國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master's Thesis

時間敏感網絡中的時序分析與優化

Timing Analysis and Optimization in Time-Sensitive Networking

王仲琦

Chung-Chi Wang

指導教授：林忠緯 博士

Advisor: Chung-Wei Lin, Ph.D.

中華民國 113 年 06 月

June 2024

doi:10.6342/NTU202401726

國立臺灣大學碩士學位論文
口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

時效性網路中的時序分析與最佳化

Timing Analysis and Optimization in Time-Sensitive Networking

本論文係王仲琦君（學號 R11922195）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 113 年 6 月 19 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 19 June 2024 have examined a Master's thesis entitled above presented by CHUNG-CHI WANG (student ID: R11922195) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

林忠緯

（指導教授 Advisor）

張耀文　周詩梵

系主任/所長 Director: 陳祝嵩

國立臺灣大學碩士學位論文
口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

時效性網路中的時序分析與最佳化

Timing Analysis and Optimization in Time-Sensitive Networking

本論文係王仲琦君（學號 R11922195）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 113 年 6 月 19 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 19 June 2024 have examined a Master's thesis entitled above presented by CHUNG-CHI WANG (student ID: R11922195) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

林忠緯 _____ _____ _____

（指導教授 Advisor）

_____ _____ _____

_____ _____ _____

系主任/所長 Director: 陳祝嵩

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to Professor Chung-Wei Lin. His meticulous guidance and feedback have been invaluable to me. His passion for research and integrity have significantly influenced me. Whenever I encountered bottlenecks or difficulties in my research, it was his patient guidance that helped me overcome them. It has been an incredible fortune to be part of the CPS Lab, my time here has been fulfilling and far beyond my expectations.

I also want to extend my thanks to the members of the CPS Lab. Their willingness to help whenever I needed it has been a tremendous support. Having someone to discuss ideas with when I feel stuck has been immensely beneficial.

Lastly, I am deeply grateful to my family. Their unwavering support has allowed me to complete my studies at NTU without any worries. They have always been my strongest pillars of support, and I hope one day become theirs.

Chung-Chi Wang

National Taiwan University
June 2024

doi:10.6342/NTU202401726

時間敏感網絡中的時序分析與優化

研究生:王仲琦 指導教授: 林忠緯 博士

國立台灣大學資訊工程學研究所

摘要

時間敏感網絡（TSN）是由 IEEE 802.1 時間敏感網絡任務組制定的一套標準，它增強了以太網網絡的功能，以滿足實時應用的要求。其基本概念是實現實時通信，確保有界的低延遲、高可靠性和精確的定時。IEEE 802.1Qbv 標準定義了時間感知調整器（TAS）機制，該機制通過根據閘控列表開啟和關閉閘口來調度流量。在時間敏感網絡中，流的延遲不僅受其個別調度的影響，還受其他流的交互和其相應隊列的優先級的影響。因此，在優化過程中需要考慮流與隊列之間的映射。在這項工作中，我們專注於配置閘控列表和流到隊列的映射，以最小化在 IEEE 802.1Qbv 標準下的隊列延遲。我們提出了一種針對時間敏感網絡場景的時間分析方法，其中包括同步和異步流，以及它們的混合組合。我們通過使用模擬退火方法進行結果分析並尋找最優解。我們還提出了在模擬退火過程中增加找到可行解的概率的方法，以及更好的鄰近解選擇策略。結果表明，初始時將閘控列表的所有位設置為 1 可以顯著增加找到可行解的概率。此外，在模擬退火過程中在調整閘控列表之前分階段進行映射可以更有效地導致更好的解。

關鍵詞： 網路排程，時間敏感網絡，非同步流量整形，QoS 服務品質。

doi:10.6342/NTU202401726

TIMING ANALYSIS AND OPTIMIZATION IN TIME-SENSITIVE NETWORKING

Student: Chung-Chi Wang **Advisor: Dr. Chung-Wei Lin**

Department of Computer Science and Information Engineering
National Taiwan University

