近几年的发展，在 Linux 生态中对 TSN 的支持逐渐完善，例如通过 LinuxPTP 项目[18]可以实现高精度的时间同步等。在 Linux TSN 项目[19]中，提供了一套基于 Linux 系统搭建真实物理实验系统的方案，该方案利用 Linux 操作系统和 Intel I210 网卡，

配合实时性内核，在 Linux 系统中实现 TSN 相关机制。2019 年，为了解决工业场景中语义的互操作性问题和实时通信问题，OPC UA 基金会将 OPC UA 规范与 TSN 技术相结合，提出了 OPC UA-TSN 的标准[20]。Pfrommer 等人[21]基于 OPC UA-TSN 的标准，利用 Linux 系统构建了一套开源的真实物理实验系统。Senk 等人[22]针对 5G 园区场景，构建了一套开源的 TSN-5G 的真实物理实验系统，为 TSN 与 5G 的结合研究提供基础。除了利用 Linux 系统搭建真实物理实验系统外，还可以考虑利用 TSN 芯片搭建。国防科技大学的 Fu 等人[23]提出了一种开源的 TSN 架构并利用该架构开发 TSN 芯片，对 TSN 网络进行验证。

TSN 协议还处于高速发展阶段，部分协议功能还处于开发中，因此需要通过网络仿真的方式，对功能研发进行前期的验证。国内外大量学者利用软件仿真的方式，对所提理论进行仿真验证。Le 等人[24]利用软件仿真的方式，对 TSN 中时间同步以及 TAS 机制进行仿真验证。Arestova 等人[25]利用软件仿真对 TSN 帧抢占机制进行研究，并评估其在工业领域的适用性。Pahlevan 等人[26]针对工业自动化以及车载以太网领域关注的通信可靠性问题，对 TSN 帧复制与消除机制进行仿真研究。Zhu 等人[27]针对车载以太网环境，对 TSN 调度算法进行仿真研究。

半实物仿真验证介于软件仿真验证与真实物理实验验证之间，既可以简化验证难度，又可以提供更贴近真实的仿真结果。然而目前关于 TSN 的半实物仿真研究相对较少，Mariño 等人[28]针对车载以太网环境，提出了一种基于硬件的环回策略，用于在汽车网关中高效、经济地集成 TSN 功能。国防科技大学的汪铮等人[29]基于他们开发的 TSN 芯片，开发了一种硬件仿真器 OpenEmulator。通过时间同步互锁机制，将芯片中的物理时间与仿真器中的仿真时间进行同步，辅助 TSN 芯片的功能验证。然而，这些半实物仿真方法需要配合特定的网络设备或仿真软件，同时源程序并未实现完全开源。因此本文基于独立于网络设备的开源软件，实现 TSN 的半实物仿真。

1.2.2 TSN 调度算法研究现状

为了满足网络中不同类型流量的服务质量需求，TSN 制定了一系列流量调度机制。然而 TSN 工作组只规定了流量调度的一些准则，并没有明确指定具体调度方法，这也让 TSN 流量调度问题成为学术界的研究热点。流量调度是指通过规定数据帧的发送时间或 GCL 信息等，使各个数据帧满足一系列约束条件，如实时性约束、带宽约束、传输隔离约束等。通过合理的调度算法，可以有效地分配资源，提高数据传输的可靠性和效率。流量调度问题已经被证明是一个 NP 完全问题[30]。

针对这类问题，一般可以采用基于搜索的调度算法或直接求解的调度算法进行求解。其

中基于搜索的调度算法一般为启发式算法，如禁忌搜索算法、遗传算法等。直接求解的调度算法一般利用可满足模理论（Satisfiability Modulo Theories, SMT）或整数线性规划（Integer Linear Programming, ILP）的数学方法进行求解。基于 SMT 或 ILP 的方法通过提出一组一阶逻辑约束来定义调度问题，选择不同的优化目标，通过求解器搜索整个解空间来计算最优解。Dürr 等人[31]通过引入无等待约束，将 TSN 调度问题简化为无等待车间调度问题，并利用禁忌搜索进行求解。Pahlevan 等人[32]利用遗传算法，解决 TSN 调度和路由问题。Oliver 等人[33]利用数组理论将 TAS 机制的约束形式化为一组一阶逻辑约束，并利用 SMT 方法求解。Falk 等人[34]利用 ILP 方法求解 TSN 中的调度与路由问题。然而，基于搜索的调度算法在解空间内通过一定的搜索策略来逼近最优解或次优解，在一定的时间限制内停止搜索，求解效率较高，但无法保障解的质量[35]。而直接求解的调度算法通过搜索整个解空间寻找最优解，求解代价随着网络规模的增大呈指数增长，不适合大规模网络。目前也有部分学者通过将大规模求解问题进行分割，采用迭代求解的方式降低求解代价[36][37]。

这些调度算法是针对静态网络场景的，即需要在已知全局网络信息和流量信息的前提下进行调度。虽然这样可以得到接近最优的调度方案，但不适用于动态变化的网络场景，如支持动态重构的工业自动化场景。当网络中信息发生变化时，往往需要对整个网络进行重新调度计算，消耗大量资源。因此，部分学者对动态网络场景中的流量调度问题展开研究。Nasrallah 等人[38]提出了一种作用于 TAS 机制下的网络重调度算法。冯泽坤等人[39]利用 ILP 方法，在已有的静态网络调度结果的基础上对新增流量进行调度规划。周阳等人[40]提出了一个在线启发式算法，解决新增流量的调度与路由问题。这些算法的核心思路是在网络发生变化的情况下，基于现有的调度结果对网络进行重新调度规划，并以最小化更新期间的资源开销为调度目标。同时，这些算法的约束条件都包括 CT 流的无抖动传输，这极大限制了解空间的大小，进而限制了求解速度。Abe 等人[41]证明了在 TSN 中允许小范围的网络抖动可以增加解空间大小，提高网络可调度性。Nasrallah 等人[42]提出了一种基于时隙窗口的动态带宽调节机制，称为自适应时隙窗口（Adaptive Slotted Window, ASW）。该算法根据 CT 流时延动态调节 GCL 中 CT 流和 BE 流之间的时隙窗口大小比例。虽然在时隙窗口变化的过程中，会导致一定的网络抖动，但 Zhao 等人[43]利用网络演算理论，证明合理的时隙窗口大小可以保障 CT 流在最坏情况下满足时延要求。Shen 等人[44]利用 OMNeT++仿真软件验证了基于时隙窗口的动态带宽调节机制的有效性。基于时隙窗口的调度算法相比于全局调度算法，计算机复杂度小，更加灵活。但是上述基于时隙窗口的调度算法大都针对单个交换机，每个交换机独立进行时隙窗口调节，没有考虑多交换机之间的相互影响。因此在面向复杂网络拓扑时，调度结果偏向保守，降低了网络带宽的使用效率。

1.3 主要研究内容

本文面向动态变化的网络，提出一种基于自适应时隙窗口的调度算法，并在自主开发的半实物仿真系统进行验证。本文主要研究内容如下：

（1）针对 TSN 半实物仿真问题，本文将半实物仿真分为实时调度机制和半实物仿真接口两部分，并分别进行设计实现。其次，为了满足真实设备和真实程序的半实物仿真需求，本文设计开发了两种不同类型的半实物仿真接口，通过对比实验验证两种半实物仿真接口的性能。最后，利用真实物理实验系统、软件仿真系统和半实物仿真系统进行对比实验，验证半实物仿真系统的性能。

（2）针对 TSN 中网络动态变化情况下的流量调度问题，本文提出了一种基于自适应时隙窗口的调度算法。首先，对 TSN 调度问题进行分析建模。相比于传统 ASW 算法，本文提出了三点改进：1. 针对复杂网络环境，提出了一种选择被调节端口的方法；2. 提出了一种结合可变步长和固定步长优势的增大时隙窗口的方法；3. 提出了一种根据流传输时间减小时隙窗口的方法。然后，利用 OMNeT++仿真软件和 NeSTiNg 仿真框架，搭建多种 TSN 仿真网络，对本文所提算法进行验证。最后，利用本文所开发的 TSN 半实物仿真系统，对本文所提算法进行更贴近真实的仿真验证。

1.4 论文组织结构

本文的章节内容安排如下：

第一章为绪论部分。首先介绍了本文的研究背景和研究意义，然后对研究现状进行了分析，最后介绍了本文主要研究内容和论文组织结构。

第二章为相关技术介绍。首先对本文所涉及到的 TSN 协议以及功能进行介绍，然后对不同网络仿真方法、工具以及 TSN 仿真框架进行了分析介绍，最后根据 TSN 特点选择合适的仿真工具和框架。

第三章为 TSN 半实物仿真系统。首先对 OMNeT++仿真软件和 NeSTiNg 仿真框架进行分析，将半实物仿真分为实时调度机制和半实物仿真接口两部分并分别设计实现。然后设计对比实验对本文所设计的半实物仿真接口进行验证分析，并对本文所设计的半实物仿真系统性能进行验证分析。

第四章为基于自适应时隙窗口的调度算法。首先对 TSN 流量调度问题进行分析模型，在传统 ASW 算法的基础上进行改进，使其适用于复杂网络的同时尽可能减小网络中的抖动。

然后利用 OMNeT++仿真软件和 NeSTiNg 仿真框架，在不同仿真网络中设计对比实验，验证本文所提算法的可行性。最后利用本文中开发的半实物仿真系统，对所提出的算法进行更贴近真实的仿真。

第五章为总结与展望。对本文中主要工作和创新性进行总结，同时分析了目前的不足，为今后研究和改进的重点进行展望。

第二章 相关技术介绍

本章介绍了本文所涉及的 TSN 协议及其功能，同时介绍了网络仿真方法、工具以及 TSN 仿真框架，并根据 TSN 的特点选择合适的仿真工具与仿真框架。

2.1 TSN 协议

时间敏感网络是由 IEEE 802.1 TSN 工作组提出的一系列以太网协议标准，通过保障时间敏感流传输所需的网络资源，实现时间敏感流的确定性传输，同时兼容非时间敏感流的传输。TSN 是数据链路层协议，对上层协议没有特定要求，因此具有广泛的适用性。TSN 主要通过时间同步、流量调度、路径控制和网络用户配置等机制实现时间敏感流的低时延、低抖动和低丢包率传输。目前，IEEE 802.1 TSN 工作组已经提出了数个协议，并正在标准化过程中。图 2.1 展示了其中标准化程度较高的部分协议。

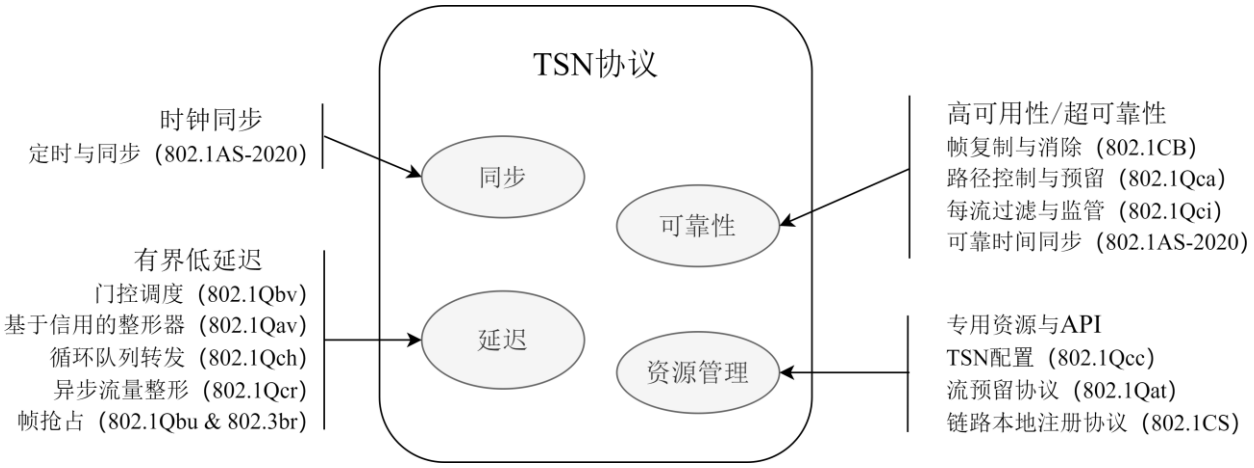


图 2.1 TSN 协议

表 2.1 展示了本文涉及的部分协议，下面将对以下协议具体功能进行描述。

表 2.1 本文研究涉及的 TSN 协议

标准	名称	描述
IEEE 802.1Q-2014	Bridges and Bridged Networks	虚拟桥接局域网协议
IEEE 802.1AS-2011[45]	Timing and Synchronization	时间同步协议
IEEE 802.1Qbv-2015	Enhancements for Scheduled Traffic	门控调度协议
IEEE 802.1Qcc-2018[46]	Stream Reservation Protocol Enhancements and Performance Improvements	流预留协议与系统配置协议

2.1.1 IEEE 802.1Q-2014 虚拟桥接局域网协议

IEEE 802.1Q 协议定义了 TSN 中帧格式，该协议是在局域网中实现虚拟局域网的标准协议。其通过在传统以太网帧的基础上增加 4 字节的 VLAN 标签，将网络划分为多个虚拟子网，使得不同的网络设备可以隔离和通信，提高网络的安全性和灵活性。表 2.2 为 VLAN 标签内容。其中标签协议标识（Tag Protocol Identifier, TPID）固定为 0x8100，占 16 位，用于识别数据帧是否采用 802.1Q 协议；优先级代码（Priority Code Point, PCP）取 0~7，占 3 位，用于确定数据帧的优先级；标准格式指示（Canonical Format Indicator, CFI）默认为 0，占 1 位，标识 MAC 地址是否采用了规范格式；虚拟局域网标识（VLAN Identifier, VID）占 12 位，用于标识数据帧所属的 VLAN。

表 2.2 VLAN 标签

标签协议识别符	优先级代码	标准格式指示	虚拟局域网识别符
TPID (16bits)	PCP (3bits)	CFI (1bit)	VID (12bits)

在 TSN 网络中一个完整的数据帧包括以太网帧头、载荷和循环冗余校验码等。其中，VLAN 标签中的 PCP 标志位负责标识流量优先级。表 2.3 总结了 TSN 网络中不同优先级的流量类型。

表 2.3 流量 QoS 划分

PCP 值	数据流量类型	英文
0	尽力而为	Best Effort
1	背景流	Background
2	卓越努力	Excellent Effort
3	严苛应用	Critical Application
4	时延和抖动小于 100ms 的视频流	Video
5	时延和抖动小于 10ms 的音频流	Voice
6	内部网络控制	Internetwork Control
7	工业网络控制	Network Control

2.1.2 IEEE 802.1AS-2020 时间同步协议

在 TSN 中，为了让网络中所有设备按照特定的调度顺序允许，需要让网络中所有设备对

时间有统一的认知，因此时间同步机制在 TSN 中占重要地位。通过网络内部各个设备之间交换必要的本地时钟信息可以实现局域网范围内的时间同步。时间同步标准主要采用 IEEE 802.1AS。IEEE 802.1AS 协议采用软件和硬件相结合的方法，通过对精确时间同步协议协议进行改进和简化，采用通用精确时间同步协议，在数据链路层实现纳秒级时间同步。

在局域网中，通过最优主时钟算法选取域内主时钟，其余设备作为从时钟，并将自身时间向主时钟进行同步。其核心是通过时间戳机制，时间同步信息在局域网传输的过程中，会触发网络中从时钟设备的采样，从时钟设备通过对称的路径传播时延和时钟补偿技术对本地时钟进行校准，将本地时钟与域内最佳主时钟进行同步，进而实现整个网络的时间同步。

时间同步具体过程如图 2.2 所示。主时钟向从时钟发送同步报文 sync，并记录报文发送的硬件时间戳t_1。从时钟收到同步报文后，记录报文接收的硬件时间戳t_2。主时钟向从时钟发送包含时间戳t_1的跟随报文 Follow_up。从时钟向主时钟发送时延请求报文 Delay_Req，并记录报文发送的硬件时间戳t_3。主时钟收到时延请求报文后，记录报文接收的硬件时间戳t_4，并将该信息封装在延时响应报文 Delay_Resp 中，发送给从时钟。

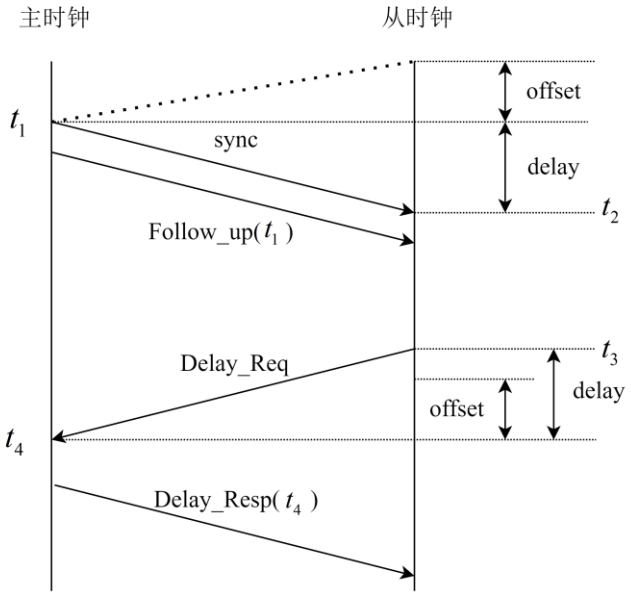


图 2.2 时间同步过程

此时，在从时钟端记录了$t_1 \sim t_4$时间戳。假设主时钟和从时钟的上下行链路是对称的，即主时钟与从时钟间传播时延是相等的。那么，根据公式 2.1 和公式 2.2，从时钟端可以计算出传播时延（delay）和时钟偏移（offset），用于修正自身时钟，实现与主时钟的时间同步。

$$t_{offset} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.1}$$

$$t_{delay} = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{2.2}$$

2.1.3 IEEE 802.1Qbv-2015 门控调度协议

在传统以太网中，多个设备共享有限的网络资源，如带宽等。在数据传输过程中，不同设备之间会竞争网络资源。为了避免时间敏感流与非时间敏感流之间的竞争，IEEE 802.1Qbv 协议提出了基于时间同步的 TAS 整形器，该整形器根据门控列表，周期性控制交换机队列门的开关。只有当对应门打开时，队列中的数据才可以被传输。该协议的基本思想是利用时分多址机制，将传输信道划分为多个时隙，保障特定优先级的数据帧在预定时间段的传输。

本文假定交换机采用输出缓存架构[47]，对于支持 IEEE 802.1Qbv 协议的交换机，数据帧从入端口进入，经过解包封包等一系列流程后，进入输出端口队列等待发送。在输出端口中，根据报文 VLAN 标签中的 PCP 字段，将报文传输至不同的队列进行排队。门控调度机制实现方式如图 2.3 所示。在T_0时刻，交换机打开 7 号队列门，关闭 0~6 号队列门，保障 7 号队列中的数据传输不受干扰。当T_1在时刻时，交换机关闭 7 号队列门，打开 0~6 号队列门。当有多个门同时打开时，传输选择模块会根据不同配置选择优先传输的队列。

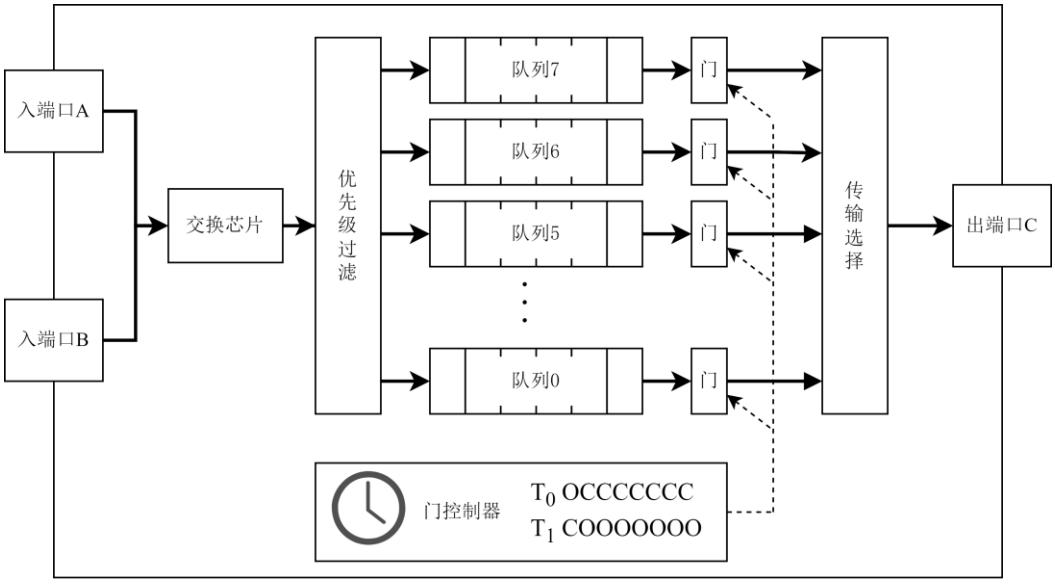


图 2.3 门控调度实现方式

在传统以太网中，数据传输是无法被抢占的，即当一个数据帧开始传输后，只有当该数据帧完全被传输后，才可以传输其他数据帧。如果一个数据帧较大，在预定的时隙中无法完成传输，则会阻碍下一个时隙中数据的传输。针对此问题，IEEE 802.1Qbv 协议提出了保护带的概念，即在保护带期间关闭所有门，不能进行新的数据帧传输，而正在传输的数据帧可以继续传输。保护带通常设置在时间敏感流时隙前，保障时间敏感流的传输。合理大小的保护带可以在保障时间敏感流传输的同时，提高带宽利用率[48]。通常，保护带的大小设置为最大数据帧传输时间。图 2.4 为带有保护带的周期数据调度示意图。