Abstract

Time-Sensitive Networking (TSN) is a set of standards formulated by IEEE 802.1 Time-Sensitive Networking Task Group, which enhances the functions of Ethernet networks to meet the requirements of real-time applications. The basic concept is to enable real-time communication and ensure bounded low latency, high reliability, and precise timing. The IEEE 802.1Qbv standard defines the Time-Aware Shaper (TAS) mechanism, which schedules traffic by opening and closing the gate according to the Gate-Control List (GCL). In TSN, the latency of flows is influenced not only by their individual scheduling but also by the interactions with other flows and the priority of their respective queues. Therefore, the mapping between flows and queues is a crucial factor that needs to be considered during optimization. In this work, we focus on configuring the gate control list and mapping between flows and queues to minimize the queueing latency under the standard of IEEE 802.1Qbv. We propose a timing analysis method for TSN scenarios, which includes synchronous and asynchronous flows, as well as their hybrid combinations. We conduct an analysis of the results and seek the optimal solution using the SA (Simulated Annealing)

doi:10.6342/NTU202401726

method. We also propose methods to increase the probability of finding a feasible solution during the SA process, along with better neighbor solution selection strategies. The results indicate that initializing all bits of the GCL to 1 at the beginning can significantly increase the probability of finding a feasible solution. Additionally, performing mapping in stages before adjusting the GCL during the SA process can more efficiently lead to better solutions.

Keywords: Traffic Scheduling, Time Sensitive Networking, Asynchronous Traffic Shaping, Quality-of-Service, Simulated Annealing

doi:10.6342/NTU202401726

Table of Contents

doi:10.6342/NTU202401726

doi:10.6342/NTU202401726

List of Tables

doi:10.6342/NTU202401726

List of Figures

doi:10.6342/NTU202401726

doi:10.6342/NTU202401726

Chapter 1

Introduction

1.1 Background

The self-driving car is a new challenging trend that vehicle manufacturers are aiming to implement. Recently, the development of the automotive system has caused an increase in the requirement for timing. Therefore, more functions are added to vehicles, such as streaming services and communication with the surrounding environment. The number of sensors in a vehicle like video cameras, lidar, sonar, etc. has been increasing. However, the traditional Controller Area Network (CAN) bus system cannot meet the high bandwidth and timing requirements demanded by new features. As a result, Time-Sensitive Networking (TSN), a set of real-time Ethernet standards has been proposed and developed to cater to the needs of real-time critical systems. The 802.1 enhancement standards achieve bounded low latency, high reliability, and precise timing through various mechanisms, such as timing and synchronization (IEEE 802.1AS), traffic shaping and scheduling (IEEE 802.1Qbv, IEEE 802.1Qav, IEEE 802.1Qcr), and frame preemption (IEEE 802.1Qbu, IEEE 802.3br). These mechanisms allow critical data to be prioritized and transmitted with minimal latency, guaranteeing reliable network behavior.

doi:10.6342/NTU202401726

1.2 IEEE 802.1Qbv

IEEE 802.1Qbv [5] is one of the key sub-standards in TSN. The IEEE 802.1Qbv time-aware scheduler is specifically designed to provide time-sensitive traffic scheduling and prioritization, thereby ensuring low latency and predictability for real-time applications. The key features in Time-Aware Shaper (TAS) include time-aware queueing settings and time-window-based traffic scheduling. In terms of queueing, flows are allocated into queues with priority. For traffic scheduling, TAS utilizes the Time Division Multiple Access (TDMA) concept and controls the gate switches, known as the Gate Control List (GCL). A GCL includes a sequence of 8-bit numbers, which corresponds to the 8 queues in TAS. As shown in Figure 1.1, when a frame arrives at the switch, it first passes through a priority filter and is then routed to a specific queue based on PCP or stream identification function. Each of the 8 queues is connected to a transmission gate controlled by GCL. All gates are linked to the transmission selection block, and when multiple gates are open simultaneously with frames awaiting transmission, the one with the highest priority will be forwarded to the egress port. IEEE 802.1Qbv ensures no frame can be transmitted during gate close. Additionally, a look-ahead check is performed to ensure the remaining time of the time slot is enough to transmit the whole frame, thus preventing transmission during gate closure periods.

1.3 IEEE 802.1Qbu

Priority queueing can be broadly classified into preemptive and non-preemptive. IEEE 802.3br [6] and IEEE 802.1Qbu [7] define frame preemption. Specifically, IEEE 802.1Qbu is for bridge management components, while IEEE 802.3br is for Ethernet MAC components. A preemptable frame can be divided into multiple fragments for transmission. When a high-priority frame arrives during transmission

doi:10.6342/NTU202401726

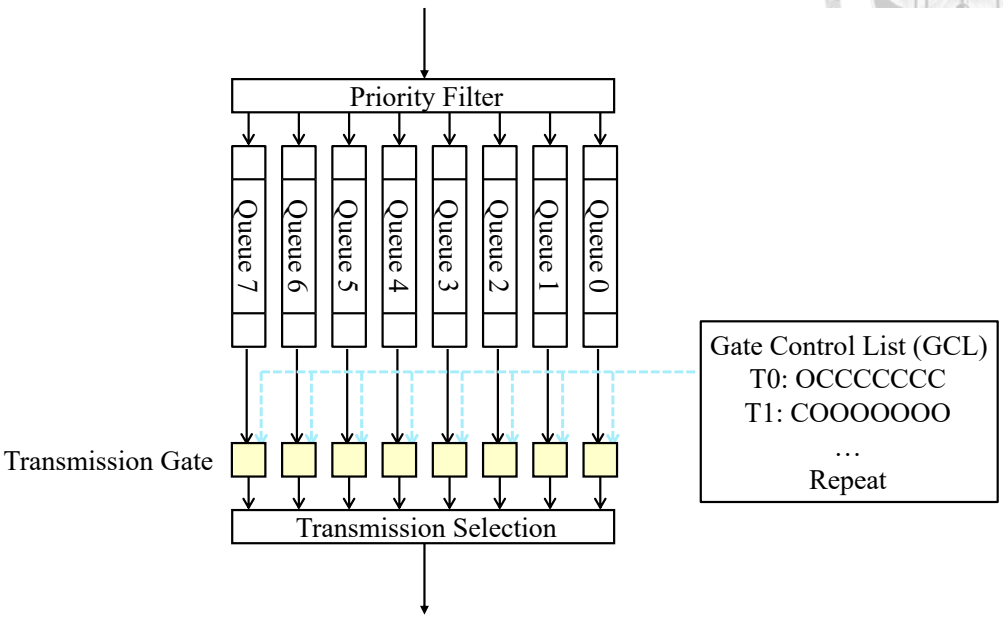


Figure 1.1: The path of a frame through a bridge. GCL controls the opening and closing of the transmission gate on each queue, the frame with the highest priority in the opened gate will be selected to transmit.

of a low-priority frame, it preempts the low-priority frame, and the transmission of the low-priority frame is paused until the high-priority frame is transmitted. Once the transmission of the high-priority frame is completed, the transmission of the low-priority frame resumes. Figure 1.2 illustrates the difference between preemptable and non-preemptable frames during frame preemption. In Figure 1.2a, lower priority frames are continually interrupted and retransmitted. In Figure 1.2b, with preemptable frames, transmission can continue from where it left off even if interrupted.

1.4 Related Work

In order to accurately measure the performance of TSN and ensure timeliness, some timing analysis methods for TSN are proposed. Traditionally, networks are

doi:10.6342/NTU202401726

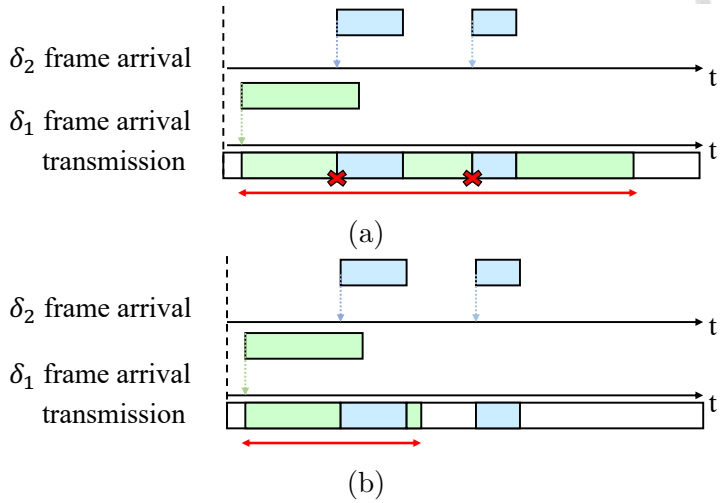


Figure 1.2: Preemptability allows frames to be fragmented when preempted, thus reducing the latency of low-priority frames.