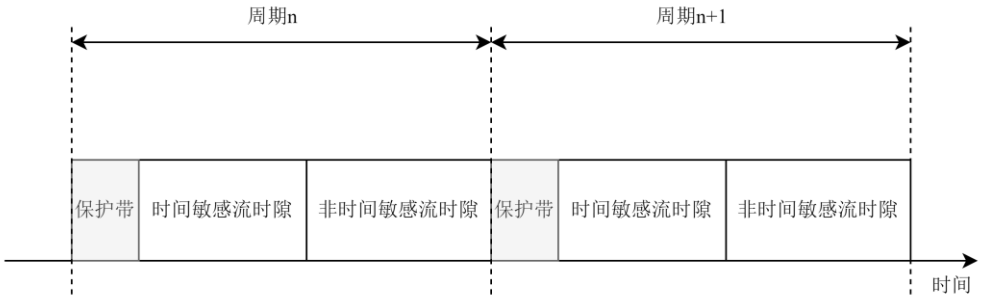


图 2.4 带有保护带的周期数据调度示意图

2.1.4 IEEE 802.1Qcc-2018 流预留协议与系统配置协议

IEEE 802.1Qcc 协议主要用于 TSN 网络的配置，它通过对发送端、接收端和 TSN 交换机的配置，来实现对全局网络的配置。该协议主要定义了三种网络架构：全分布式模型、集中式网络/分布式用户模型和全集中式模型。全集中式模型具有配置简单高效、统一管理以及便于维护等特点，在工业自动化领域受到广泛的关注。全集中式模型实现方式由集中式用户配置（Centralized User Configuration, CUC）、集中式网络配置（Centralized Network Configuration, CNC）、网络设备以及终端节点组成。图 2.5 为全集中式模型实现方式。首先 CNC 通过网络管理协议获取网络中设备、拓扑等信息，终端通过用户配置协议向 CUC 发送资源配置请求。其次，CUC 通过用户网络接口（User Network Interface, UNI）,向 CNC 发送资源配置需求。然后 CNC 按照终端用户需求和网络设备信息，生成网络调度方案，并通过网络管理协议控制网络中的设备按照设定的方案运行。最后，CNC 向 CUC 发送网络调度方案，CUC 通过用户配置协议配置终端按照指定方案运行。值得一提的是，IEEE 802.1Qcc 协议全集中式架构仅定义了配置的实现方式，但没有具体规定使用哪种协议，用户可以根据实际情况选择合适的协议来实现配置和管理功能[49][50]。

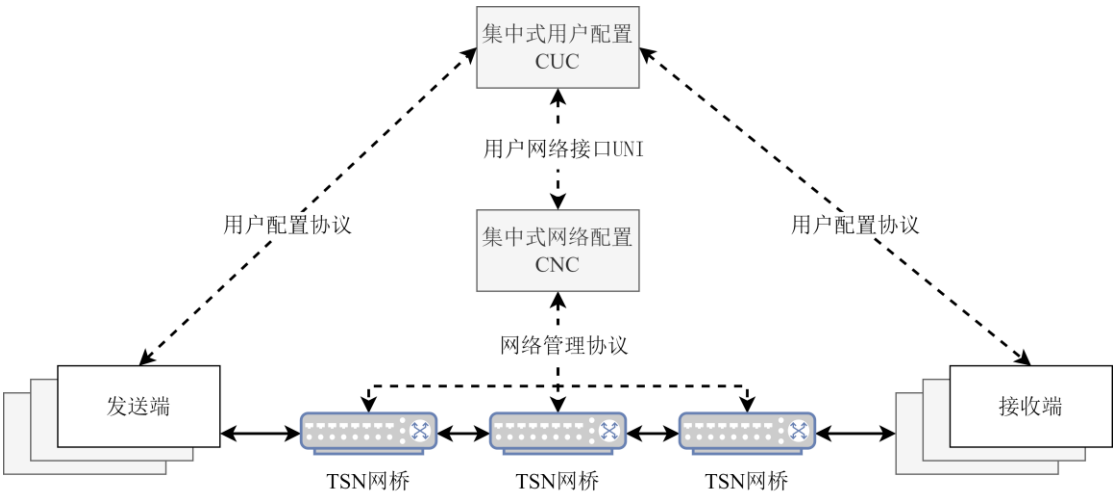


图 2.5 全集中式模型实现方式

2.2 TSN 网络仿真工具

网络仿真是研究网络性能的重要方法之一，不同的仿真方法在网络性能研究中具有不同的特点。因此，在进行 TSN 网络仿真时，选择合适的仿真方法与工具显得尤为重要[51]。在本小节中，首先对几种常见的网络仿真方法以及工具进行介绍，然后对不同 TSN 仿真框架及其特点进行介绍，为后文中搭建 TSN 网络仿真环境提供基础。

2.2.1 网络仿真工具

网络仿真是指使用计算机技术对网络进行模拟，以评估网络的性能、可靠性和安全性等。为了模拟网络系统的各种特性，网络仿真方法应该综合考虑多个因素，包括网络拓扑结构、网络协议、流量模型和传输媒介等。按照仿真环境的不同，可以分为真实物理系统、软件仿真系统和半实物仿真系统。

真实物理系统是指使用真实的网络设备和通信链路构建网络，并对其进行研究分析。其优势是精度高，结果可信度高。但由于其依赖真实的网络设备，因此成本较高，可扩展性差，并且不适用于新技术的研究和评估。

软件仿真系统是指使用计算机软件模拟物理环境以及网络设备的行为，并对其进行研究分析。其优势是仿真效率高，成本低，可扩展性强，可以快速实现新技术的研究和评估。同时，由于不受物理环境影响，仿真结果具有确定性。但其仿真精度与可靠性受仿真软件以及仿真模型共同影响。

半实物仿真系统是一种介于真实物理系统和软件仿真系统之间的系统，它利用仿真软件和真实硬件之间的接口，将物理硬件设备与仿真环境相结合，来模拟实际的网络环境。这种系统结合了真实物理系统和软件仿真系统的优势，但需要仿真软件以实时运行，对仿真软件以及物理硬件设备有一定的要求。

不同的网络仿真方法具有不同的特点，总结如表 2.4 所示。需要根据 TSN 的特点以及仿真验证的目的，选择合适的仿真方法。目前，部分 TSN 协议处于草案阶段，同时 TSN 交换机对流调度、流过滤等相关功能还没有统一的实现方式。因此，目前真实物理系统适用于相对成熟的 TSN 技术的落地验证，而软件仿真更适用于前沿技术的前期验证。

针对 TSN 的软件仿真，目前主流的仿真方法是离散事件仿真，即将系统行为分解为一系列事件，通过模拟这些事件之间的交互，来模拟整个系统的行为。典型的 TSN 仿真软件有 OPNET[52]、OMNeT++[53]以及 NS-3[54]。

表 2.4 网络仿真方法及其特点

	真实物理系统	软件仿真系统	半实物仿真系统
结果可靠性	高	低	中
灵活性	低	高	中
可扩展性	低	高	中
成本	高	低	中
仿真精度	高	低	中
仿真效率	低	高	中
新技术评估难度	高	低	中

OPNET 是一款用于网络建模、仿真和性能评估的商业软件。该软件采用分层建模机制，分为网络拓扑层、节点层和进程层，分别对应实际拓扑、设备和网络协议，完全反映了网络的相关特征。OPNET 被广泛应用于各种网络的性能仿真。Pahlevan 等人[55]利用 OPNET 对 TSN 中的 IEEE 802.1Qbv 和 802.1Qci 流控制机制进行了仿真，并分析了其时延特性。

OMNeT++是一个可扩展的、模块化的、基于组件的开源仿真软件，可用于网络建模、仿真和分析。其采用 C++语言编写，可运行在多种操作系统上。其提供了多种网络仿真和分析的工具和插件，如可视化器、调试器和结果分析器等，可对仿真结果进行细致的分析和处理。Simon 等人[56]利用 OMNeT++仿真软件研究 TSN 流量调度机制在工业自动化和移动前向网络中的应用。Heise 等人[57]提出了一种基于 OMNeT++的 TSN 仿真框架，对 TSN 中的帧复制与消除机制以及 TSN 帧抢占机制进行仿真。

NS-3 是一个用于网络模拟的开源仿真软件，支持大量网络协议和技术的建模和仿真，包括无线网络，移动网络，卫星网络，蜂窝网络，数据中心网络，以太网和 IP 协议。NS-3 基于 C++语言编写，并提供了 Python 接口以便于用户的使用和扩展。Krummacker 等人[58]利用 NS-3 对 TSN 中的 IEEE 802.1Qbv 机制进行了仿真。

OPNET、OMNeT++以及 NS-3 具有各自独特的特点和优势，总结如表 2.5 所示。其中，OPNET 对于复杂的商业应用场景具有更好的适用性，但对 TSN 支持程度相对较低，同时该软件不开源，可扩展性较差。虽然 OMNeT++和 NS-3 都是开源软件，但在 NS-3 中针对 TSN 的仿真框架较少，不利于进行 TSN 仿真。在 OMNeT++中，支持多个不同的 TSN 仿真框架，可以快速建立 TSN 仿真网络。同时，在 OMNeT++中提供了特定接口实现半实物仿真，可以对 TSN 网络进行更贴近真实的仿真。综上所述，在本文中选择 OMNeT++仿真软件进行 TSN 网络仿真。

表 2.5 网络仿真软件及其特点

	OPNET	OMNeT++	NS-3
性质	商业软件	开源软件	开源软件
语言	C++	C++	C++ 、Python 等
平台	Windows	Windows，Linux 等	Windows，Linux 等
GUI	是	是	是
分析工具	是	是	是
计算效率	高	高	低
仿真框架数量	少	多	中
支持协议数量	少	多	中
支持 TSN 程度	中	高	低

2.2.2 TSN 仿真框架

OMNeT++作为网络仿真软件仅提供了网络仿真架构，并没有提供具体网络协议或网络设备的实现方式，因此需要配合特定仿真框架使用。

CoRE4INET[59]框架由德国汉堡的大学的 CoRE 研究小组于 2011 年提出，该框架在 INET 3.6.6 版本的基础上，提供了包括 TSN，AVB 以及 AS6802 等多种网络的仿真。同时随着多年的发展，在该框架的基础上，衍生出如 SDN4CoRE[62]等仿真框架，为车载以太网、软件定义时间敏感网络的仿真提供重要支撑。但由于该框架提出时间较早，且依赖的 OMNeT++和 INET 版本兼容性较差，在仿真过程中容易产生未知错误。

NeSTiNg[60]框架由德国斯图加特大学 IPVS 研究小组于 2019 年提出，该框架在 INET 4.1.2 版本的基础上，提供了 TSN 网络的仿真。通过对数据链路层以及物理层的重构，实现了支持 TAS 功能的以太网网卡和可抢占全双工数据链路等，实现 TSN 中的 TAS、帧抢占等机制。

INET[61]框架是一个基于网络分层思想的网络仿真框架，该框架提供了大量网络协议和网络设备的实现，包括以太网、TCP/IP 等协议，以及交换机、路由器等网络设备。以该框架为基础，已经衍生出多款支持 TSN 的仿真框架，包括 CoRE4INET 和 NeSTiNg。在 2023 年推出的 INET 4.5 版本中，集成了大量 TSN 协议，包括 IEEE 802.1Qbv，IEEE 802.1Qci 等。同时，该框架支持多种 TSN 场景的仿真，如网络故障场景等。但目前该框架仍处于开发状态，尚未达到成熟阶段。

表 2.6 比较了 3 种常见的 TSN 仿真框架。其中 NeSTiNg 框架成熟度较高，且兼容性好，

适合在其基础上进行二次开发，因此在本文中选用该框架进行 TSN 网络仿真。

表 2.6 TSN 仿真框架对比

	CoRE4INET	NeSTiNg	INET 4.5
推出时间	2011 年	2019 年	2023 年
依赖 OMNeT++版本	5.5.1	5.5.1	6.0
依赖 INET 版本	3.6.6	4.1.2	/
支持协议	IEEE 802.1Q IEEE 802.1Qbv IEEE 802.1Qci 等	IEEE 802.1Q IEEE 802.1Qbv IEEE 802.1Qbu 等	IEEE 802.1Q IEEE 802.1AS IEEE 802.1Qbv IEEE 802.1Qbu 等
优势	支持车载以太网仿真	成熟度高	支持功能多，兼容性好
劣势	兼容性差	功能相对较少	成熟度低

在 NeSTiNg 框架中，提供了三种不同类型的 TSN 设备模块，分别为 VlanEtherHostQ、VlanEtherHostSched 以及 VlanEtherSwitchPreemptable。其中 VlanEtherHostQ 模块为定时发送终端，该模块可以创建和接收带有 VLAN 标签的以太网流量，主要充当接收端或 BE 流的发送端；VlanEtherHostSched 模块为可调度定时发送终端，该模块可按照给定的调度表，定时发送带有 VLAN 标签的以太网流量，主要充当 CT 流的发送端；VlanEtherSwitchPreemptable 模块为支持帧抢占功能的 TSN 交换机。

表 2.7 展示了 NeSTiNg 框架中三种不同 TSN 设备模块内部结构。其中，在数据链路层，三种模块均采用 TSN 的网卡 NestingEthernetInterface 模块。该模块支持出入口流量的管控、数据帧的封装与解封装、VLAN 标签的封装与解封装以及队列模块。在该模块的队列子模块中，通过增加门控功能，实现 TAS 机制。

表 2.7 NeSTiNg 框架中三种不同 TSN 设备内部主要子模块

	定时发送终端 VlanEtherHostQ	可调度定时发送终端 VlanEtherHostSched	支持抢占的 TSN 交换机 VlanEtherSwitchPreemptable
应用层	定时发送程序 VlanEtherTrafGen	可调度定时发送程序 VlanEtherTrafGenSched	/
数据链路层	/	/	基础转发模块 ForwardingRelayUnit
	TSN 网卡模块 NestingEthernetInterface		
物理层	帧抢占全双工链路 EtherMACFullDuplexPreemptable		

假设一条流从 VlanEtherHostSched 模块发出，经过 VlanEtherSwitchPreemptable 模块，到达 VlanEtherHostQ 模块，则该流主要经历以下过程。首先，发送端应用层程序按照特定调度表生成数据帧，数据传输至 NestingEthernetInterface 模块，在此模块中进行封装和添加 VLAN 标签后，传输至仿真物理层。数据帧到达交换机入端口，经过解封包后，根据目地地址查找并转发至出端口。在出端口中，根据携带的 PCP 信息传输至指定队列排队。当该队列门打开且没有更高优先级的数据帧传输时，发送该数据帧。数据帧到达接收端，经过解封包后，传输至接收端应用层程序。

2.3 本章小结

在本章中，首先对本文所涉及到的 TSN 协议进行简单的介绍，并分析其具体实现方式。然后对本文中所涉及的 TSN 仿真工具以及仿真框架进行介绍，最后根据 TSN 特点选择合适的仿真工具与仿真框架，为后文提供基础。

第三章 TSN 半实物仿真系统

本章基于 OMNeT++仿真软件设计并实现 TSN 半实物仿真系统，并通过对比实验，对半实物仿真接口模块进行分析验证。在此基础上，通过与真实物理系统以及软件仿真系统的实验结果进行对比，验证了本文所设计半实物仿真系统的性能。

3.1 引言

TSN 网络在部署前，需要对网络性能进行验证以确保可靠性。然而在真实的 TSN 网络中模拟各种可能遇到的情况是很困难的。同时，目前部分 TSN 协议仍处于草案阶段，并且对协议的具体实现方式还在进行讨论。不同的设备厂商对协议的实现方式也有所不同。若使用 TSN 芯片或 FPGA 等方式进行验证，则开发难度大，经济效益低，不利于新技术的前期验证。

为了解决以上问题，可以利用半实物仿真对 TSN 进行测试验证。半实物仿真是一种将真实硬件和仿真软件相结合的仿真方法，其中真实硬件受仿真软件控制，并与仿真软件所提供的仿真环境紧密结合，从而提供更真实准确的仿真测试结果。这种仿真方法可用于评估网络设备的功能和性能，也可用于测试网络协议或应用程序的效果。目前，支持 TSN 半实物仿真的仿真软件相对较少，其中典型代表包括 OpenTSN 团队[29]、NXP 公司以及 TTTech 公司推出的仿真软件。然而，这些仿真软件需要配合特定的网络设备才可以实现 TSN 半实物仿真。同时，部分软件并未开源，可扩展性差，不利于新技术的验证。因此，本文希望在独立于网络设备的开源软件上，实现 TSN 的半实物仿真。

OMNeT++是一款开源的网络仿真软件，通过特定的仿真框架，可以实现 TSN 的软件仿真。与此同时，该软件也支持半实物仿真。Böhm 等人[63]利用 OMNeT++仿真软件，配合 RoSeNet 仿真软件，实现了无线网络的半实物仿真。Nirav 等人[64]提出了一种适用于 OMNeT++仿真软件动态路由框架，可以实现半实物仿真网络的动态路由。汤淼[65]和李自州[66]利用该软件，实现了移动自组网的半实物仿真。Kölsch 等人[67]对 OMNeT++仿真软件半实物仿真实现机制进行了分析。然而，目前并没有支持 TSN 半实物仿真的框架，因此需要对现有的软件以及框架进行修改。

在本章中，首先将 TSN 半实物仿真的实现分为实时调度机制部分与半实物仿真接口部分，并分别进行实现。其次，根据半实物仿真类型的不同，设计并实现了两种不同的半实物仿真接口。然后，设计对比实验，对两种不同的半实物仿真接口进行分析验证。最后，分别搭建软件仿真实验系统、真实物理实验系统以及半实物仿真系统。通过这三者实验结果的对比分

析，验证半实物仿真系统的性能。

3.2 TSN 半实物仿真实现

目前，NeSTiNg 框架本身并不支持 TSN 半实物仿真，因此需要对该框架中的相应模块进行修改。在半实物仿真过程中，真实事件需要与仿真事件相互映射，同时映射后的仿真事件的执行顺序需要与真实事件的执行顺序一致。因此，首先需要实现实时调度机制，控制仿真网络按照真实时间运行。其次需要开发特定的半实物仿真接口，实现真实事件到仿真事件的映射。

3.2.1 实时调度机制实现

OMNeT++仿真软件中，通过 cScheduler 调度器实现事件的调度和处理。在仿真的初始化阶段，程序会创建一个 cScheduler 类的实例，它通过维护和管理一个未来事件队列（Future Event Set, FES），将仿真环境中的事件按照其执行时间，依次传递到相应模块的 handleMessage 函数中，以进行进一步的处理。此外，cScheduler 还负责控制事件处理程序的执行时间和保证仿真的连续性。该过程被封装在 OMNeT++仿真软件中，对用户来说是透明的，即用户无法直接干预 cScheduler 类实例的调度过程。

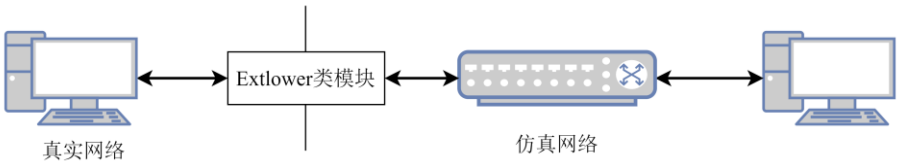
软件仿真过程中采用 cScheduler 的顺序调度器子类 cSequentialScheduler 进行调度，该调度器根据 CPU 时间计算当前仿真时间。通常来说，仿真环境中的时间尺度远大于真实时间尺度，即在仿真环境中经过的 1s 时间远大于真实环境中经过的 1s 时间。但对于半实物仿真，由于仿真事件需要与真实事件进行交互，同时仿真事件的执行顺序需要与真实事件的执行顺序一致。因此需要仿真时间与真实时间一致。然而，在半实物仿真的过程中，由于物理设备和仿真系统的性能不同，仿真时间可能会与真实时间不一致，这将导致仿真网络中出现多个不同的仿真时间值，进而直接导致仿真系统出现错误。为了保证仿真的准确性和可靠性，必须采取一系列措施来确保仿真时间与真实时间一致。在 OMNeT++仿真软件中，提供了 cScheduler 的实时调度器子类 cRealTimeScheduler，该调度器负责将仿真网络中的时间与操作系统时间进行同步，使仿真网络实时运行。

在仿真网络的初始化阶段，实时调度器获取当前操作系统时间，作为仿真网络的基准时间t_{base}，同时初始化仿真时间$t_{sim} = 0$。在仿真网络运行过程中，实时调度器获取当前操作系统时间t_{now}，判断当前真实时间与仿真时间关系。若仿真时间快于真实时间，则按照当前操作系统时间和基准时间更新仿真时间。若仿真时间慢于真实时间，则调度器通过禁用休眠调