typically analyzed for worst-case end-to-end communication latencies using methods such as network calculus. For instance, Zhao et al. propose a method for timing analysis of IEEE 802.1Qbv utilizing network calculus [14]. Their approach computes the worst-case end-to-end latency based on the given network topology, Gate Control List (GCL), and flow priority.

Thangamuthu et al. have performed an analysis and comparison of the worst-case end-to-end latencies of TAS, BLS, and Peristaltic shaper (PS) [12]. Similarly, Thiele et al. proposed a set of formal timing analysis methods for TSN with TAS and PS to compare the two shapers [13]. Different from analysis in [12], the authors consider the same-priority traffic stream blocking effect during analysis.

In the TAS scenario, the aspects where scheduling can occur include priority assignment, GCL design, path selection, and so on. Consequently, some studies seek optimal scheduling configurations under existing TAS rules. For example, Craciunas et al. address the scheduling problem in IEEE 802.1Qbv multi-hop fully switched TSN networks [1]. The scheduling for the 802.1Qbv-capable network is considered

doi:10.6342/NTU202401726

Satisfiability Modulo Theories (SMT) and Optimization Modulo Theories (OMT) problem. In their work, frame isolation and flow isolation are defined to avoid uncertainties caused by synchronization differences or frame loss. Additionally, Oliver et al. formalize the synthesis process using the first-order theory of arrays and establish an SMT model based on windows defined by opening and closing times to map flows to queues [11]. Furthermore, Dürr et al. formulate the No-Wait Packet Scheduling Problem (NW-PSP) for modeling the scheduling in TSN and map it to the No-Wait Job-shop Scheduling Problem (NW-JSP) which is a NP-Hard problem [2]. The authors formulated NW-PSP as Integer Linear Programs (ILP) and presented a Tabu search algorithm for efficient computation.

Some studies propose further scheduling rules within the framework of TAS. For instance, Hellmanns et al. compare stream-based scheduling, class-based scheduling, and frame preemption by simulation and provide a formula for class-based TAS configuration [4]. Similarly, Heilmann et al. proposed a queuing strategy called Size-Based Queuing (SBQ) to improve bandwidth without impacting the TAS transmissions for the high-priority messages [3]. Moreover, Kim et al. proposed a scheduling method for traffic with the highest priority that occurs without prior notice [8]. The authors add a priority class called emergency event traffic (EV traffic) to the TAS scheduling algorithm.

1.5 Contributions

The main contributions of this paper are summarized as follows:

- We propose methods for analyzing the queueing latency of synchronous and asynchronous flows and present methods to reduce computation time.
- We introduce SA to optimize GCL settings and mappings between flows and

doi:10.6342/NTU202401726

queues.

- We compare maximum and average queueing latency as objective functions to find optimal GCL settings and mappings.
- We propose techniques for initializing and searching for neighboring solutions during the search of the solution space.

The chapter in this thesis is organized as follows. Chapter 2 provides the system model and problem formulation. Chapter 3 introduces the timing analysis methods. Chapter 4 introduces the simulated annealing approaches. Chapter 5 presents the experimental results. Chapter 6 summarizes the conclusion.

doi:10.6342/NTU202401726

Chapter 2

System Model and Problem Formulation

A typical TAS model includes eight queues, each queue has a gate. A frame is considered to transmit only when the gate of its queue is open. In each timeslot, if multiple queues are eligible for transmission, which means the gate is open and there are frames in the queue, the frame in the highest priority queue is considered. Specifically, different priority queues determine whether the gate should be opened or closed based on the GCL. During the egress phase, transmission is allowed when the gate is open, there are frames in the queue, and the remaining time in the time slot is enough time to send the entire frame. Multiple gates can be simultaneously opened, and when multiple queues desire transmission, it is the queue with the highest priority that will be served.

The gate of each queue is controlled by a Centralized Network Controller (CNC) in the form of GCL. Essentially, a GCL is a list that is associated with the statement of gates. In a GCL, each gate statement contains two variables: GateState and TimeInterval. GateState uses 8 bits to indicate the statement of the 8 queues, the control passes to the next gate operation after TimeInterval ticks. When the gate control list reaches the end of the list, it cycles back to the beginning for execution. The notation used in the problem formulation is listed in Table 2.1.

We consider a set of network traffic flows $\Phi = \phi_1, \phi_2, ..., \phi_n$. Every flow

doi:10.6342/NTU202401726

Table 2.1: The notations of indices, elements, sets, quantities, constant parameters, real variables and binary variable (the binary variable value is 1 if the condition is true).

g	index of priority
i	index of flow
j	index of frame
k	index of open time slot
f	index of fragment
$\tau_{i,j}$	$j \in [1, \infty)$, the j-th frame of ϕ_i
$\tau_{i,j,f}$	the f-th fragment of $\tau_{i,j}$
δ_g	queue for storing Γ_g frames
ϕ_i	the flow with index i
Γ_g	set of flows with priority g, $g \in [0, 7]$
Φ	$\Phi = \phi_1, \phi_2, ..., \phi_n$, the set of network traffic flows
m	number of frames in a round
n	number of time slots in a round
P_i	the period of flow ϕ_i
O_i	the offset of flow ϕ_i
T_i	transmission time of the frame in ϕ_i
L	length of a time slot
$a_{i,j}$	the arrival time of $\tau_{i,j}$
$v_{i,j,f}$	the starting time of serving $\tau_{i,j,f}$
$v_{i,j}$	the starting time of serving $\tau_{i,j}$
$q_{i,j}$	queueing latency of the $\tau_{i,j}$
$s_{g,k}$	the starting time of the k-th time slot from δ_g
d_i	the maximum $q_{i,j}$ of ϕ_i
l_i	the length of frames in ϕ_i
$o_{g,k}$	$s_{g,k}$ is open

doi:10.6342/NTU202401726

$\phi_i = (P_i, O_i, T_i)$ is composed of several frames with period P_i and offset (the time the first frame arrives) O_i. T_i is the transmission time of frames in flow ϕ_i.

All flows in Φ have the same destination, and frames are scheduled by the TAS model. Each flow is mapped to a queue. Γ_g, $g \in [0, 7]$ is the set of flows with priority g, and δ_g is the queue for storing Γ_g frames. In every example of this paper, a larger priority number means a higher priority. Every flow with priority g maps to δ_g.

The GCL can be converted to the pattern of timeslot for every queue. In our work, TimeInterval is set to the size of a single time slot. The time slot of δ_g is open when $s_{g,k}.head \times o_{g,k} < t < s_{g,k}.head \times o_{g,k} + L$ at time t. For a frame $\tau_{i,j}$ to be served by the time $v_{i,j}$, there must exist a k such that $s_{g,k}.head \times o_{g,k} \leq v_{i,j} \leq s_{g,k}.head \times o_{g,k} + L$ and either $v_{i,j} + T_i \leq s_{g,k}.head \times o_{g,k}$ or $v_{i,j} + T_i \leq s_{g,k+1}.head \times o_{g,k+1}$. Notably, $s_{g,k}.head$ is the beginning of $s_{g,k}$.

The system follows a preemptive model, aligning with the IEEE 802.1Qbu standard. IEEE 802.1Qbu provides frame preemption and allows lower priority frames to be preemptable, often used with IEEE 802.1Qbv. In our system model, low-priority frames are fragmented into fragments and complete the transmission of remaining fragments after preemption. For instance, suppose $i_1 < i_2$, if a higher-priority τ_{i_2,j_2} arrives during the transmission of τ_{i_1,j_1} and queue δ_2 is open, then τ_{i_1,j_1} is fragmented to $\tau_{i_1,j_1,1}$ which is the transmitted part and $\tau_{i_1,j_1,2}$ which is the remain part. After τ_{i_2,j_2} is transmitted, $\tau_{i_1,j_1,2}$ start the transmission.