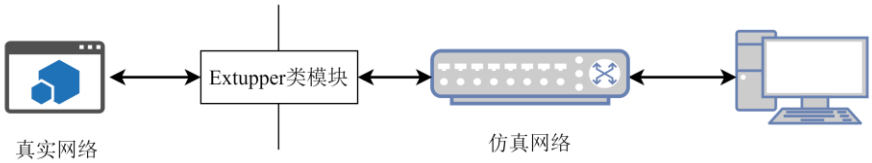
用等方式尝试重新时间同步。之后调度器根据仿真时间，将 FES 中相应事件传递到应模块的 handleMessage 函数中进行处理。并计算处理所需的 CPU 时间t_{proc}，增加仿真时间。在仿真运行的过程中，每次事件调用前，都会调用实时调度器，以保障仿真时间与真实时间的同步。时间同步的精度受到事件执行所需的 CPU 时间影响。对于一个事件，如果其在仿真网络中被处理所需时间超出了其在真实网络中所需时间，则仿真时间会落后于真实时间，虽然仿真软件会尝试通过忽略睡眠调用等方式追赶，但随着仿真规模、仿真时间、仿真事件的增加，仿真时间与真实时间的误差将逐渐增大，进而影响半实物仿真精度。可以通过减少仿真事件数量，提高 CPU 性能等方式，提高半实物仿真精度[68]。

3.2.2 半实物仿真接口实现

半实物仿真过程中真实事件与仿真事件需要通过半实物仿真接口实现相互映射。按照半实物仿真类型的不同，可以将半实物仿真接口分为真实设备的半实物仿真接口和真实程序的半实物仿真接口，如图 3.1 所示。其中真实设备的半实物仿真接口可以将独立且完整的数据帧传输至仿真网络，而真实程序的半实物仿真接口可以将应用层软件发出的数据包传输至仿真网络，在仿真网络中完成网络层、数据链路层封装等操作之后进行传输。将这两种方式所涉及的半实物仿真接口分别命名为 Extlower 类模块和 Extupper 类模块。



(a) 真实设备的半实物仿真



(b) 真实程序的半实物仿真

图 3.1 半实物仿真分类

在 NeSTiNg 框架中提供了三种设备模块，分别为定时发送终端 VlanEtherHostQ、可调度定时发送终端 VlanEtherHostSched 以及支持抢占的 TSN 交换机 VlanEtherSwitchPreemptable。虽然其内部结构各不相同，但在数据链路层均采用 TSN 网卡模块 NestingEthernetInterface。图 3.2 为 TSN 网卡模块内部结构图，该模块负责数据包的封装、排队等功能，同时该模块通过

在 Queuing 子模块中增加门控机制，实现 TSN 中的 TAS 机制。将其中的 mac 子模块划分为 lowerLayer，剩余子模块划分为 upperLayer。可以分别针对 upperLayer 和 lowerLayer 进行改造，使其可以接收真实环境中的流量，并可以将其转换为仿真环境中的仿真事件。在本文中，将基于 NeSTiNg 框架开发的 Extlower 类网卡模块命名为 NestingExtlowerNIC 模块，将开发的 Extupper 类网卡模块命名为 NestingExtupperNIC。通过这两种不同模块，可以实现两种不同类型的半实物仿真。

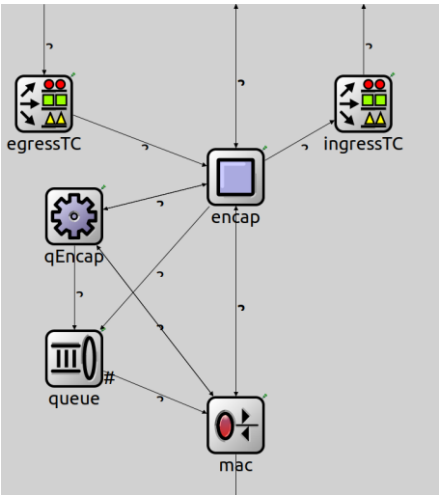


图 3.2 NestingEthernetInterface 模块内部结构

（A）NestingExtlowerNIC 模块实现

NestingExtlowerNIC 模块可以将真实的设备和仿真的网络进行连接。在仿真的过程，该模块需要不断捕获网络中需要传输至仿真网络的数据帧，将其反序列化为仿真事件并送至 FES。同时需要将仿真网络中需要传输至真实网络的数据帧，序列化为真实的报文并传输至真实网络。在 Linux 系统中，可以利用 libpcap 库[69]实现数据帧的捕获与重放。当网络中收到一个数据帧后，该库通过数据链路层驱动程序对数据帧进行复制，并送至伯克利包过滤器（Berkeley Packet Filter, BPF）。若数据帧满足过滤条件，则进入缓冲区。默认情况下，当缓冲区满时，会触发应用层程序读取数据。这种方式的优势是丢包率低，CPU 占用率，但会导致数据实时性不足，影响半实物仿真精度[70]。为了提高半实物仿真精度，本文采用 pcap_set_immediate_mode 模式。在该模式下，程序捕获到数据帧后会立即触发应用层程序读取。这种方式虽然带来了一定的丢包风险，但提高了数据的实时性。同时，可以利用 Linux 的 namespace 机制[71]，将目标网口与其他网口进行划分，减少干扰。

NestingExtlowerNIC 模块采用 Socket 通信的方式，将应用层捕获的 PCAP 文件传输至仿真环境。图 3.3 为 NestingExtlowerNIC 模块将真实流量映射至仿真环境过程。在仿真初始化阶段，仿真网络与真实网络通过 Socket 建立双向连接。在仿真网络运行过程中，若收到应用

层捕获到的数据帧，则调度器会中断当前事件，将数据帧封装在 ExtFrame 事件中。然后使用当前仿真时间作为该事件的执行时间，并将其传输至 FES 等待处理。由于 ExtFrame 事件的执行时间是当前仿真时间，因此该事件会立刻被调度器调用执行。调度器将其传递至 NestingExtlowerNIC 模块，在该模块中数据帧被反序列化为仿真数据帧 EtherFrame，并传输至 FES 中等待调度。至此，实现真实系统到仿真系统的映射。仿真系统到真实系统的映射与之类似。

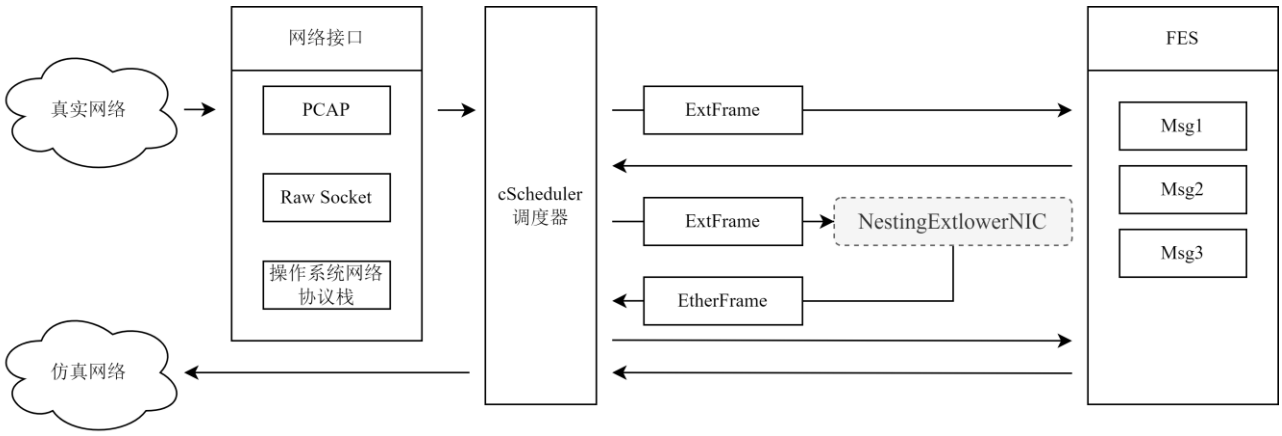


图 3.3 NestingExtlowerNIC 模块将真实流量映射至仿真环境过程

（B）NestingExtupperNIC 模块实现

NestingExtupperNIC 模块可以将真实的程序和仿真的网络进行连接。即该模块需要将收到的真实应用层数据帧，转换为仿真应用层数据帧，并对数据帧进行封装之后，传输至仿真网络环境。同时需要将收到的仿真应用层数据帧转换为真实应用层数据帧并送至目标程序。在 Linux 系统中，可以通过虚拟 TAP 设备，使真实应用层程序和 NestingExtupperNIC 模块通过读写同一个 TAP 接口中的内容，实现真实环境与仿真环境的连接。

图 3.4 为 NestingExtupperNIC 模块将真实流量映射至仿真环境过程。在仿真初始化阶段，仿真程序与指定的真实 TAP 设备建立双向连接。在仿真网络运行过程中，与 NestingExtlowerNIC 模块处理过程类似，收到的真实数据帧将被转换成 ExtFrame 事件，最终转换至仿真数据帧 EtherFrame。与之不同的是，在该模块中，包含一个队列模块和一个 MAC 模块，仿真数据帧将会进入队列排队，最终进入 MAC 模块中转换为仿真数据流。值得一提的是，在 NeSTiNg 框架中，MAC 层采用支持帧抢占的全双工链路，但在真实网络中暂时不支持帧抢占功能，因此需要将 MAC 层模块替换为普通全双工链路。

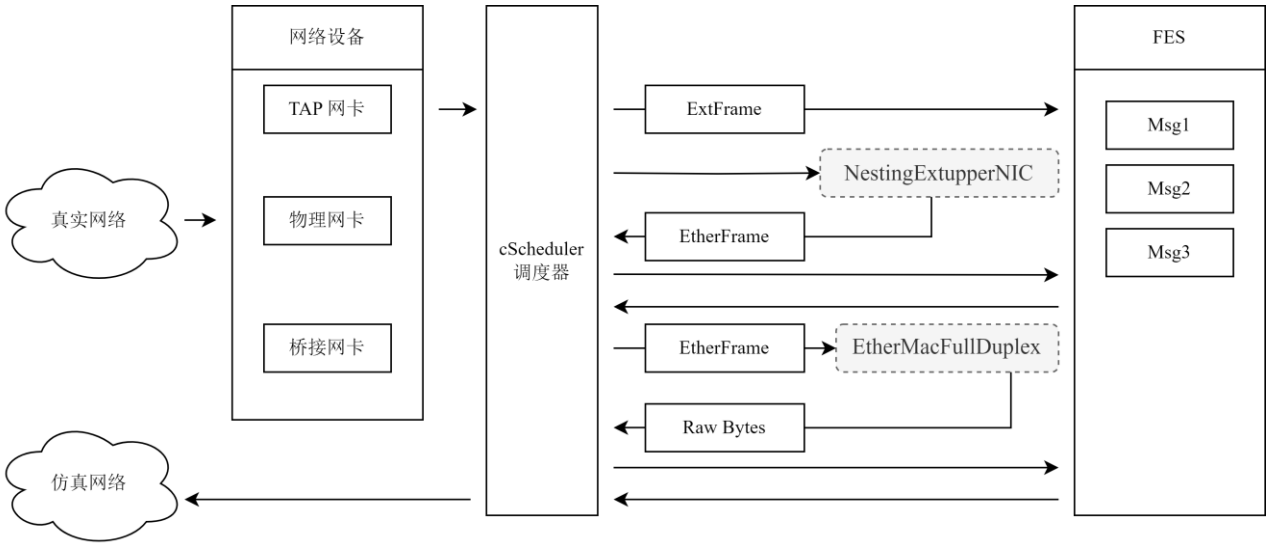


图 3.4 NestingExtupperNIC 模块将真实流量映射至仿真环境过程

3.3 半实物仿真接口模块可行性验证

在本小节中，将利用 Linux 操作系统中的虚拟网络接口，对本文所提出的半实物仿真接口进行可行性验证。

3.3.1 实验设置

图 3.5 为半实物仿真实验环境，其中 CPU 采用 10 核心 16 线程的 Intel i5-12600KF，内存为 16GB，操作系统采用 Ubuntu 18.04。仿真软件选择 OMNeT++ 5.6.1，INET 版本为 4.1.2，GCC 版本为 7.5.0。在 OMNeT++仿真软件中，默认关闭半实物仿真功能，需要在仿真软件设置中打开该功能，并重新编译 INET 框架和 NeSTiNg 框架。同时，由于半实物仿真需要对操作系统内核中的网卡接口进行修改，因此需要使用 setcap 指令为仿真程序赋予获取网络原始套接字和网络管理的权限，使其可以监听和修改网络数据帧，并进行网络配置等。

图 3.5 半实物仿真实验环境

3.3.2 NestingExtlowerNIC 模块可行性验证

首先在 Linux 系统中利用 namespace 机制创建两个独立的内核空间，并利用 IP 指令创建一组虚拟网络设备 Veth0 和 Veth1。将两个虚拟网络设备分别划分到两个独立的内核空间，并指定其 IP 地址、子网掩码和路由等信息。指定发往 Veth0 设备的流量路由转发至 Veth1 设备，图 3.6 为网络中的虚拟网络设备信息。

```
root@ubuntu:/home/tsn3# ip netns exec net0 ifconfig
veth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.1  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::44af:e0ff:fe31:18db  prefixlen 64  scopeid 0x20<link>
        ether 46:af:e0:31:18:db  txqueuelen 1000  (Ethernet)
        RX packets 10  bytes 796 (796.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/tsn3# ip netns exec net1 ifconfig
veth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.2  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::2077:2dff:fece:5aac  prefixlen 64  scopeid 0x20<link>
        ether 22:77:2d:ce:5a:ac  txqueuelen 1000  (Ethernet)
        RX packets 10  bytes 796 (796.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 3.6 创建虚拟网卡网络设备

利用 Ostinato 软件生成数据包，并向 Veth0 设备发送数据。Ostinato 是一款开源的、跨平台的网络数据包和流量生成器和分析器，支持图形化界面操作。在 OMNeT++仿真软件中，通过 ini 文件指定调度器的类型并设置 NestingExtlowerNIC 模块监听 Veth1 设备。利用 Wireshark 软件捕获网络中数据包信息。当仿真软件运行前，Ostinato 软件生成的数据包无法转发至仿真网络环境，数据包将被丢弃。当仿真软件运行后，Ostinato 软件生成的数据包将被仿真网络环境中的仿真网络设备接收。图 3.7 为 Wireshark 捕获 NestingExtlowerNIC 模块收到的数据包，证明 NestingExtlowerNIC 模块运行正常，可以完成数据收发。

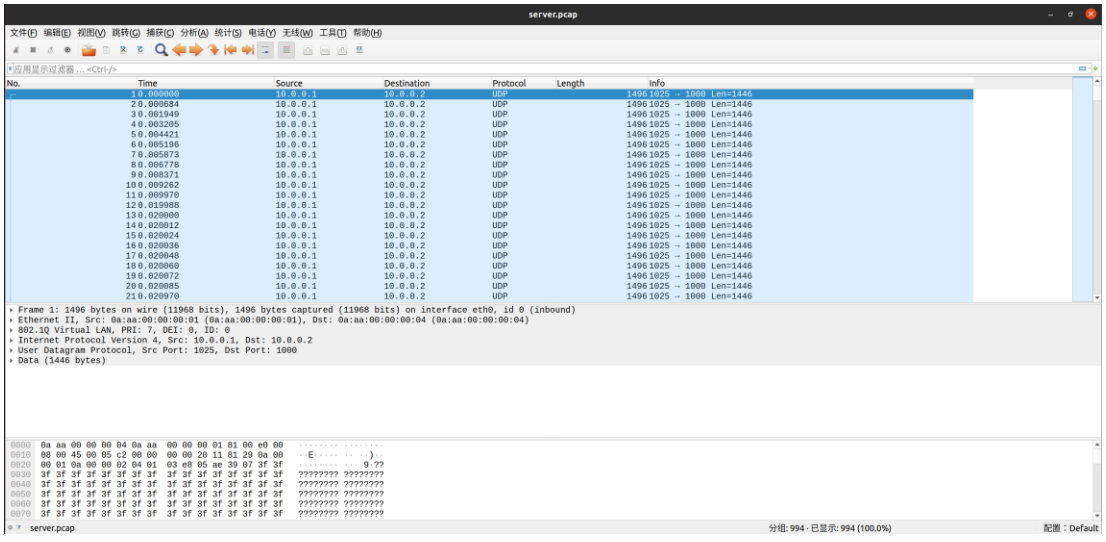


图 3.7 Wireshark 捕获 NestingExtlowerNIC 模块收到的数据包

3.3.3 NestingExtupperNIC 模块可行性验证

首先在 Linux 系统中利用 IP 指令创建虚拟 TAP 设备，并指定其 IP 地址、子网掩码和路由等信息，图 3.8 为网络中的虚拟网络设备信息。

```
root@ubuntu:/home/tsn3# ifconfig
ens33: flags=4419<UP,BROADCAST,RUNNING,PROMISC,MULTICAST>  mtu 1500
        inet 192.168.15.129  netmask 255.255.255.0  broadcast 192.168.15.255
        inet6 fe80::20c:29ff:fea8:440d  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a8:44:0d  txqueuelen 1000  (Ethernet)
        RX packets 262553  bytes 375314147 (375.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27464  bytes 6033036 (6.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=329<UP,LOOPBACK,RUNNING,PROMISC>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 24350  bytes 7964764 (7.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24350  bytes 7964764 (7.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tapa: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.2.20  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 06:6d:51:67:a8:32  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 3.8 创建虚拟 TAP 网络设备

利用 Ostinato 软件生成数据包并向指定 TAP 设备发送数据。在 OMNeT++仿真软件中，通过 ini 文件指定 NestingExtupperNIC 模块监听对应 TAP 设备，利用 Wireshark 软件捕获网络中数据包信息。当仿真软件运行前，Ostinato 软件生成的数据包被传输至 TAP 设备，但由于没有应用程序读取对应内容，数据包将被丢弃。当仿真软件运行后，NestingExtupperNIC 模块监听 TAP 设备中的数据，将 TAP 设备中的流量转换为仿真流量，并传输至仿真网络设备。图 3.9 为 Wireshark 捕获 NestingExtupperNIC 模块收到的数据包，证明 NestingExtupperNIC 模块运行正常，可以完成数据收发。

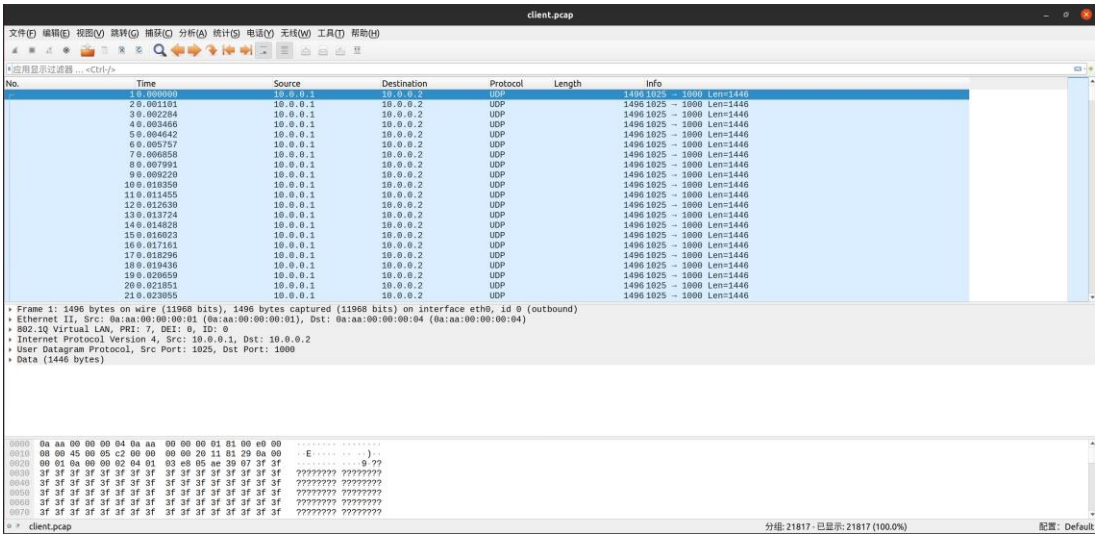


图 3.9 Wireshark 捕获 NestingExtupperNIC 模块收到的数据包

3.4 半实物仿真接口模块性能分析

在半实物仿真的过程中，真实系统中的事件会映射为仿真系统中的事件，在此过程中会引入一定的时延。为了尽可能精确的反映真实设备在仿真网络中的表现，需要尽可能小的时延。在本小节中，将通过真实系统到仿真系统映射过程中的时延，评估上文中开发的两种不同半实物仿真接口。

3.4.1 实验设置

采用图 3.10 所示的网络拓扑，其中PC_1为发送端，PC_2为接收端。发送端向接收端发送 PCP=7 的 UDP 数据包，数据帧大小为 1500B，发送间隔为 1ms。设置 UDP 数据帧中携带当前操作系统时间t_{now}^{i}，当真实数据帧映射为仿真数据帧后，记录当前仿真时间t_{sim}^{i}，其中i为数据帧序号。利用t_{now}^{i}与t_{sim}^{i}的差异评估真实系统到仿真系统映射过程中的时延。为了避免 TAS 机制对实验结果的影响，配置 TSN 交换机所有门一直打开。

图 3.10 半实物仿真接口模块实验网络拓扑

为了探究半实物仿真接口模块性能，本文设计了三组对比实验。第一组实验采用软件仿真系统，利用仿真软件生成 UDP 数据包；第二组实验采用真实设备的半实物仿真系统，利用 Ostinato 软件和虚拟网络设备组成真实发送端，TSN 交换机搭载 NestingExtlowerNIC 模块；第三组实验采用真实程序的半实物仿真系统，利用 Ostinato 软件充当发送端应用层程序，数据链路层采用 NestingExtupperNIC 模块。

第二组实验配置如图 3.11 所示，在 Linux 系统中利用 namespace 机制创建两个独立的内核空间，并利用 IP 指令创建一组虚拟网络设备 Veth0 和 Veth1。将两个虚拟设备分别划分到两个独立的内核空间，并指定其 IP 地址和路由信息。其中，利用 Ostinato 软件充当应用层发送程序，与 Veth0 共同构成真实设备，向 Veth1 发送 UDP 流量。当 Veth1 收到 UDP 报文后，触发 NestingExtlowerNIC 模块，该模块将真实流量映射为仿真流量。仿真流量经过 TSN 交换机的排队转发后，传输至仿真接收端。

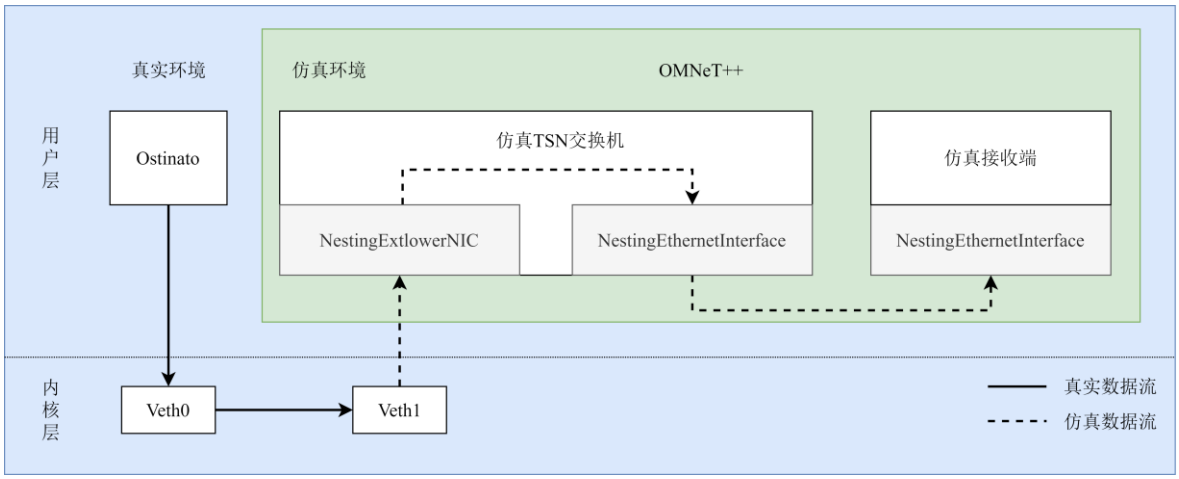


图 3.11 真实设备的半实物仿真实验配置

第三组实验配置如图 3.12 所示，在 Linux 系统中创建虚拟 TAP 设备，并指定其 IP 地址和路由信息。其中，Ostinato 软件和 NestingExtupperNIC 模块共同构成发送端。Ostinato 软件生成 UDP 数据帧后，写入 TAP 设备。当 TAP 设备收到信息后，触发 NestingExtupperNIC 模块读取，该模块将真实流量映射为仿真流量，并送至仿真 MAC 层进行传输。仿真流量经过 TSN 交换机后，传输至仿真接收端。

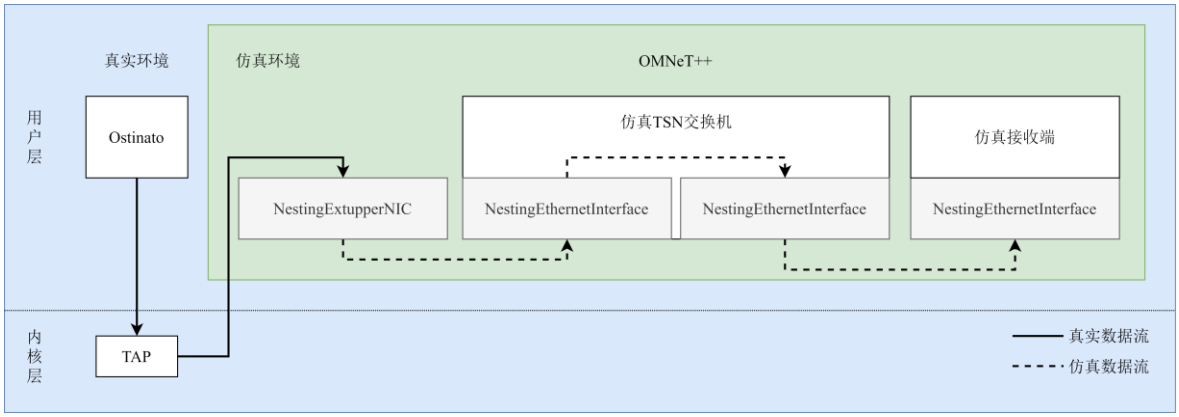
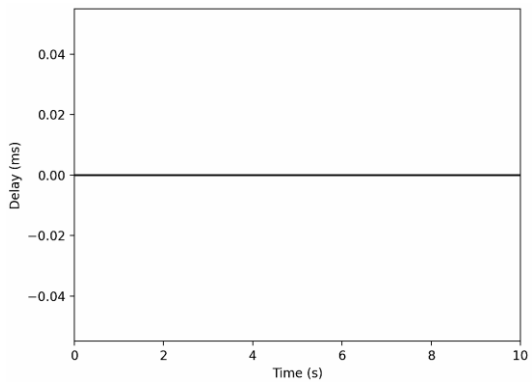


图 3.12 真实程序的半实物仿真实验配置

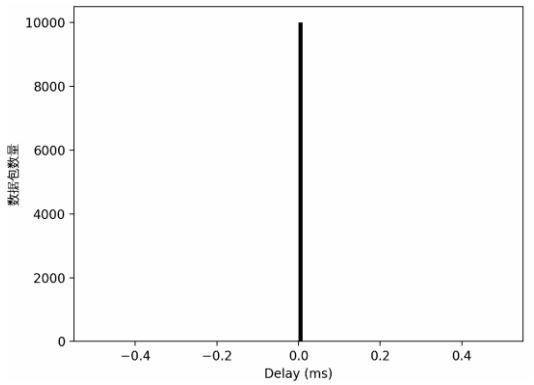
3.4.2 结果分析

图 3.13 为半实物仿真接口模块对比实验结果，其中图 a、c、e 为 10s 内流量映射过程中的时延统计，图 b、d、f 为时延分布情况。图 3.13(a)和图 3.13(b)为软件仿真环境中的实验结果，在软件仿真过程中，不需要将真实流量映射为仿真流量，因此所有时延等于 0。图 3.13(c)和 3.13(d)为真实设备半实物仿真环境中的时延情况。在此过程中，真实数据帧以 PCAP 文件的形式，通过 Socket 通信传输至仿真环境。由于捕获 PCAP 文件以及 Socket 通信会消耗一定的时间，因此时延在 0.05ms 到 0.25ms 之间波动。由于在实验中利用 namespace 将 veth1 与其

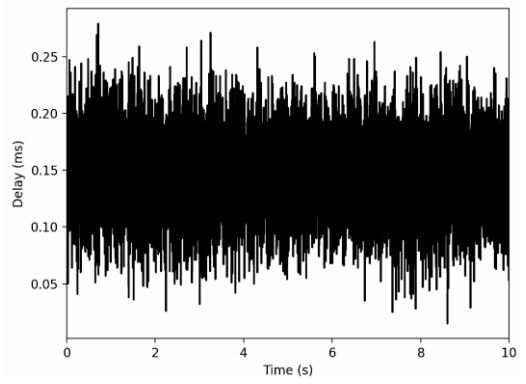
他接口进行隔离，因此时延主要集中在 0.15ms。图 3.13(e)和图 3.13(f)为真实程序半实物仿真环境中的时延情况。在此过程中，真实应用层程序向指定 TAP 设备发送数据，半实物仿真接口通过监听对应 TAP 设备中的信息，将指定 TAP 设备中的数据转换为仿真数据并送至仿真网络传输。由于读写 TAP 设备会消耗时间，因此时延在 0.05ms 到 0.2ms 之间波动。在实验过程中，没有其他软件向指定 TAP 接口发送，因此时延相对集中，主要集中在 0.13ms 附近。



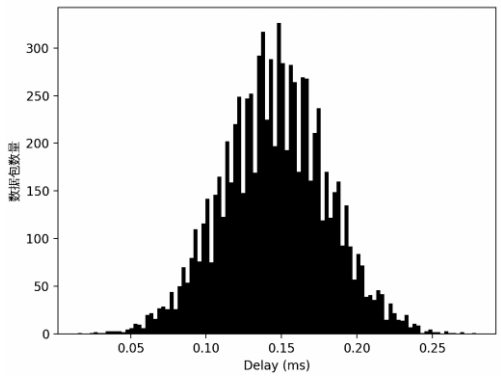
(a) 软件仿真环境时延统计



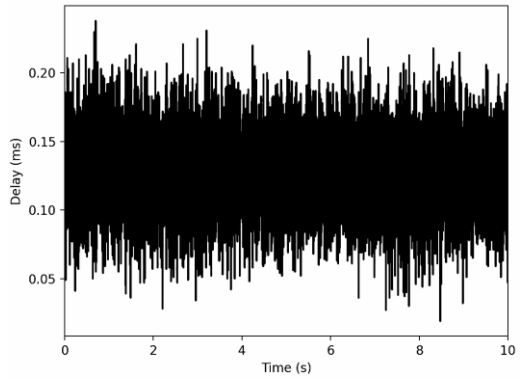
(b) 软件仿真时延分布



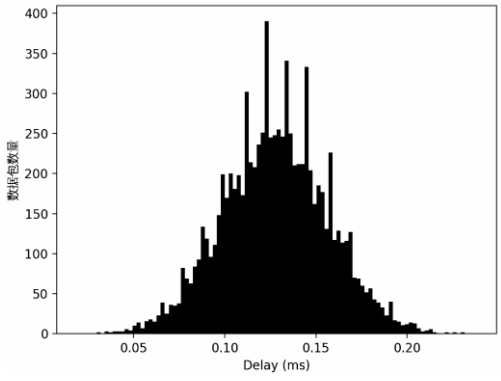
(c) 真实设备半实物仿真环境时延统计



(d) 真实设备半实物仿真环境时延分布



(e) 真实程序半实物仿真环境时延统计



(f) 真实程序半实物仿真环境时延分布

图 3.13 半实物仿真接口模块对比实验结果

表 3.1 为不同仿真环境下流量映射过程中的时延统计。对于软件仿真环境，数据流量由软件进行仿真，不需要进行真实流量与仿真流量之间的映射，因此所有时延等于 0。对于真实

设备的半实物仿真环境，由于需要通过 libpcap 工具捕获数据包，因此时延分布于 0.015ms 到 0.279ms，平均时延为 0.146ms。对于真实程序的半实物仿真环境，由于需要读写 TAP 设备，因此时延分布于 0.019ms 到 0.238ms，平均时延为 0.128ms。真实程序半实物仿真接口模块所引入的时延更加集中，标准差为 0.029ms。对比发现真实程序的半实物仿真接口在时延方面的表现优于真实设备的半实物仿真接口，其主要原因是真实设备的半实物仿真接口模块依赖于 socket 通信，实时性相对较差。与软件仿真相比，半实物仿真会引入一定量的时延。虽然这些时延相对较小，但是对时间敏感的 CT 流来说，这是不可忽略的。因此在半实物仿真过程中更适合使用真实流量充当对时间不敏感的 BE 流。同时真实程序的半实物仿真效果优于真实设备的半实物仿真，可以提供更贴近真实的仿真结果。

表 3.1 不同仿真环境下流量映射过程中时延统计（单位 ms）

	平均值	标准差	最大值	最小值
软件仿真环境	0	0	0	0
真实设备半实物仿真环境	0.146	0.035	0.279	0.015
真实程序半实物仿真环境	0.128	0.029	0.238	0.019

3.5 半实物仿真系统性能分析

在本小节中，将分别在物理实验系统、软件仿真系统以及半实物仿真系统中配置相同网络，对比不同环境下实验结果，验证本文所提出的半实物仿真系统性能。

3.5.1 物理实验系统搭建设置

图 3.14 为 TSN 物理实验环境，由两台 PC 和一台 TSN 交换机共同组成。其中两台 PC 采用相同配置，CPU 采用 Intel i7-6700K，网卡采用 Intel I210，操作系统采用 Debian 10.09，搭配 4.19.0-18-rt-amd64 实时性内核补丁。TSN 交换机采用 NXP LS1028ARDB。

图 3.14 TSN 物理实验环境

在 TSN 中，为了实现 TAS 机制，通常有两种方式可供选择。第一种方式是仅在 TSN 交换机上配置 TAS。尽管这种方式可以确保 CT 流和 BE 流在时域上互不干扰，但是 CT 流可能需要在交换机上排队等待，直到对应门打开时才能发送。这种方式会导致 CT 流时延较高，确定性差。第二种方式是在终端和交换机上都配置 TAS，这样 CT 流就可以在指定的时间发送，而交换机也可以在指定的时间开门，从而提高 CT 流的确定性。在本文中采用第二种方式进行配置。

利用开源的实时 OPC UA Pub/sub 程序[72]充当周期性时间敏感流的通信程序，网卡使用支持 TSN 功能的 Intel I210 网卡。在数据传输的过程中，需要涉及到 ISO 七层网络，为了提高数据的实时性，在终端中实现 TAS 机制，需要在用户层通信程序，操作系统内核以及网络协议栈中进行一定的配置。在 Linux 系统中，利用实时性内核补丁提高内核实时性。该补丁通过最大化内核中可抢占部分，将 Linux 内核转换为硬实时操作系统，同时提供如实时调度器、实时内存分配器和实时锁等功能，使其能够满足实时应用程序的需求。在 Linux 系统中，流量的转发是通过内核的流量控制（Traffic Control, TC）子系统[73]完成的。该子系统运行在网络协议栈与网卡驱动程序之间，负责向网卡驱动程序提供需要发送的数据帧。TC 可以根据 Socket 优先级，将不同优先级的流量映射至不同队列，并通过为不同队列配置排队规则（queueing discipline, qdisc）实现流量控制。TC 根据 qdisc 策略，对队列中的流量进行重新排队，并按照新的顺序发送它们。默认情况下，qdisc 配置为先进先出。为了实现 TAS 机制，实时性内核补丁引入了两个新的 qdisc，分别为最早发送时间优先和时间感知优先级整形器。其中 ETF 通过在应用层程序设置数据帧的 SO_TXTIME 字段，指定数据帧进入网卡驱动程序的准确时间，实现数据帧的确定性发送。TAPRIO 通过门控机制，按照配置的调度表控制指定门的开关，实现数据帧的确定性发送。数据帧经过 TC 处理后，进入网卡驱动程序的环形发送缓冲区。I210 网卡驱动程序支持两种类型的环形发送缓冲区，分别为严格保留队列和严格优先级队列。严格保留队列根据时间调度来传输数据，适合时间敏感流的传输。严格优先级队列根据严格优先级来传输数据，适合非时间敏感流的传输。在 Intel I210 网卡中支持硬件时间戳功能。网卡驱动程序可以通过硬件时间戳实现高精度的时间同步。同时，可以根据网卡中硬件时钟同步操作系统中的系统时钟，使操作系统按照精确的时间控制应用层程序运行。

3.5.2 实验参数设置

上一节中的实验结果表明半实物仿真接口在将真实流量映射为仿真流量的过程中会引入一定的时延，不适合 CT 流的仿真，因此在本实验中使用半实物仿真模拟 BE 流的传输。采用

图 3.15 所示网络拓扑，其中PC_1为 CT 流发送端，PC_2为 BE 流发送端。PC_3为接收端。为了探究半实物仿真系统性能，本文设计了四组对比实验。第一组实验采用真实物理实验系统，在PC_1中利用实时 OPC UA PubSub 程序发送 PCP = 7 的流量充当 CT 流，在PC_2中利用 Iperf3 程序发送 PCP = 0 的 UDP 流量充当 BE 流；第二组实验采用软件仿真系统，在PC_1中利用可调度定时发送模块发送 PCP = 7 的流量充当 CT 流，在PC_2中利用定时发送模块发送 PCP = 0 的流量充当 BE 流；第三组实验采用真实设备的半实物仿真系统，在PC_1中利用可调度定时发送模块发送 PCP = 7 的流量充当 CT 流，在PC_2中，利用 Ostinato 软件和 Linux 虚拟网络接口共同构成 BE 流发送端，发送 PCP=0 的流量；第四组实验采用真实程序的半实物仿真系统，在PC_1中利用可调度定时发送模块发送 PCP = 7 的流量充当 CT 流，在PC_2中，利用 Ostinato 软件和 NestingExtupperNIC 模块共同构成 BE 流发送端，发送 PCP=0 的流量。

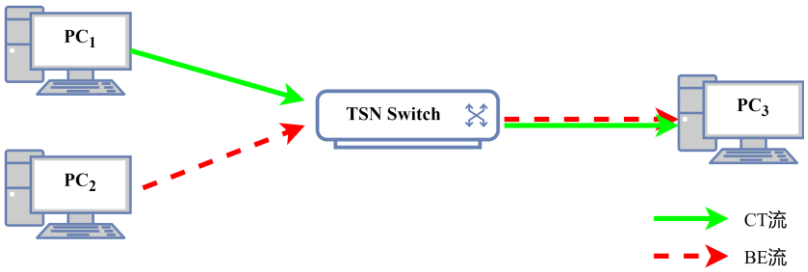


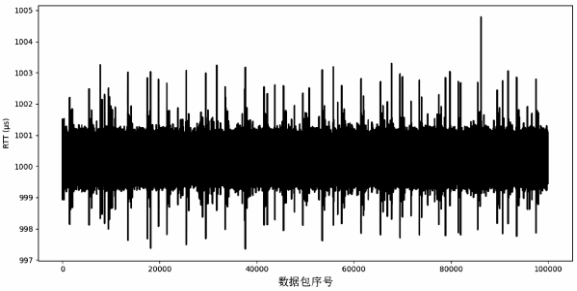
图 3.15 半实物仿真系统性能实验网络拓扑

在实验中，设置PC_1与PC_3应用层软件发送循环周期为 $250\mu s$，且指定在每个周期中先接收数据后发送数据。在当前周期收到的数据帧需要在下一个周期中进行回传。在PC_1中计算 CT 流数据帧往返时间（Round-Trip Time, RTT），CT 流 RTT = 4 * $250\mu s$ + 交换机处排队时间。设置PC_2应用层软件发送间隔为 $60\mu s$。设置 GCL 周期为 $250\mu s$，其中$T_{CT} = 100\mu s$，$T_{BE} = 150\mu s$。理论上 CT 流 RTT 为$1000\mu s$，BE 流时延为$29.62\mu s$。

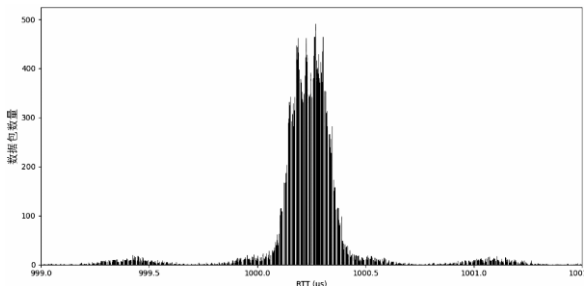
3.5.3 结果分析

图 3.16 为不同仿真环境中 CT 流实验结果统计，其中图 a、c、e、g 为 25s 内 CT 流数据帧（共计 100000 个）的 RTT 统计，图 b、d、f、h 为 CT 流数据帧的 RTT 分布情况。图 3.16(a)和图 3.16(b)为真实物理实验环境中 CT 流的 RTT 统计分析，通过在 TSN 交换机和端系统中进行相关配置，使 CT 流数据帧在交换机出端口处无需排队，因此 RTT 集中在$1000\mu s$附近。但由于时间同步等因素造成的误差，RTT 在$1000\mu s$处呈泊松分布。图 3.16(c)至图 3.16(h)分别为软件仿真环境、真实设备半实物仿真环境和真实程序半实物仿真环境中的 CT 流 RTT 分布情况。在以上三种环境中，均通过仿真软件生成 CT 流，由于仿真软件严格按照预设的时间间

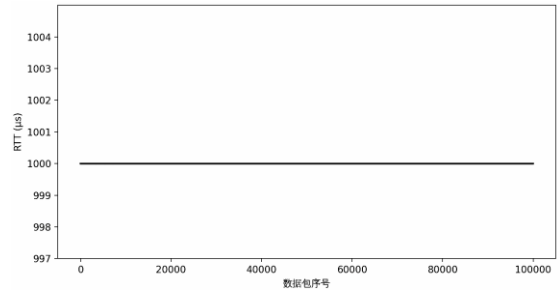
隔发送数据，同时不会出现时间同步等问题，因此 CT 流的 RTT 全部集中在$1000\mu s$。