For frames larger than one time slot, we assume they are split into multiple fragments, each of which is at most the size of a time slot. These frames have the same enqueue time and deadline but are not required to be transmitted consecutively. For a frame $\tau_{i,j}$, if $T_i \geq L$, $\tau_{i,j}$ will be separate to several frag-

doi:10.6342/NTU202401726

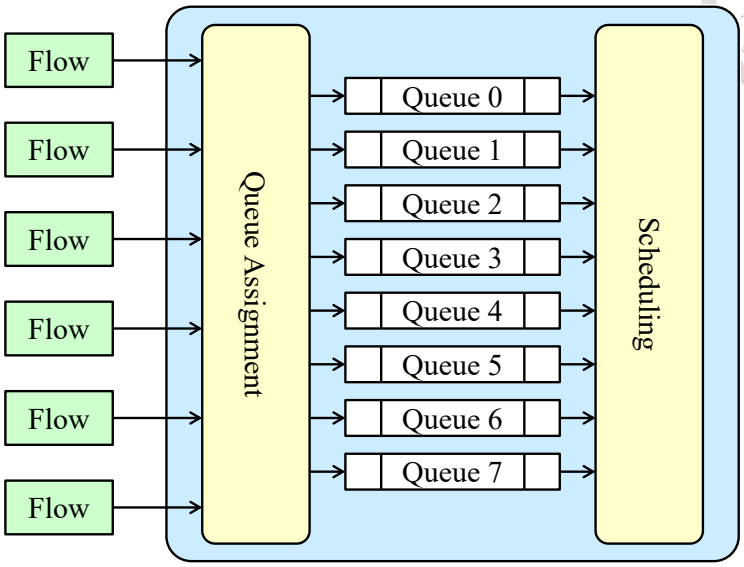


Figure 2.1: Queue assignment and scheduling are used as factors to influence queueing latency in this paper.

ments $\tau_{i,j,1}, \tau_{i,j,2}, \ldots, \tau_{i,j,\lceil \frac{T_i}{L} \rceil}$. Every fragment except for $\tau_{i,j,\lceil \frac{T_i}{L} \rceil}$ has a transmission time of L. Regarding fragmented frames, their queueing time is defined as: $v_{i,j,\lceil \frac{T_i}{L} \rceil} - v_{i,j,1} - T_i + T_{i,j,f}$. Here, L represents the timeslot length, $v_{i,j,1}$ is the start time of transmitting $\tau_{i,j,1}$, T_i is the transmission time of a single frame in Φ.

When analyzing worst-case latency, we consider scenarios for synchronous flows and asynchronous flows. Synchronous flows have a precisely known arrival time throughout the entire network. Conversely, for asynchronous flows, the network only knows the period and frame sizes, but the specific arrival times are unknown for the entire network. As such, we treat asynchronous flows as having an unknown offset before the arrival of the first frame. This offset remains unknown during the design phase, necessitating consideration of the worst-case scenario among all possible offsets during timing analysis.

Given a set of flows $\Phi = \phi_1, \phi_2, ..., \phi_n$ with period and frame transmission time of each flow, we would like to find a solution to minimize the objective function.

doi:10.6342/NTU202401726

As illustrated in 2.1, each flow maps to a queue, and the GCL controls the gate. The queueing latency for each flow should not exceed its period. If any flow violates this constraint, then the solution is considered infeasible. A solution is a combination of queue assignments (mapping between flows and priorities) and a schedule (GCL). In our work, we tried two objective functions, namely the maximum queueing latency for all flows and the average maximum queueing latency of all flows. Our goal is to explore the solution space using SA to minimize the objective function.

doi:10.6342/NTU202401726

Chapter 3

Timing Analysis

The difference between synchronous flows and asynchronous flows is whether O_i is known. While analyzing asynchronous flows, all possible values of O_i need to be considered to calculate the worst-case queueing latency. Therefore, it will be easier to analyze the worst-case queueing time in synchronous flows. As Figure 3.1 shows, with both Figure 3.1a and Figure 3.1b having the same schedule pattern, frame length, and frame period, the pattern of frame transmission varies depending on the different frame arrival offsets.