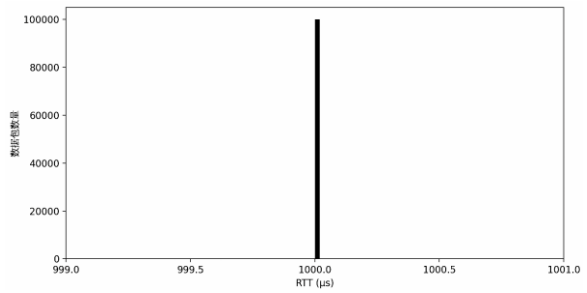
(a) 真实物理环境 CT 流 RTT 统计



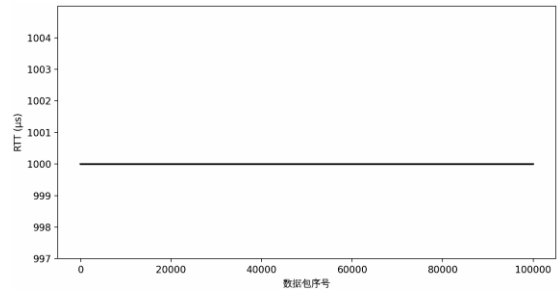
(b) 真实物理环境 CT 流 RTT 分布



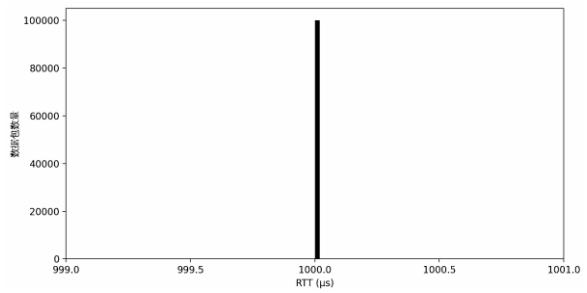
(c) 软件仿真环境 CT 流 RTT 统计



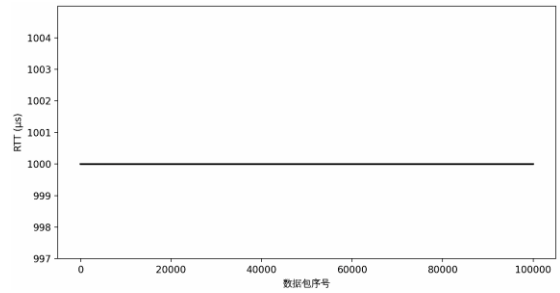
(d) 软件仿真环境 CT 流 RTT 分布



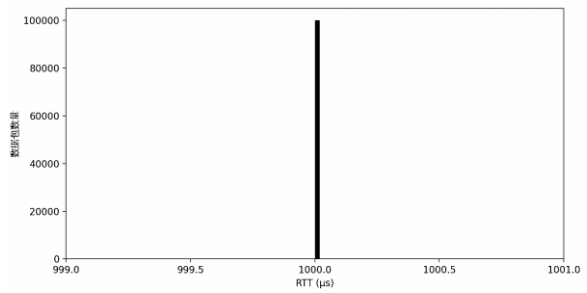
(e) 真实设备半实物仿真环境 CT 流 RTT 统计



(f) 真实设备半实物仿真环境 CT 流 RTT 统计



(g) 真实程序半实物仿真环境 CT 流 RTT 统计

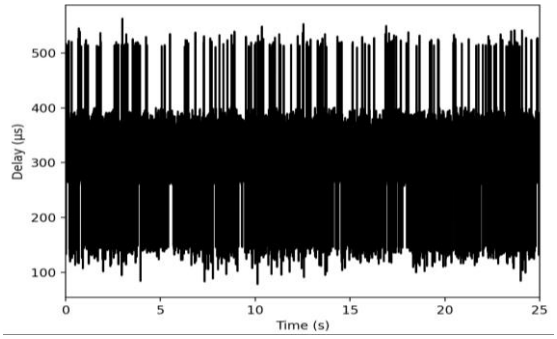


(h) 真实程序半实物仿真环境 CT 流 RTT 统计

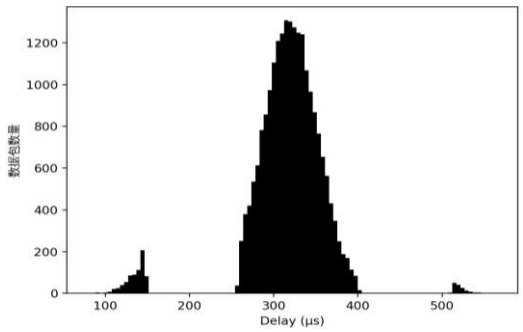
图 3.16 不同仿真环境中 CT 流实验结果统计

图 3.17 为不同仿真环境中 BE 流实验结果的统计，其中图 a、c、e、g 为 25s 内 BE 流时延统计，图 b、d、f、h 为时延分布情况。图 3.17(a)和 3.17(b)为真实物理环境中 BE 流时延统计，由于在真实环境中无法严格控制数据的发送时间，因此大部分流量由于抖动等因素错过当前周期的 BE 时隙窗口，需要排队等待至下一个时隙进行传输，大约需要排队$100\mu s$。由于流量在交换机处堆积，极少部分流量需要排队至更晚的时隙中传输。图 3.17(c)和图 3.17(d)为软件仿真环境中 BE 流时延统计，由于仿真软件严格控制流量的发送时间，因此 BE 流将会在

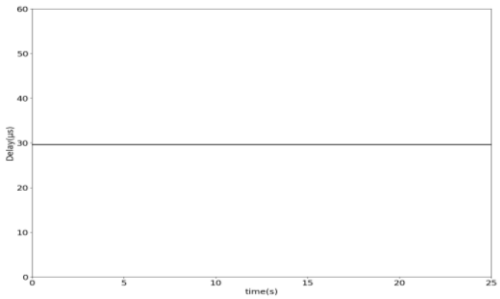
预设的 BE 时隙窗口中进行传输，无需排队等待。图 3.17(e)和图 3.17(f)为真实设备半实物仿真环境中 BE 流时延统计，在真实流量通过 Socket 通信进入仿真环境过程中会带来一定的时延和抖动，导致部分 BE 流错过当前 BE 时隙，需要排队等待。图 3.17(g)和图 3.17(h)为真实程序半实物仿真环境中 BE 流时延统计，真实流量通过特定 TAP 设备进入仿真环境。由于该半实物仿真接口引入的时延较小，因此大部分 BE 流在当前周期的时隙窗口中进行传输。



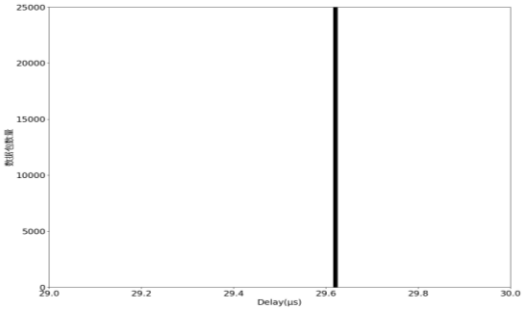
(a) 真实物理环境 BE 流时延统计



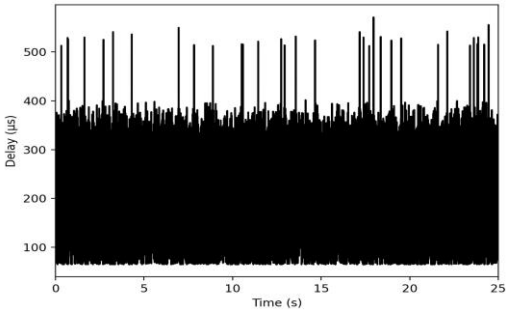
(b) 真实物理环境 BE 流时延分布



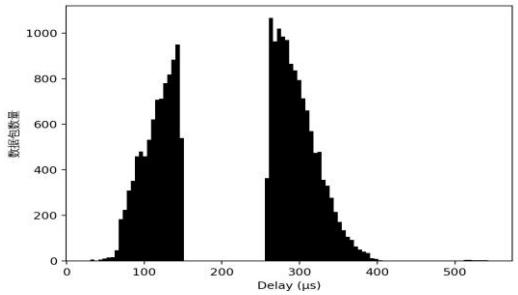
(c) 软件仿真环境 BE 流时延统计



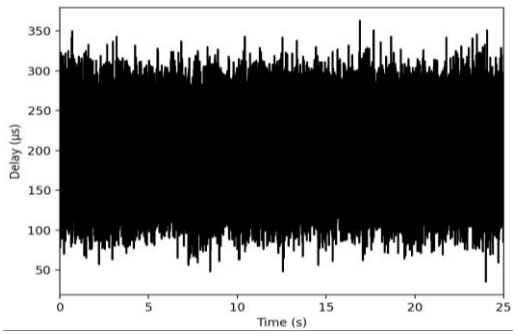
(d) 软件仿真环境 BE 流时延分布



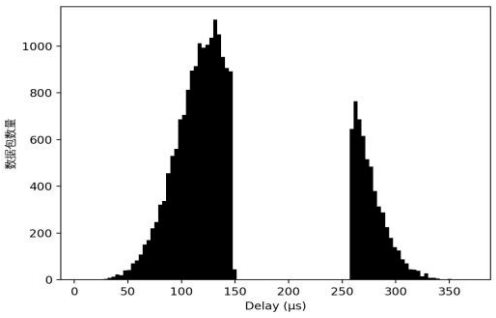
(e) 真实设备半实物仿真环境 BE 流时延统计



(f) 真实设备半实物仿真环境 BE 流时延分布



(g) 真实程序半实物仿真环境 BE 流时延统计



(h) 真实程序半实物仿真环境 BE 流时延分布

图 3.17 不同仿真环境中 BE 流实验结果统计

表 3.2 总结了不同仿真环境下 CT 流 RTT 与 BE 流时延统计。在真实物理环境中，由于在端系统中对 CT 流进行优化，提高了数据发送的确定性，使 CT 流可以在预期的时隙窗口中进行传输，CT 流的 RTT 在 997.35μs到 1004.79μs之间小幅度波动，符合预期。而在端系统中并未对 BE 流进行优化，流量抖动较大。这将导致大量 BE 流错过当前周期的 BE 时隙窗口，进而需要排队等待，导致 BE 流延迟在 78.63μs到 565.41μs之间波动。在软件仿真环境中，由于仿真软件可以严格按照预设的时间间隔发送数据，且不存在干扰因素，因此 CT 流和 BE 流严格按照预期运行，所有 CT 流的 RTT 为 1000μs，所有 BE 流的时延为 29.62μs。在真实设备的半实物仿真环境中，利用仿真软件生成 CT 流，因此实验结果和软件仿真中一致。而 BE 流通过 NestingExtlowerNIC 模块进入仿真网络，在此过程中会引入一定的时延，导致大部分 BE 流错过当前时隙窗口，BE 流时延在 25.63μs到 547.45μs之间波动。在真实程序的半实物仿真环境中，BE 流通过 NestingExtupperNIC 模块进行仿真网络。由于在此过程中会引入一定的时延，导致少部分 BE 流错过当前时隙窗口，BE 流时延在 34.21μs到 357.23μs之间波动。与真实设备的半实物仿真环境相比，真实程序的半实物仿真环境将引入更小的时延，因此 BE 流时延标准差更小，为 73.89μs。通过在不同环境下进行实验对比，发现在本文所开发的半实物仿真系统中可以得到与真实物理系统相近的实验结果，证明本文所开发的半实物仿真系统可以对 TSN 网络进行更贴近真实的仿真验证。

表 3.2 不同仿真环境下 CT 流 RTT 与 BE 流时延统计（单位μs）

	CT 流 RTT				BE 流时延			
	平均值	标准差	最大值	最小值	平均值	标准差	最大值	最小值
真实物理环境	1000.25	0.25	1004.79	997.35	316.61	47.71	565.41	78.63
软件仿真环境	1000	0	1000	1000	29.62	0	29.62	29.62
真实设备半实物仿真环境	1000	0	1000	1000	222.33	93.38	547.45	25.63
真实程序半实物仿真环境	1000	0	1000	1000	162.45	73.89	357.23	34.21

3.6 本章小结

在本章中，首先对 TSN 半实物仿真问题进行分析，将问题划分为实时调度机制和半实物仿真接口并分别进行设计实现。其次，针对真实设备的半实物仿真与真实程序的半实物仿真，设计开发了两种不同的半实物仿真接口模块。然后，对本文所提出的半实物仿真接口可行性进行验证，并利用真实流量到仿真流量映射过程中引入的延迟，对本文所设计的半实物仿真

接口性能进行验证。最后，利用 Linux 操作系统和 TSN 交换机，搭建真实物理实验系统，并通过真实物理系统、软件仿真系统以及半实物仿真系统中的实验结果对比，对本文所设计的半实物仿真系统性能进行验证。结果表明，本文提出的半实物仿真接口模块可以实现真实流量和仿真流量的相互映射，相比于仿真软件，可以提供更贴近真实的仿真结果。但在目前半实物仿真的实现过程中会引入一定的时延，不适合对时间敏感的 CT 流的仿真，更适合对时间不敏感的 BE 流的仿真。

第四章 基于自适应时隙窗口的调度算法

本章针对 TSN 动态网络调度问题，提出一种基于自适应时隙窗口的调度算法。并利用软件仿真系统与半实物仿真系统，对所提调度算法进行性能验证。

4.1 引言

在 TSN 中通过配合适当的流量调度机制，可以保障时间敏感流的确定性传输，即满足时间敏感流的时延和抖动等要求。其中，IEEE 802.1Qbv 协议提出的 TAS 机制备受关注。该机制使流量最多进入 8 个队列，根据 GCL 控制队列门的开启和关闭，以调节队列中流量的传输。通过配置合理的 GCL 信息，可以将时间敏感流和非时间敏感流在时域上进行划分，保障时间敏感流量的传输。

传统的 TAS 调度算法针对静态网络，且需要提前已知网络中流量特征，如报文大小，周期等信息，不适用于动态改变的网络。为了解决此问题，Nasrallah 等人[42]提出了一种基于窗口的动态带宽调节算法，称为 ASW。该算法根据 CT 流时延动态调节 GCL 中 CT 流和 BE 流之间的时隙窗口大小比例。然而，时隙窗口的动态变化会导致 CT 流在传输过程中发生抖动，可能导致时延超出允许的最大范围。Reusch 等人[74]提出，利用网络演算估算网络中流量的最差时延上界，对于超出时延上界的流量，调整其经过的交换机端口的 GCL 对应时隙大小。Su 等人[75]针对 5G 前传网络，提出了一种传输窗口动态调整机制，在 5G 领域保障时间敏感流量的传输。Li 等人[76]针对工业物联网应用的时间敏感网络中流量的路由与调度问题，考虑传输时延以及剩余带宽确定每条增量流量的最优路由路径，为 CT 流按照传输时延进行带宽分配，进而确定每个交换机的 CT 时隙。

从上面的工作可以看出，基于时隙窗口的调度算法的核心思想是通过流量时延等信息，动态调节对应时隙窗口的大小，进而满足对应流量的传输，保障流量的实时性。以 ASW 为核心的算法大都采用固定步长或者可变步长的方式调节时隙窗口长度。其中，固定步长调节的优势是计算简单，缺点是步长难以确定。过大的步长可能会造成时隙窗口调节不精确，过小的步长可能会造成时隙窗口调节速度慢，造成流量超出时延上限。可变步长调节的优势是控制精度和稳定性更高，缺点是计算复杂，实现难度相对较高。在算法适用的网络规模方面，ASW 算法仅考虑了单交换机的简单网络拓扑，不适应于多交换机复杂网络拓扑。

为了解决动态流量调度问题，本章首先对 TSN 流量调度问题进行分析建模，然后在 ASW 算法的基础上，提出了三点改进：提出了一种选择被调节端口的方法；提出了一种结合可变

步长和固定步长优势的增大时隙窗口的方法；提出了一种根据流传输时间减小时隙窗口的方法。通过这三点改进，使得改进的调度算法保障 CT 流确定性传输的同时，尽可能减小网络抖动。最后利用软件仿真系统和半实物仿真系统，对所提算法进行仿真验证。

4.2 问题模型

4.2.1 网络模型

TSN 网络由终端节点ES，交换机节点SW以及全双工数据链路组成。假设网络中的所有节点都支持 TSN 功能，支持 IEEE 802.1AS 和 IEEE 802.1Qbv 标准，网络中所有节点时间均已同步，TSN 交换机通过 TAS 机制控制数据帧的传输。

利用有向图$G=(E,V)$表示 TSN 网络，其中$V=ES\cup SW$表示网络中所有的设备的集合，包括终端节点ES和交换机SW。E表示网络中的数据链路的集合，$[v_a,v_b]\in E$表示从发送设备v_a到接收设备v_b的单向链路，$[v_a,v_b]\in E$和$[v_b,v_a]\in E$表示连接设备v_a和设备v_b的全双工链路。图 4.1 是一个 TSN 网络拓扑的举例，包括 4 个ES和 2 个SW，双向箭头表示设备之间的全双工数据链路。网络中交换机出端口为p_j，利用出端口的集合U_i表示流f_i在网络中传输的路径，其中i为流的编号，j为交换机端口的编号。

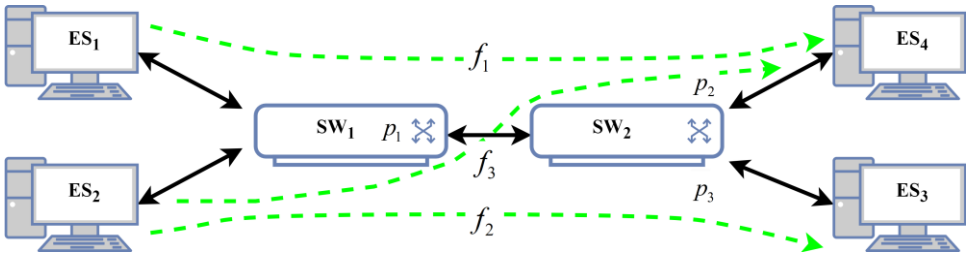


图 4.1 TSN 网络拓扑

4.2.2 流量模型

在 TSN 网络中，通过 VLAN 标签中的 PCP 字段，将流量划分为 8 种不同的优先级，其中 CT 流具有最高的优先级，对时延和抖动具有最严格的要求。利用集合F表示网络中流量的集合，利用五元组$f_i=\{T_i,L_i,D_i,ES_i^{src},ES_i^{dst}\}$描述网络中的流$f_i,i\in\{1,2,\cdots,n\}$，其中$T_i$为流的传输周期，$L_i$为流的报文长度，$D_i$为流允许的最大时延，$ES_i^{src}$为流的源地址，$ES_i^{dst}$表示流的目的地址。

4.2.3 问题模型

TSN 网络设计的首要目标是要保障实时应用中具有严格时间约束的 CT 流的实时性能。如果一个有严格时间要求的数据帧在交换机的出口端口队列中等待，将导致一个非确定的排队时延。如果该数据帧的端到端时延d超过其允许的最大时延D，则无法满足实时应用要求。

利用 CNC 获取当前网络中 GCL 周期T_{GCL}，假设 GCL 中只包含两个条目，即一个 GCL 周期中只包括 CT 流时隙T_{CT}和 BE 流时隙T_{BE}。T_{GCL}可以用公式 4.1 进行计算。

$$T_{GCL} = T_{CT} + T_{BE} \tag{4.1}$$

本文要解决的问题可以描述为：对于给定的 TSN 网络拓扑G，网络中的所有 CT 流f_i从发送端经过若干个交换机节点出端口后到达接收端的实际时延为d_i，要求动态调节 CT 时隙窗口T_{CT}和 BE 时隙窗口T_{BE}的大小，不仅要满足 CT 流的确定性传输要求，还要为 BE 流提供尽可能大的时隙窗口。优化目标函数可以用公式 4.2 进行表示。

$$\min K = \frac{T_{CT}}{T_{GCL}} \tag{4.2}$$

约束条件为所有 CT 流时延小于允许的最大时延，即约束条件可以用公式 4.3 进行表示。

$$d_i < D_i \, , \forall f_i \in F \tag{4.3}$$

4.3 基于自适应时隙窗口的调度算法

在本小节中，将在传统 ASW 算法的基础上进行改进。首先利用流量时延，计算网络中交换机端口的权重系数，并根据权重系数选择被调节的交换机端口。其次通过流量时延和流最后传输时间，调节对应交换机端口的时隙窗口大小。

4.3.1 选择被调节端口

为网络中所有交换机端口p_j赋予权重系数w_j，其中$j \in M$，表示网络中交换机端口的编号，M为网络中所有交换机端口数量。在初始化阶段，令所有交换机端口的权重系数为 0，即$w_j = 0 \, , j \in M$。在每个 GCL 周期结束后，CNC 计算流f_i的实际时延d_i，将网络中时延大于触发时隙调节上限D_i^{upper}的 CT 流f_i添加到集合Q中。若集合$Q \neq \emptyset$，表示网络中存在 CT 流不满足时延要求。依次取出集合Q中的流f_i，根据流f_i的源地址ES_i^{src}和目的地址ES_i^{dst}，结合网络拓扑G，得到流f_i在传输过程中经过的交换机出端口集合U_i。交换机出端口p_j的权重系数w_j可以用

公式 4.4 进行计算。

$$w_j = w_j + \frac{d_i}{D_i^{upper}}, p_j \in U_i \tag{4.4}$$

若集合$Q = \emptyset$，表示网络中所有 CT 流均满足时延要求。交换机出端口p_j在当前 GCL 周期内的偏移时间O_j为 CT 流最后传输时间t_j^{end}与当前 GCL 周期开始时间t_j^{start}差值。O_j可以用公式 4.5 进行计算。

$$O_j = t_j^{end} - t_j^{start}, j \in \{1,2,\cdots,M\} \tag{4.5}$$

交换机出端口p_j权重系数w_j可以用公式 4.6 进行计算。

$$w_j = \frac{1}{O_j}, j \in \{1,2,\cdots,M\} \tag{4.6}$$

CNC 选择网络中权重系数w_j最大的端口作为被调节端口，获取对应端口的 GCL 信息。

4.3.2 调节时隙分配

当网络中 CT 流增加时，原先的 CT 流时隙窗口长度T_{CT}不足以满足所有 CT 流的传输，部分 CT 流需要排队至下一个 CT 时隙窗口进行传输，这可能导致 CT 流时延超出允许的最大时延。因此需要增大 CT 时隙窗口长度，以满足所有 CT 流的传输。CT 时隙窗口增大步长为单位调节步长$Step$与被调节端口的权重系数w_j的乘积。同时，限制单次调节步长上限为$Step_{max}$，并且 CT 时隙窗口T_{CT}不得超过 GCL 周期T_{GCL}。CT 时隙窗口长度T_{CT}可以用公式 4.7 进行计算。

$$T_{CT} = \begin{cases} T_{CT} + \min(w_j \cdot Step, Step_{max}), & T_{CT} + \min(w_j \cdot Step, Step_{max}) \le T_{GCL} \\ T_{GCL}, & T_{CT} + \min(w_j \cdot Step, Step_{max}) > T_{GCL} \end{cases} \tag{4.7}$$

当网络中 CT 流减少时，需要减小 CT 时隙窗口长度，为 BE 流提供尽可能大的时隙窗口。若网络中采用无等待调度，则 CT 流在传输过程中不会在交换机处排队，CT 流时延为理论最小时延，CNC 无法通过 CT 流时延判断是否需要减小对应时隙。针对此问题，本文提出根据交换机出端口p_j在当前 GCL 周期内的偏移时间O_j来减小 CT 时隙窗口。由于流量传输过程中可能发生抖动，为了提高系统稳定性和数据传输的可靠性，引入时隙调节比例因子Exp，为 CT 时隙预留一定比例的缓冲区。CT 时隙窗口长度T_{CT}可以用的计公式 4.8 进行计算。

$$T_{CT} = Exp \cdot O_j \tag{4.8}$$

4.3.3 算法实现

算法 4-1 为基于自适应时隙窗口的调度算法伪代码。将网络中的所有交换机出端口的权值系数赋值为 0（第 1 行）。CNC 在每次 GCL 周期结束之后，获取所有 CT 流的时延。将时延大于触发时延调节上限的 CT 流加入到集合Q中（第 2~8 行）。若触发流集合$Q \neq \emptyset$，依次取出集合中的流f_i，查询流经过的交换机出端口，并更新对应端口权重系数（第 9~14 行）。选择权重系数最大的端口p_j作为被调节端口，增大对应端口中 CT 时隙窗口长度（第 15~16 行）。将被调节端口的权重系数赋值为 0（第 17 行）。若触发流集合$Q = \emptyset$，表示网络中所有 CT 流的时延均可以满足允许的最大时延要求。网络中所有交换机出端口的权重系数w_j等于当前 GCL 周期中 CT 流最后传输时间与 GCL 周期开始时间的偏移时间O_j的倒数（第 19~20 行）。选择权重系数最大的端口p_j作为被调节端口，减小对应端口中 CT 时隙窗口长度（第 21~22 行）。将网络中的所有交换机出端口的权值系数赋值为 0（第 23 行）。

算法 4-1 基于自适应时隙窗口的调度算法

输入：网络拓扑G，网络中交换机端口数量M，流集合F，流f_i在网络中传输路径的向量U_i，触发时延调节上限D_i^{upper}，时延超出触发时延调节上限的流集合Q，交换机端口权重系数w_j，单位调节步长$Step$，单次调节步长上限$Step_{max}$，交换机出端口p_j的 GCL 周期内最后一帧 CT 流流出交换机出端口的偏移时间为O_j，时隙调节比例因子Exp

输出：调节后的 CT 流时隙窗口T_{CT}

1 $w_j \leftarrow 0, j \in \{1,2,\cdots,M\}$
2 **while** $(t \bmod T_{GCL}) == 0$ **do**
3 　**for** f_i **in** F **do**
4 　　计算 d_i
5 　　**if** $d_i > D_i^{upper}$ **then**
6 　　$Q \cdot append(f_i)$
7 　　**end if**
8 　**end for**
9 　**if** $Q \neq \emptyset$ **then**
10 　　**for** f_i **in** Q **do**
11 　　**for** p_j **in** U_i **do**

$w_j \leftarrow w_j + \frac{d_i}{D_i^{upper}}$

end for

end for

选择最大权重系数对应端口p_j作为调节端口

$T_{CT} \leftarrow min\left(T_{CT} + min\left(w_j \cdot Step, Step_{max}\right), T_{GCL}\right)$

$w_j = 0$

end if

if $Q = \emptyset$ **then**

$w_j \leftarrow \frac{1}{O_j}, j \in \{1,2,\cdots,M\}$

选择最大权重系数对应端口p_j作为调节端口

$T_{CT} \leftarrow max\left(O_j \cdot Exp, T_{CT}^{low}\right)$

$w_j \leftarrow 0, j \in \{1,2,\cdots,M\}$

end if

end while

以图 4.1 网络拓扑为例，假设第一个 GCL 周期结束后，CNC 获得f_1、f_2、f_3的实际时延$d_1 = 510\mu s$，$d_2 = 450\mu s$，$d_3 = 300\mu s$，f_1、f_2、f_3的时延均大于触发时隙调节上限$150\mu s$,根据公式 4.11 计算得交换机端口权重系数$w_1 = 8.4$，$w_2 = 5.4$，$w_3 = 3.0$，CNC 选择网络中权重系数最大的出端口p_1作为被调节端口，调节该出端口的时隙，并更新权重系数$w_1 = 0$。第二个 GCL 周期结束后，CNC 获得f_1、f_2、f_3的实际时延$d_1 = 420\mu s$，$d_2 = 210\mu s$，$d_3 = 100\mu s$，f_1、f_2的时延大于触发时隙调节上限，根据公式 4.11 计算得交换机出端口权重系数$w_1 = 4.2$，$w_2 = 8.2$，$w_3 = 4.4$，CNC 选择网络中权重系数最大的出端口p_2作为调节端口，调节该出端口的时隙，并更新权重系数$w_2 = 0$。第三个 GCL 周期结束后，CNC 获得f_1、f_2、f_3的时延均小于触发时隙调节上限，CNC 计算当前 GCL 周期中 CT 流最后传输时间与 GCL 周期开始时间的偏移时间，分别为O_1、O_2、O_3，根据公式 4.7 计算得交换机出端口权重系数$w_1 = \frac{1}{O_1}$，$w_2 = \frac{1}{O_2}$，$w_3 = \frac{1}{O_3}$，CNC 选择网络中权重系数最大的端口作为被调节端口，调节该端口的时隙，并更新权重系数$w_1 = 0$，$w_2 = 0$，$w_3 = 0$。

4.4 软件仿真结果分析

在本小节中，将利用 OMNeT++仿真软件搭建软件仿真环境，对本文所提出的算法与 ASW 算法以及禁用 CNC 的网络进行对比，验证本文所提算法的有效性。

4.4.1 实验设置

在 Ubuntu 18.04 系统中，利用 OMNeT++仿真软件并基于 NeSTiNg 框架，建立仿真网络环境。为了验证算法性能，实验设置了单交换机拓扑和多交换机拓扑两种应用场景。

图 4.2 为一个包含 1 个交换机和 13 个终端单交换机网络拓扑。假设交换机的服务能力相同，带宽均为 1Gbps，处理时延$t_{proc}=5\mu s$，链路传播时延$t_{trans}=0.1\mu s$。交换机SW_1初始$T_{GCL}=400\mu s$，$T_{CT}=100\mu s$，$T_{BE}=300\mu s$。CT 流共 10 条，流属性设置如下：$f_i=\{T_i=400\mu s, L_i=800B, D_i=1ms, ES_i^{src}=ES_i, ES_i^{dst}=ES_{13}\}, i\in\{1,2,\cdots,10\}$。其中前 5 条流从 0 时刻开始周期性发送。在网络流量增加的实验中，后 5 条流从 20ms 周期性发送。在网络流量减少的实验中，后 5 条流仅在 0ms 至 20ms 期间周期性发送。BE 流共 2 条，流属性设置如下：$T_i=60\mu s$，$L_i=1500B$，$ES_i^{src}=ES_i$，$ES_i^{dst}=ES_{13}$，$i\in\{11,12\}$。

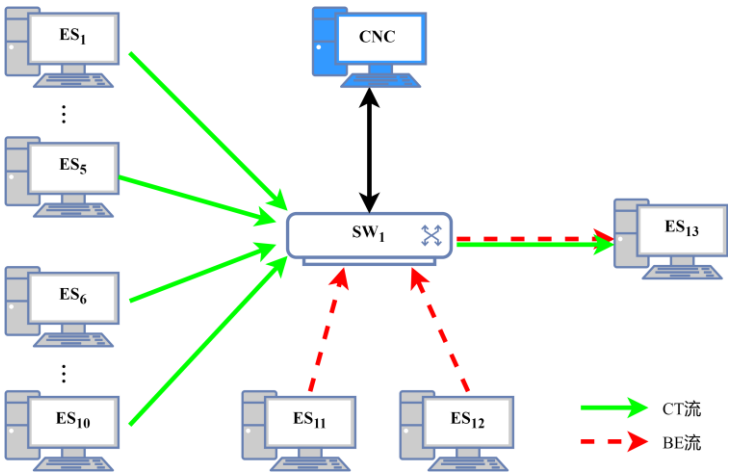


图 4.2 单交换机网络拓扑

图 4.3 为一个包含 2 个交换机和 15 个终端的多交换机网络拓扑。假设交换机的服务能力相同，带宽均为 1Gbps，处理时延$t_{proc}=5\mu s$，链路传播时延$t_{trans}=0.1\mu s$。交换机SW_1初始$T_{GCL}=400\mu s$，$T_{CT}=100\mu s$，$T_{BE}=300\mu s$。交换机SW_2初始$T_{GCL}=400\mu s$，$T_{CT}=90\mu s$，$T_{BE}=310\mu s$。CT 流共 10 条，流属性设置如下：$f_i=\{T_i=400\mu s, L_i=800B, D_i=1ms, ES_i^{src}=$

$ES_i, ES_i^{dst} = ES_{14}\}, i \in \{1,2\}$。$f_i = \{T_i = 400\mu s, L_i = 800B, D_i = 1ms, ES_i^{src} = ES_i, ES_i^{dst} = ES_{14}\}, i \in \{3,4\cdots,10\}$。其中前 5 条从 0 时刻开始周期性发送，后 5 条仅在 10ms 至 30ms 期间周期性发送。BE 流共 2 条，流属性设置如下：$T_i = 60\mu s$，$L_i = 1500B$，$ES_i^{src} = ES_i$，$ES_{11}^{dst} = ES_{13}$，$ES_{12}^{dst} = ES_{15}$，$i \in \{11,12\}$。

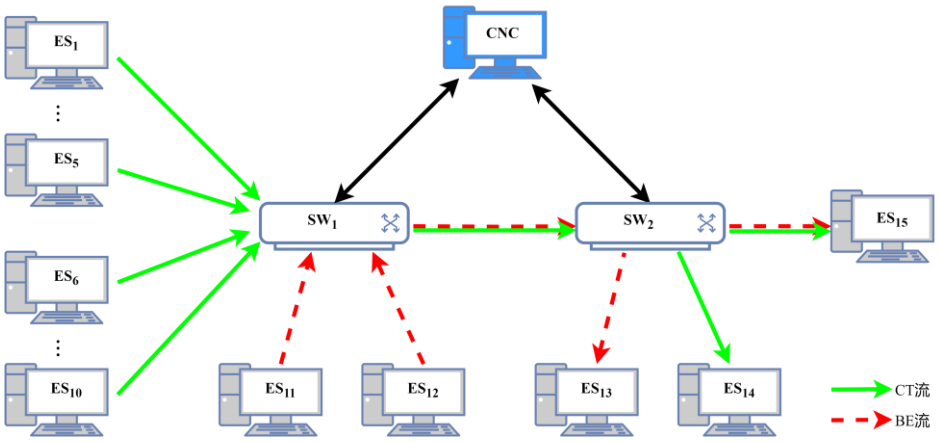


图 4.3 多交换机网络拓

参考 Nasrallah 等人[42]设置 ASW 算法的触发时隙调节下限为$50\mu s$，单次调节步长为$40\mu s$。本文所提的自适应时隙窗口调度算法的$D_i^{upper} = 150\mu s$，$Step = 5\mu s$，$Step_{max} = 40\mu s$。

4.4.2 单交换机网络拓扑的结果分析

（A）网络流量增加

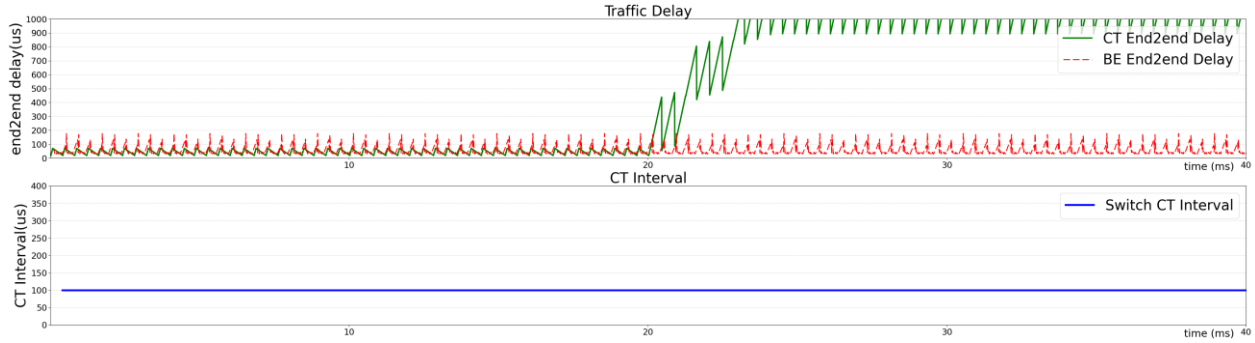
图 4.4 为单交换机网络中流量增加情况下，在三种不同算法控制下的网络仿真结果。包括网络中 CT 流时延与 BE 流时延的统计以及 CT 时隙窗口变化情况。

图 4.4(a)为禁用 CNC 情况下的网络仿真结果。在禁用 CNC 时，交换机的 CT 时隙大小保持不变。在 0 至 20ms 期间，CT 时隙满足 CT 流传输需求，CT 流时延较小。在 20ms 时，ES_6至ES_{10}开始发送，网络中 CT 流数量增加，此时 CT 时隙大小不足以满足流量传输需求，CT 流在交换机中排队等待发送，大部分 CT 流的时延超出了 1ms，超出允许的最大时延，图中超出纵坐标最大范围部分未显示。

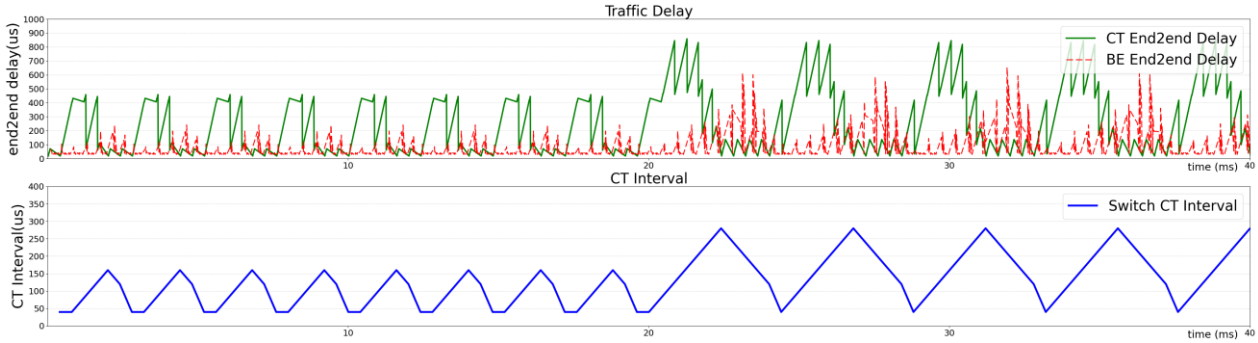
图 4.4(b)为采用 ASW 算法情况下的网络仿真结果。在使用 ASW 算法时，CNC 根据 CT 流时延采用固定步长调节交换机 CT 时隙。在 0 至 20ms 期间，ASW 算法不断试探最合理的时隙大小，造成时隙大小的上下波动，导致 CT 流的抖动较大。从 20ms 开始，网络中 CT 流数量增加，导致当前 CT 时隙无法满足其传输，流量时延增大。ASW 算法通过固定步长增大

CT 时隙窗口，使得 CT 流的传输时延逐渐满足最大传输时延要求。

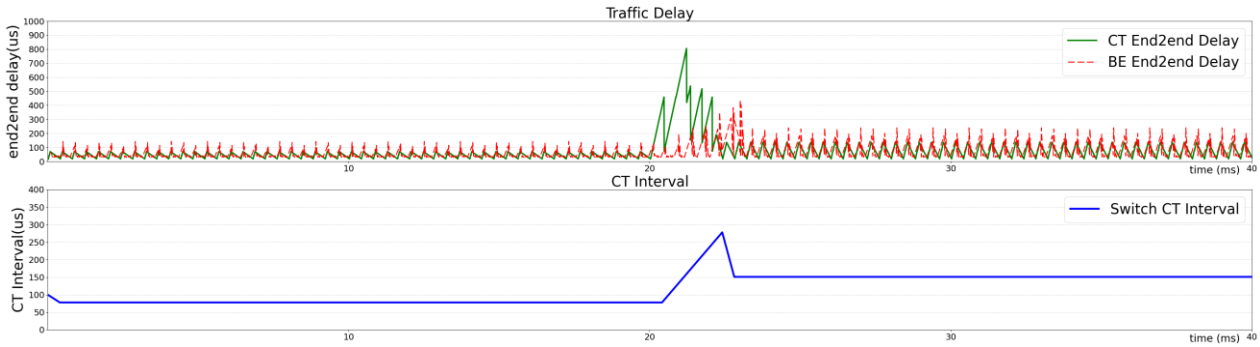
图 4.4(c)为采用本文所提出的算法的网络仿真结果。在使用本文所提算法时，CNC 根据 CT 流时延和数据传输时间调整时隙。在 0 至 20ms 期间，CNC 控制 CT 时隙减小至合理值，由于网络中没有新增流量，CT 时隙保持不变。从 20ms 开始，网络中 CT 流的数量增加，导致当前 CT 时隙无法满足其传输，流量时延增大，本算法增大 CT 时隙窗口。为了便于比较，本算法设置单次调节步长上限与 ASW 算法调节步长相同，在时隙窗口变化速度方面与 ASW 算法相近。本算法的单次调节步长为单位调节步长与调节权重系数的乘积值，并且限制单次调节步长不超过单次调节步长上限。当网络中 CT 流时延减小，算法中调节权重系数缩小，导致单次调节步长缩小，避免了时隙大小的上下波动。



(a) 禁用 CNC



(b) ASW 算法



(c) 本文所提算法

图 4.4 单交换机网络流量中增加情况

表 4.1 为在单交换机网络中流量增加情况下，三种不同算法控制的流量时延统计。在禁用 CNC 情况下，无法调节时隙窗口大小，因此当网络中流量增加的情况下，CT 时隙窗口无法满足所有 CT 流的传输，CT 流时延最大值为 1278.42μs，超出 CT 流允许的最大时延。而由于时隙窗口不发生变化，因此新增的 CT 流不影响 BE 流的传输，BE 流时延较小，平均时延为 40.74μs。在采用 ASW 算法的情况下，当网络中流量增加时，该算法控制时隙窗口增大以满足 CT 流的传输，CT 流最大时延为 858.5μs。但由于该算法需要不断改变时隙窗口大小，导致网络中流量抖动较大，CT 流时延标准差为 238.97μs，而 BE 流时延标准差为 131.01μs。在采用本文所提出的算法情况下，当网络中流量增加时，本算法控制时隙窗口增大，虽然会牺牲部分 BE 流传输所需的资源，但可以保障 CT 流的传输。CT 流时延最大值为 805.38μs，满足 CT 流传输要求。同时本算法可以避免时隙窗口的频繁变化，因此流量时延标准差较小，CT 流时延标准差为 58.47μs，BE 流时延标准差为 62.53μs。实验结果表明，在网络中流量增加的情况下，本章所提调度算法可以及时改变时隙窗口的大小，保障 CT 流的传输，同时减小网络中的抖动。

表 4.1 单交换机网络中流量增加情况时延统计（单位：μs）

	CT 流				BE 流			
	平均值	标准差	最大值	最小值	平均值	标准差	最大值	最小值
禁用 CNC	1040.19	400.57	1278.42	18.48	61.47	40.74	176.72	29.62
ASW 算法	282.28	238.97	858.5	18.48	129.28	131.01	653.35	29.62
本文算法	80.35	58.47	805.38	18.48	85.29	62.53	439.93	29.62

（B）网络流量减少

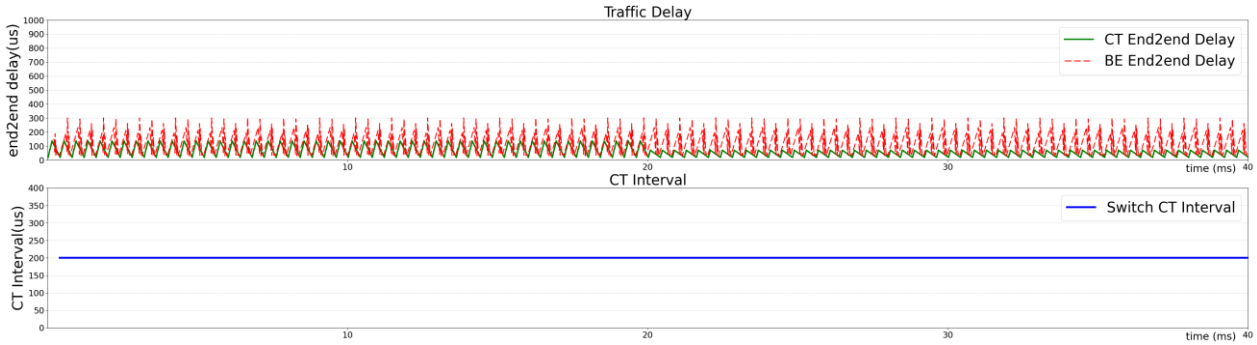
图 4.5 为单交换机网络中流量减少情况下，在三种不同算法控制下的网络仿真结果。包括网络中 CT 流时延与 BE 流时延的统计以及 CT 时隙窗口变化情况。

图 4.5(a)为禁用 CNC 情况下的网络仿真结果。在禁用 CNC 时，交换机的 CT 时隙大小保持不变。在 20ms 时，ES_6至ES_{10}停止发送流量，网络中 CT 流的数量减少，由于无法减小 CT 时隙大小，BE 流时延始终保持不变。

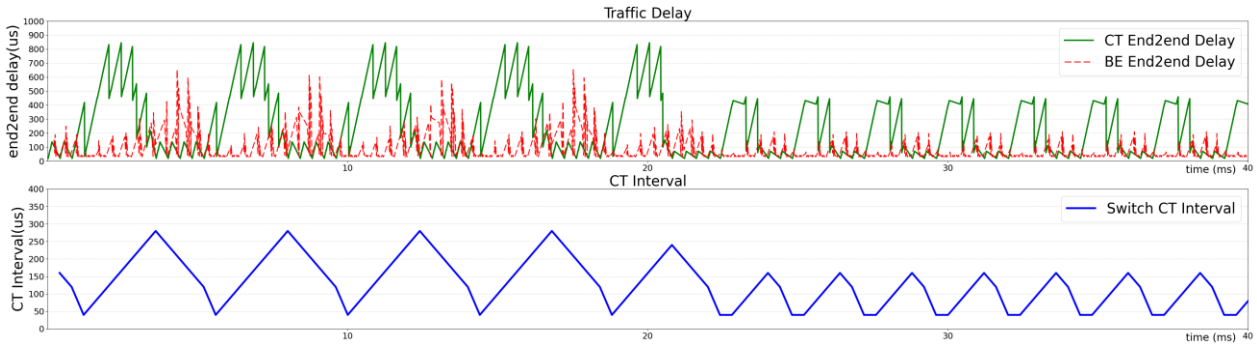
图 4.5(b)为采用 ASW 算法情况下的网络仿真结果。在使用 ASW 算法时，CNC 根据 CT 流时延调节时隙。从 20ms 开始，网络中 CT 流的数量减少，ASW 算法通过固定步长减小 CT 时隙窗口，不断试探最合理的时隙大小，虽然尽可能保障了 BE 流的传输，但造成时隙大小的上下波动，导致 CT 流的抖动较大。

图 4.5(c)为采用本文所提出的算法的网络仿真结果。在使用本文所提算法时，CNC 根据

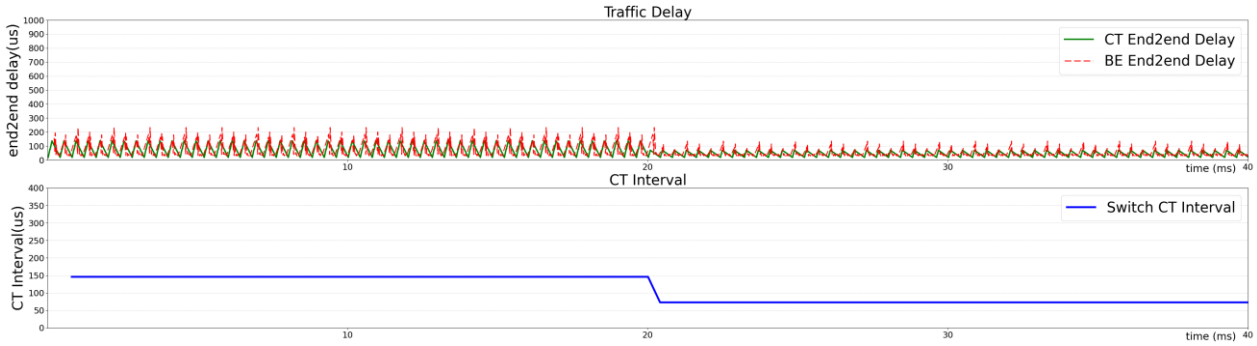
当前 GCL 周期中 CT 流最后传输时间减小对应时隙窗口大小。在 20ms 时，网络中 CT 流的数量减少，本算法控制 CT 时隙窗口减小，为 BE 流提供尽可能多的网络资源，BE 流时延减小。由于不需要不断试探最优的 CT 时隙窗口大小，可以在网络中流量减少的情况下有效减小网络抖动。



(a) 禁用 CNC



(b) ASW 算法



(c) 本文所提算法

图 4.5 单交换机网络中流量减小情况

表 4.2 为在单交换机网络中流量减少情况下，三种不同算法控制的流量时延统计。在禁用 CNC 情况下，无法调节时隙窗口大小，因此当网络中流量减少时，CT 时隙窗口保持不变，无法为 BE 流提供尽可能多的网络资源。BE 流时延始终保持不变，平均时延为 129.06μs。在采用 ASW 算法的情况下，当网络中流量减少时，该算法控制时隙窗口减小。但由于该算法需

要不断改变时隙窗口大小，导致网络中流量抖动较大，CT 流时延标准差为 206.74μs，而 BE 流时延标准差为 74.97μs。在采用本文所提出的算法情况下，当网络中流量较少时，本算法控制时隙窗口减小，为 BE 流提供尽可能多的网络资源，进而使 BE 流时延减小，BE 流平均时延为 55.12μs。同时本算法可以避免时隙窗口的频繁变化，因此流量时延标准差较小，CT 流时延标准差为 27.37μs，BE 流时延标准差为 36.17μs。由于在时隙窗口中最先发送的一帧 CT 流不需要排队，因此其时延最小，最小时延$= t_{proc} + 2 \cdot t_{trans} + 2.t_{prop} = 18.48\mu s$，其中交换机处理时延$t_{proc} = 5\mu s$，流量传输时延$t_{trans} = 6.64\mu s = \frac{800B+30B}{1Gbps}$，链路传播时延$t_{prop} = 0.1\mu s$。实验结果证明，在网络中流量减少的情况下，本章所提调度算法可以及时改变时隙窗口的大小，在保障 CT 流传输的同时，为 BE 流提供尽可能多的网络资源。

表 4.2 单交换机网络中流量减少情况时延统计（单位:μs）

	CT 流				BE 流			
	平均值	标准差	最大值	最小值	平均值	标准差	最大值	最小值
禁用 CNC	53.85	27.37	136.64	18.48	129.06	82.47	301.14	29.62
ASW 算法	228.83	206.74	845.22	18.48	75.55	74.97	653.35	29.62
本文算法	53.85	27.37	136.64	18.48	55.12	36.17	232.14	29.62

4.4.3 多交换机网络拓扑的结果分析

图 4.6 为多交换机网络中流量变化情况下，在三种不同算法控制下的网络仿真结果。包括网络中 CT 流时延与 BE 流时延的统计以及不同交换机的 CT 时隙窗口变化情况。

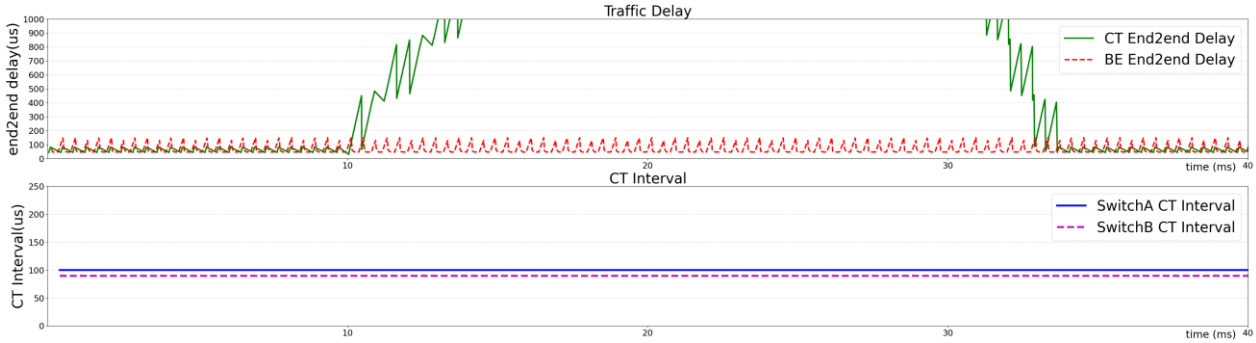
图 4.6(a)为禁用 CNC 情况下的网络仿真结果。在禁用 CNC 时，交换机时隙保持不变。在 0 至 10ms 期间，CT 时隙满足 CT 流传输需求，CT 流时延较小。在 10ms 时，ES_6至ES_{10}开始发送流量，网络中 CT 流的数量增加，此时 CT 时隙窗口大小不足以满足流量传输需求，CT 流在交换机中排队等待发送，大部分 CT 流的时延超出了 1ms，超出允许的最大时延，图中超出纵坐标最大范围部分未显示。在 30ms 时，ES_6至ES_{10}停止发送流量，网络中 CT 流的数量减少，交换机缓冲区中排队的流量逐渐发送，CT 流时延恢复到起始时大小。

图 4.6(b)为采用 ASW 算法情况下的网络仿真结果。在使用 ASW 算法时，CNC 根据 CT 流时延调节时隙。在 10ms 时，网络中 CT 流的数量增加，ASW 算法控制 CT 时隙窗口增大，不断寻找最优时隙窗口大小。在 30ms 时，网络中 CT 流的数量减少，ASW 算法控制 CT 时隙窗口减小。虽然该算法可以通过调整时隙窗口大小，保障 CT 流的传输。但由于无法选择出

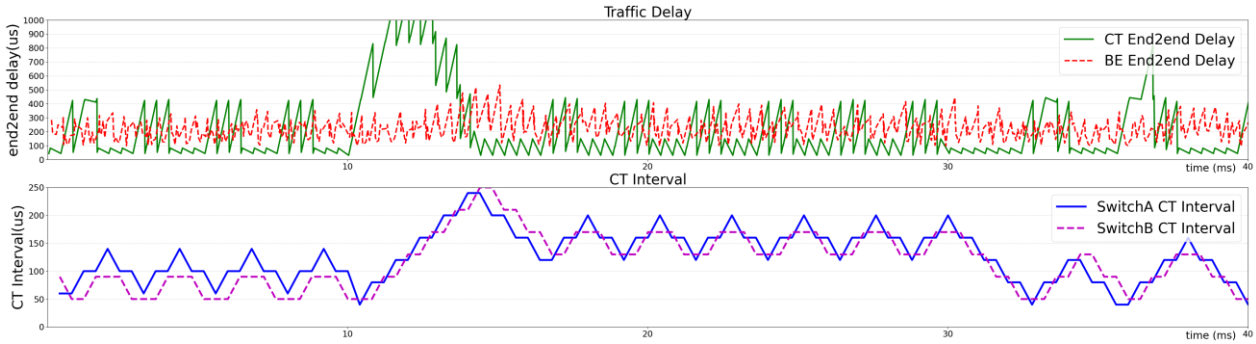
最优的被调节端口，导致无法快速通过改变时隙窗口大小来保障 CT 流的传输，在 15ms 附近，部分 CT 流由于排队等待，时延超出允许的最大时延。同时，由于时隙窗口大小的上下波动，导致网络中流量抖动较大。

图 4.5(c)为采用本文所提出的算法的网络仿真结果。在使用本文所提算法时，CNC 根据 CT 流时延和数据传输时间调节交换机 CT 时隙。在 0 至 10ms 期间，CNC 控制 CT 时隙减小至合理值，由于网络中没有新增流量，CT 时隙保持不变。从 10ms 开始，网络中 CT 流的数量增加，导致当前 CT 时隙无法满足其传输，流量时延增大。根据本算法 CNC 选择网络中权重系数最大的端口，增大其 CT 时隙，并将该端口的权值系数重置为 0。若调节完成后，网络中依然存在 CT 流时延超过触发时隙调节上限，此时再次选择网络中权重系数最大的交换机出端口，调节其 CT 时隙。如此往复，直到网络中所有 CT 流时延均小于触发时隙调节上限。从 30ms 开始，网络中 CT 流的数量减少，当网络中没有 CT 流时延超出触发时隙调节上限时，CNC 根据交换机出端口的当前 GCL 周期内 CT 流最后传输时间调整 CT 时隙大小，为 BE 流提供尽可能大的带宽。因为不需要不断试探最优 CT 时隙大小，有效减小了网络抖动。

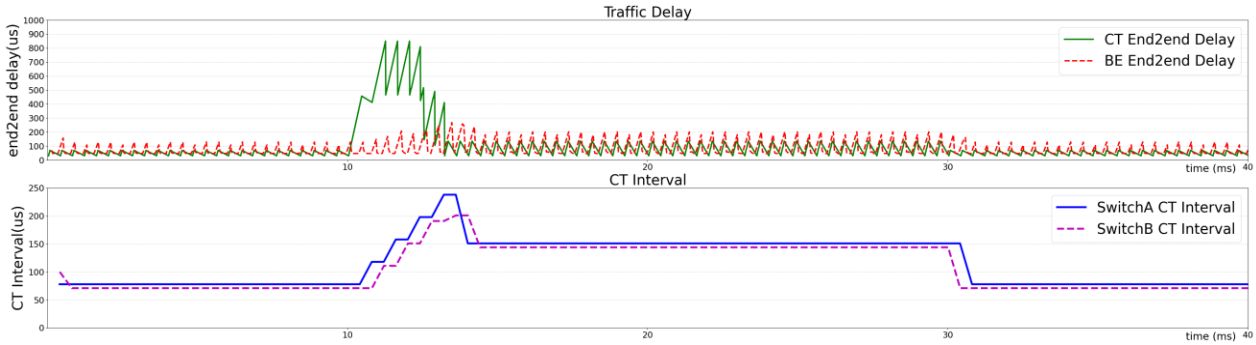
表 4.3 为在多交换机网络中流量变化情况下，三种不同算法控制的流量时延统计。在禁用 CNC 情况下，无法调节时隙窗口大小，因此当网络中流量变化时，CT 时隙窗口保持不变，部分 CT 流时延超出允许的最大时延。而在网络变化过程中，BE 流并未发生改变，因此 BE 流时延始终保持不变，平均时延为 70.66μs。在采用 ASW 算法的情况下，当网络中流量发生变化时，该算法控制时隙窗口改变。但由于 ASW 算法无法选择网络中最优的被调节端口，因此导致 CT 流最大时延为 1223.66μs，超出允许的最大时延。同时由于该算法需要不断改变时隙窗口大小，导致网络中流量抖动较大，CT 流时延标准差为 227.45μs，而 BE 流时延标准差为 68.07μs。在采用本文所提出的算法情况下，当网络中流量变化时，本算法选择当前最优的被调节端口并改变对应时隙窗口大小，CT 流最大时延为 850.22μs，满足允许的最大时延。同时，本算法在保障 CT 流传输的同时，可以为 BE 流提供尽可能多的网络资源，BE 流平均时延为 69.0μs。与 ASW 算法相比，本算法可以避免时隙窗口的频繁变化，因此流量时延标准差较小，CT 流时延标准差为 111.79μs，BE 流时延标准差为 36.65μs。实验结果证明，在多交换机复杂网络场景下，本算法可以根据流量时延选择最优的被调节端口，并改变对应时隙窗口大小，在保障 CT 流传输的同时尽可能减小网络中的抖动。



(a) 禁用 CNC



(b) ASW 算法



(c) 本文所提算法

图 4.6 多交换机网络中流量变化情况

表 4.3 多交换机网络中流量变化情况时延统计（单位:μs）

	CT 流				BE 流			
	平均值	标准差	最大值	最小值	平均值	标准差	最大值	最小值
禁用 CNC	420.79	537.66	1603.74	30.22	70.66	35.15	149.62	46.92
ASW 算法	212.68	227.45	1223.66	30.22	216.85	68.07	533.88	96.93
本文算法	82.79	111.79	850.22	30.22	69.0	36.65	267.62	46.92

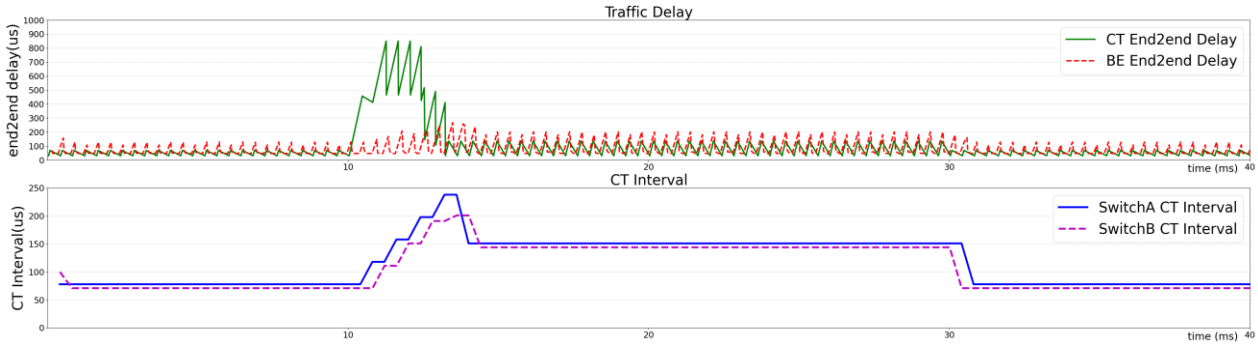
4.5 半实物仿真结果分析

本小节将所提出的调度算法运行在第三章开发的半实物仿真系统，进行性能验证。按照

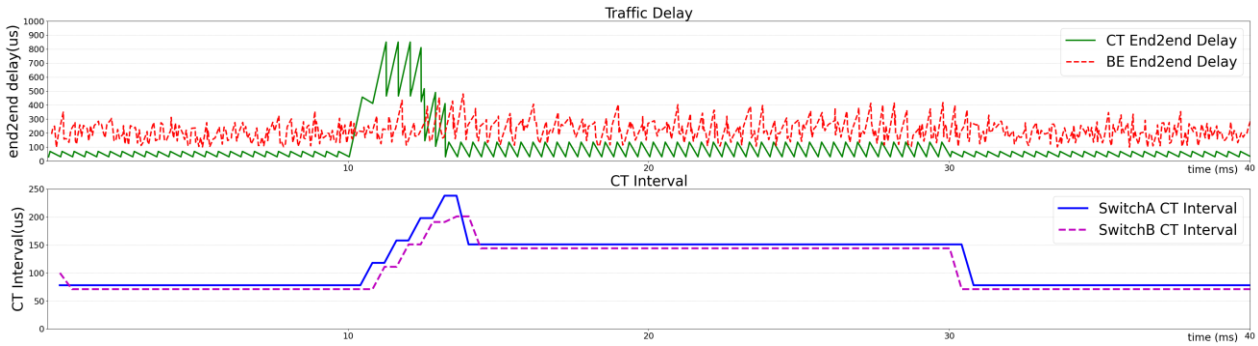
图 4.3 所示的多交换机网络拓扑搭建半实物仿真网络。其中利用 Ostinato 软件和 NestingExtupperNIC 模块共同构成ES_{12}，向ES_{15}发送 BE 流，其他实验参数设置与上文中相同。

在 0 至 10ms 期间，CNC 控制 CT 时隙减小至合理值，由于网络中没有新增流量，CT 时隙保持不变。从 10ms 开始网络中 CT 流的数量增加，当前 CT 时隙无法满足其传输，流量时延增大。CNC 选择网络中权重系数最大的交换机出端口，增大其 CT 时隙并将该端口的权值系数置为 0。若调节完成后，网络中依然存在 CT 流时延超过触发时隙调节上限，此时再次选择网络中权重系数最大的交换机出端口，调节其 CT 时隙。如此往复，直到网络中所有 CT 流时延均小于触发时隙调节上限。从 30ms 开始网络中 CT 流的数量减少，当网络中没有 CT 流时延超出触发时隙调节上限时，CNC 控制时隙窗口减小，为 BE 流提供尽可能大的带宽。

图 4.7 为调度算法在软件仿真系统与半实物仿真系统的结果对比，其中图 4.7(a)为软件仿真系统实验结果，图 4.7(b)为半实物仿真系统实验结果。在软件仿真系统中，CT 流和 BE 流均由软件进行仿真生成，仿真流量严格按照预定的时间发送。在半实物仿真系统中，CT 流由软件进行仿真生成，BE 流由真实流量进行映射。利用真实设备生成的流量充当 BE 流，在真实流量映射至仿真流量的过程中，会引入一定的时延和抖动，半实物仿真系统中的 BE 流时延比软件仿真系统中相对较大。而 CT 流的传输不受 BE 流的传输影响，因此两个系统中 CT 流的表现一致。



(a) 软件仿真



(b) 半实物仿真

图 4.7 调度算法在软件仿真系统与半实物仿真系统的结果对比

表 4.4 为半实物仿真系统与软件仿真系统中的流量时延统计。其中，在半实物仿真系统中，CT 流由仿真生成，BE 流由真实程序生成。因此半实物仿真环境和软件仿真环境下的 CT 流时延表现结果一致，平均值均为 82.79μs。而由于真实流量到仿真流量映射的过程中会引入一定的时延和抖动，因此半实物仿真环境下 BE 流时延和抖动相对较大，平均时延为 209.98μs。通过半实物仿真实验，对本文所提出的算法进行更进一步的仿真验证，实验结果证明本文所提出的算法不仅可以在网络动态变化的情况下优先保障 CT 流的传输，还可以减少网络中的抖动

表 4.4 半实物仿真与软件仿真流量时延统计（单位μs）

	CT 流				BE 流			
	平均值	标准差	最大值	最小值	平均值	标准差	最大值	最小值
软件仿真	82.79	111.79	850.22	30.22	69.0	36.65	267.62	46.92
半实物仿真	82.79	111.79	850.22	30.22	208.98	63.3	479.31	97.02

4.6 本章小结

在本章中，针对 TSN 中网络动态变化的应用场景，提出了一种基于自适应时隙窗口的调度算法。首先对 TSN 流量调度问题进行分析建模，然后在传统 ASW 算法的基础上，对被调节端口的选择方式和时隙窗口的调节方式进行优化，使其满足复杂网络场景。最后，在软件仿真环境和半实物仿真环境下设计对比实验，验证本文所提算法的可行性。结果表明，在网络动态变化的情况下，本算法可以尽可能保障 CT 流的传输，同时为 BE 流提供尽可能大的带宽，还可以有效减小由于流量变化造成的网络抖动。

第五章 总结与展望

5.1 工作总结

TSN 通过数据链路层的一系列协议，在传统非实时的以太网基础上，为时间敏感流提供低时延、低抖动、高可靠的通信。同时兼容非时间敏感流的传输，已成为工业互联网等众多领域的重要支撑技术。TSN 通过单个协议或者不同协议的组合，以满足不同类型流的 QoS 需求。目前 TSN 仍处于发展阶段，可以利用仿真软件实现 TSN 网络性能的快速验证。本文对 TSN 中的流量调度算法以及 TSN 半实物仿真展开了研究，主要完成以下工作：

（1）为了解决 TSN 半实物仿真问题，本文利用 OMNeT++仿真软件和 NeSTiNg 仿真框架设计开发了一套 TSN 半实物仿真系统。首先将半实物仿真问题分为实时调度机制和半实物仿真接口并分别设计开发。其次，为了满足真实设备和真实程序的半实物仿真需求，本文设计开发了两种不同类型的半实物仿真接口。然后，对本文所设计半实物仿真接口进行可行性验证，并利用真实流量到仿真流量映射过程中引入的延迟，对本文所设计的半实物仿真接口性能进行验证。最后，利用 Linux 操作系统和 TSN 交换机，搭建真实物理实验系统，并通过真实物理系统、软件仿真系统以及半实物仿真系统中的实验结果对比，对本文所设计的半实物仿真系统性能进行验证。实验结果表明，本文提出的半实物仿真系统能够实现 TSN 网络的半实物仿真，具有一定的可行性和适用性。

（2）为了解决网络动态变化情况下的时间敏感网络流量调度问题，本文提出了一种基于自适应时隙窗口的调度算法。首先，本文建立了 TSN 调度问题模型。其次，在传统的 ASW 算法的基础上，改进被调节端口的选择方法与时隙窗口的调节方法，使该算法可以满足多种不同网络环境。然后，利用 OMNeT++仿真软件和 NeSTiNg 仿真框架，设计单交换机网络和多交换机网络中的对比实验，验证本文所提算法的可行性。最后，利用前文中开发的半实物仿真系统，对所提出的算法进行更贴近真实的仿真。结果表明，本文提出的基于自适应时隙窗口的调度算法能够快速适应多种网络拓扑下的流量变化，可以在网络变化的情况下快速控制时隙窗口变化。它不仅能够保障 CT 流的传输性能，还能有效减小由于流量变化造成的网络抖动。

5.2 未来展望

本文研究内容仍有进一步拓展的空间。本文在 OMNeT++仿真软件中利用 NeSTiNg 仿真框架实现了 TSN 的半实物仿真，但与现有成熟的 TSN 半实物仿真系统相比，仍存在一些差距。首先，在真实流量与仿真流量映射的过程中会引入一定的时延和抖动，这对于时延要求高的 CT 流仿真并不理想。然后，与真实设备的半实物仿真接口相比，使用真实程序的半实物仿真接口会引入更大的时延和抖动。最后，为了保证半实物仿真实时调度机制的准确性，目前的实验是在同一台设备中使用虚拟网络设备来模拟真实设备。针对这些问题，可以考虑以下几个方面的解决方法。首先，可以利用 PF_RING[78]提供的高性能数据帧处理能力和共享内存的方式，以实现更高的吞吐量和更低的时延。其次，可以利用 ptp4l 和 phc2sys 实现不同设备的高精度时间同步，以满足在不同设备中实现半实物仿真时实时调度的需求。

在本文中，通过软件仿真和半实物仿真实验，验证了本文所提出的基于自适应时隙窗口的调度算法可行性。但目前该算法还停留在仿真阶段，下一步需要在实际的 TSN 交换机和控制器上实现该算法。目前，国防科技大学的 OpenTSN 团队[79]开源了系统源码，计划在此基础上展开后续的实现工作。尽管本文所提出的算法在仿真环境中确保了 CT 流的确定性传输，并为 BE 流提供了尽可能大的带宽。但在真实环境中，TSN 网络可能会面临时间同步失效，网络配置异常等问题，需要对本文所提出的算法进行更进一步的验证。

参考文献

[1] von Arnim C, Drăgan M, Frick F, et al. TSN-based converged industrial networks: Evolutionary steps and migration paths[C]//2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2020, 1: 294-301.

[2] Feld J. PROFINET-scalable factory communication for all applications[C]//IEEE International Workshop on Factory Communication Systems, 2004. Proceedings. IEEE, 2004: 33-38.

[3] Jansen D, Buttner H. Real-time Ethernet: the EtherCAT solution[J]. Computing and Control Engineering, 2004, 15(1): 16-21.

[4] Eveleens J. Ethernet AVB overview and status[C]//SMPTE 2014 Annual Technical Conference & Exhibition. SMPTE, 2014: 1-11.

[5] Makowitz R, Temple C. Flexray-a communication network for automotive control systems[C]//2006 IEEE International Workshop on Factory Communication Systems. IEEE, 2006: 207-212.

[6] Time-Sensitive Networking task group[EB/OL]. http://www.ieee802.org/1/pages/tsn.html.

[7] 蔡岳平,姚宗辰,李天驰.时间敏感网络标准与研究综述[J].计算机学报,2021,44(07):1378-1397.

[8] WG802.1. IEEE Standard for Local and Metropolitan Area Networks — Bridges and Bridged Networks. IEEE Std. IEEE 802.1Q-2014, 2014: 1-1832.

[9] WG802.1. IEEE Standard for Local and Metropolitan Area Networks — Bridges and Bridged Networks — Amendment 25: Enhancements for Scheduled Traffic. IEEE Std. IEEE 802.1Qbv-2015, 2016: 1-57.

[10] WG802.1. IEEE Standard for Local and Metropolitan Area Networks — Bridges and Bridged Networks — Amendment 29: Cyclic Queuing and Forwarding. IEEE Std. IEEE 802.1Qch-2017, 2017: 1-30.

[11] WG802.1. IEEE Standard for Local and Metropolitan Area Networks — Bridges and Bridged Networks — Amendment 34: Asynchronous Traffic Shaping. IEEE Std. IEEE 802.1Qcr-2020, 2020: 1-37.

[12] Seol Y, Hyeon D, Min J, et al. Timely survey of time-sensitive networking: Past and future directions[J]. IEEE Access, 2021, 9: 142506-142527.

[13] Bélanger J, Venne P, Paquin J N. The what, where and why of real-time simulation[J]. Planet Rt, 2010, 1(1): 25-29.

[14] Bello L L, Steiner W. A perspective on IEEE time-sensitive networking for industrial communication and automation systems[J]. Proceedings of the IEEE, 2019, 107(6): 1094-1120.

[15] Ashjaei M, Bello L L, Daneshtalab M, et al. Time-Sensitive Networking in automotive embedded systems: State of the art and research opportunities[J]. Journal of systems architecture, 2021, 117: 102-137.

[16] Avnu Alliance[EB/OL]. https://avnu.org

[17] Adopting Time-Sensitive Networking (TSN) for Automation Systems[EB/OL]. https://www.intel.com/content/www/us/en/developer/articles/technical/adopting-time-sensitive-networking-tsn-for-automation-systems-0.html

[18] The Linux PTP Project[EB/OL]. https://linuxptp.sourceforge.net

[19] TSN Documentation Project for Linux[EB/OL]. https://tsn.readthedocs.io

[20] Bruckner D, Stănică M P, Blair R, et al. An introduction to OPC UA TSN for industrial communication systems[J]. Proceedings of the IEEE, 2019, 107(6): 1121-1131.

[21] Pfrommer J, Ebner A, Ravikumar S, et al. Open source OPC UA PubSub over TSN for realtime industrial communication[C]//2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA). IEEE, 2018, 1: 1087-1090.

[22] Senk S, Nazari H K, Liu H H, et al. Open-Source Testbeds for Integrating Time-Sensitive Networking with 5G and beyond[C]//2023 IEEE 20th Consumer Communications & Networking Conference (CCNC). IEEE,

2023: 1-7.

[23] Fu W, Yan J, Quan W, et al. Fenglin-I: an open-source TSN chip for scenario-driven customization[M]//Proceedings of the SIGCOMM'20 Poster and Demo Sessions. 2020: 60-61.

[24] Le C, Qiao D. Evaluation of real-time ethernet with time synchronization and time-aware shaper using OMNeT++[C]//2019 IEEE 2nd International Conference on Electronics Technology (ICET). IEEE, 2019: 70-73.

[25] Arestova A, Hielscher K S J, German R. Simulative evaluation of the TSN mechanisms time-aware shaper and frame preemption and their suitability for industrial use cases[C]//2021 IFIP Networking Conference (IFIP Networking). IEEE, 2021: 1-6.

[26] Pahlevan M, Obermaisser R. Redundancy management for safety-critical applications with time sensitive networking[C]//2018 28th International Telecommunication Networks and Applications Conference (ITNAC). IEEE, 2018: 1-7.

[27] Libin Z, Hongchen Z, Wenchen X, et al. Research on Automotive TSN network Scheduling Analysis and Simulation[C]//2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA). IEEE, 2023: 1267-1270.

[28] Mariño A G, Fons F, Haigang Z, et al. Loopback strategy for TSN-compliant traffic queueing and shaping in automotive gateways[C]//2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). IEEE, 2021: 47-53.

[29] 汪铮,黄容,吴茂文等.OpenEmulator:一种面向 TSN 芯片验证的联合仿真平台[J].计算机工程与科学,2023,45(03):411-419.

[30] Tindell K W, Burns A, Wellings A J. Allocating hard real-time tasks: an NP-hard problem made easy[J]. Real-Time Systems, 1992, 4(2): 145-165.

[31] Dürr F, Nayak N G. No-wait packet scheduling for IEEE time-sensitive networks (TSN)[C]//Proceedings of the 24th International Conference on Real-Time Networks and Systems. 2016: 203-212.

[32] Pahlevan M, Obermaisser R. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks[C]//2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA). IEEE, 2018, 1: 337-344.

[33] Oliver R S, Craciunas S S, Steiner W. IEEE 802.1 Qbv gate control list synthesis using array theory encoding[C]//2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2018: 13-24.

[34] Falk J, Dürr F, Rothermel K. Exploring practical limitations of joint routing and scheduling for TSN with ILP[C]//2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). IEEE, 2018: 136-146.

[35] 郑宏亮. 周期任务调度技术研究[D]. 电子科技大学, 2018.

[36] Falk J, Dürr F, Rothermel K. Time-triggered traffic planning for data networks with conflict graphs[C]//2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2020: 124-136.

[37] Atallah A A, Hamad G B, Mohamed O A. Routing and scheduling of time-triggered traffic in time-sensitive networks[J]. IEEE Transactions on Industrial Informatics, 2019, 16(7): 4525-4534.

[38] Nasrallah A, Balasubramanian V, Thyagaturu A, et al. Reconfiguration algorithms for high precision communications in time sensitive networks[C]//2019 IEEE Globecom Workshops (GC Wkshps). IEEE, 2019: 1-6.

[39] 冯泽坤,龚龙庆,徐丹妮等.时间敏感网络中基于 ILP 的动态流量均衡调度算法[J].微电子学与计算机,2021,38(06):33-37.

[40] 周阳,陈鸿龙,张雷.时间敏感网络中的动态路由与调度联合优化算法[J/OL].物联网学报:1-22.（预出版）

[41] Abe H, Kawata H, Kubo T, et al. Improvement of accommodation efficiency by TAS scheduling considering jitter caused by transmission period[C]//2023 IEEE 20th Consumer Communications & Networking

Conference (CCNC). IEEE, 2023: 146-151.
[42] Nasrallah A, Thyagaturu A S, Alharbi Z, et al. Performance comparison of ieee 802.1 tsn time aware shaper (TAS) and asynchronous traffic shaper (ATS)[J]. IEEE Access, 2019, 7: 44165- 44181.
[43] Zhao L, Pop P, Craciunas S S. Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus[J]. IEEE Access, 2018, 6: 41803-41815.
[44] Shen R, Zhou Y, Ling Z, et al. Window based Dynamic Scheduling Algorithm in Time Sensitive Networks[C]//2022 China Automation Congress (CAC). IEEE, 2022: 6075-6079.
[45] WG802.1. IEEE Standard for Local and Metropolitan Area Networks — Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. IEEE Std. IEEE 802.1AS, 2011:1-292.
[46] WG802.1. IEEE Draft Standard for Local and Metropolitan Area Networks — Media Access Control(MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Stream Reservation Protocol(SRP) Enhancements and Performance Improvements. IEEE std. IEEE P802.1Qcc/D2.0, 2017: 1-207
[47] Li Z, Wan H, Deng Y, et al. Time-triggered switch-memory-switch architecture for time-sensitive networking switches[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 39(1): 185-198.
[48] Zhang C, Wang Y, Yao R, et al. Packet-size aware scheduling algorithms in guard band for time sensitive networking[J]. CCF Transactions on Networking, 2020, 3: 4-20.
[49] Kobzan T, Blöcher I, Hendel M, et al. Configuration Solution for TSN-based Industrial Networks utilizing SDN and OPC UA[C]//2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2020, 1: 1629-1636.
[50] Said S B H, Truong Q H, Boc M. SDN-based configuration solution for IEEE 802.1 time sensitive networking (TSN)[J]. ACM SIGBED Review, 2019, 16(1): 27-32.
[51] Xie D, Li J, Gao H. Comparison and Analysis of Simulation methods for TSN Performance[C]//IOP Conference Series: Materials Science and Engineering. IOP Publishing, 2020, 768(5): 052061.
[52] 陈敏, 计算机网络. OPNET 网络仿真[M]. 清华大学出版社, 2004.
[53] Varga A. OMNeT++[J]. Modeling and tools for network simulation, 2010: 35-59.
[54] Riley G F, Henderson T R. The ns-3 network simulator[J]. Modeling and tools for network simulation, 2010: 15-34.
[55] Pahlevan M, Obermaisser R. Evaluation of time-triggered traffic in time-sensitive networks using the opnet simulation framework[C]//2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). IEEE, 2018: 283-287.
[56] C. Simon, M. Maliosz and M. Mate. Design Aspects of Low-Latency Services with Time-Sensitive Networking[J]. IEEE Communications Standards Magazine, 2018, 2(2):48-54.
[57] P. Heise, F. Geyer and R. Obermaisser. TSimNet: An Industrial Time Sensitive Networking Simulation Framework Based on OMNeT++[C].2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2016:1-5.
[58] Time-Aware Shaper (TAS) implemented in ns-3[DB/OL]. https://github.com/DenKrysos/Time-Aware-Shaper-TAS-in-ns-3
[59] Steinbach T, Kenfack H D, Korf F, et al. An extension of the OMNeT++ INET framework for simulating real-time ethernet with high accuracy[C]//Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques. 2011: 375-382.
[60] Falk J, Hellmanns D, Carabelli B, et al. NeSTiNg: Simulating ieee time-sensitive networking (tsn) in omnet++[C]// Proceedings of IEEE NetSys. IEEE, 2019: 1-8.
[61] Mészáros L, Varga A, Kirsche M. Inet framework[J]. Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem, 2019: 55-106.
[62] Häckel T, Meyer P, Korf F, et al. SDN4CoRE: A Simulation Model for Software-Defined Networking for

Communication over Real-Time Ethernet[J]. Proceedings of 6th International OM, 2019, 66: 24-31.

[63] Böhm S, Kirsche M. Looking into hardware-in-the-loop coupling of omnet++ and rosenet[J]. arXiv preprint arXiv:1509.03558, 2015.

[64] Nirav G, Sivalingam K M. Dynamic routing framework for OMNeT++ based Hardware-In-The-Loop (HITL) network simulation[C]//2014 Twentieth National Conference on Communications (NCC). IEEE, 2014: 1-6.

[65] 汤淼. 大规模 Ad Hoc 网络智能组网技术的研究与仿真[D].北京邮电大学,2021.

[66] 李自州. 大规模智能组网技术的仿真设计与性能分析[D].北京邮电大学,2021.

[67] Kölsch J, Heinz C, Schumb S, et al. Hardware-in-the-loop simulation for Internet of Things scenarios[C]//2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES). IEEE, 2018: 1-6.

[68] Sun L, Li G. Research of Real-Time Performance in HILS Based on RTX and OMNeT++[C]//Proceedings of the 5th International Conference on Computer Science and Software Engineering. 2022: 185-189.

[69] 平震宇. Libpcap 数据包捕获机制剖析与研究[J]. 信息网络安全, 2008 (8): 37-39.

[70] Scussel A A, Panholzer G, Brandauer C, et al. Improvements in omnet++/inet real-time scheduler for emulation mode[J]. arXiv preprint arXiv:1509.03105, 2015.

[71] Biederman E W, Networx L. Multiple instances of the global linux namespaces[C]//Proceedings of the Linux Symposium. Citeseer, 2006, 1(1): 101-112.

[72] open62541 Realtime OPC UA PubSub Publisher[DB/OL]. https://github.com/open62541/open62541/tree/master /examples/pubsub_realtime

[73] Vila-Carbó J, Tur-Masanet J, Hernandez-Orallo E. An evaluation of switched ethernet and linux traffic control for real-time transmission[C]//2008 IEEE International Conference on Emerging Technologies and Factory Automation. IEEE, 2008: 400-407.

[74] Reusch N, Zhao L, Craciunas S S, et al. Window-based schedule synthesis for industrial IEEE 802.1 Qbv TSN networks[C]//2020 16th IEEE International Conference on Factory Communication Systems (WFCS). IEEE, 2020: 1-4.

[75] Su C, Yu H, Zhang J, et al. Adjustable Window Based Online Scheduling in Mobile Fronthaul[C]//2021 Asia Communications and Photonics Conference (ACP). IEEE, 2021: 1-3.

[76] Li Y, Jiang J, Hong S H. Joint traffic routing and scheduling algorithm eliminating the nondeterministic interruption for tsn networks used in iiot[J]. IEEE Internet of Things Journal, 2022, 9(19): 18663-18680.

[77] Network calculus: a theory of deterministic queuing systems for the internet[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[78] Gallenmüller S, Emmerich P, Wohlfart F, et al. Comparison of frameworks for high-performance packet IO[C]//2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, 2015: 29-38.

[79] Quan W, Fu W, Yan J, et al. OpenTSN: an open-source project for time-sensitive networking system development[J]. CCF Transactions on Networking, 2020, 3: 51-65.