Definition 1. *A round is the least common multiple (LCM) of all flow schedules and queue schedules' periods.*

Definition 2. *A frame with transmission time T_i just misses a time slot when it arrives $T_i + \varepsilon$ time unit before the time slot ends, ε is a time unit, $\varepsilon \to 0$.*

Definition 3. *The maximum end-to-end latency of a flow should be less or equal to the period of the flow. $d_i + T_i \times 2 \leq P_i$, i.e., each frame $\tau_{i,j}$ must complete transmission before the next frame $\tau_{i,j+1}$ arrives. Else, it is infeasible.*

3.1 Synchronous Flows

For synchronous flows, we only need to consider two rounds for each case to analyze the worst-case latency. [10] proved that in TDMA-based systems only

doi:10.6342/NTU202401726

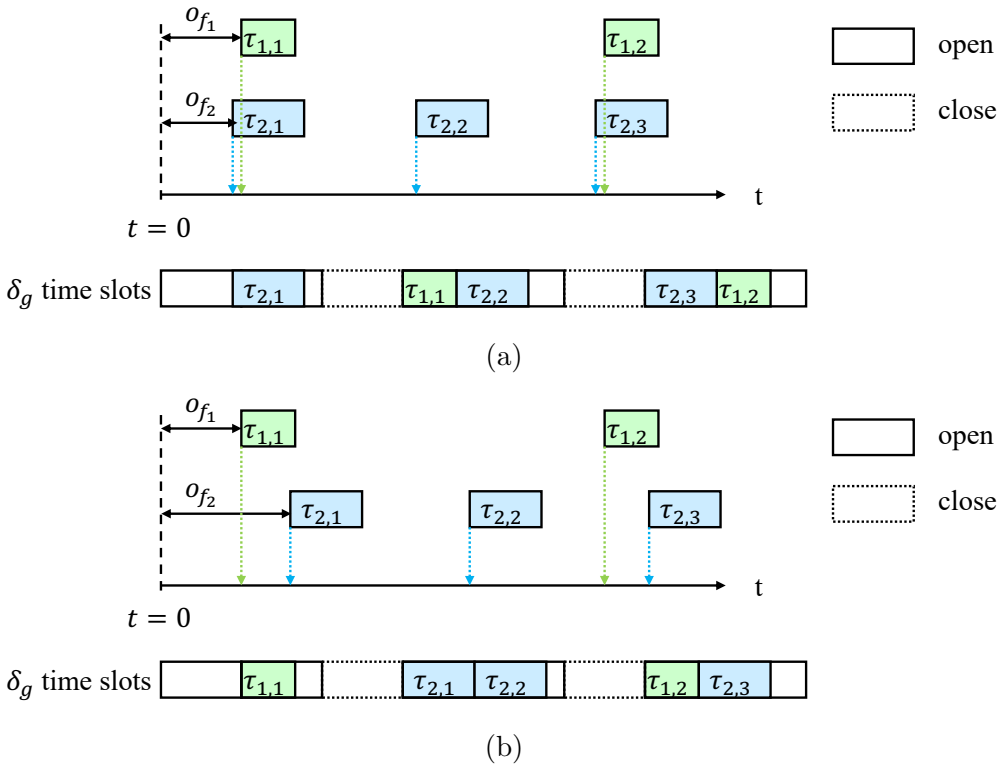


Figure 3.1: Flows with different offsets might have different transmission patterns.

two rounds of analysis are required for each independent queue to determine the maximum queueing latency, and the transmission pattern will repeat after the second round. Although the number of arriving frames remains constant in each round, there may be unscheduled frames remaining after the first round, making subsequent rounds more challenging. Consequently, the quantity of unscheduled frames following the first round is no greater than the unscheduled frames for the second round. This principle extends to subsequent rounds as well. The number of time slots in every round remains constant, so the unscheduled frames after the second should not decrease. Hence, unscheduled frames after the first round are non-decreasing.

Assume the number of frames in a round is never larger than the number of time slots in that round. Consequently, when the first round ends, the number of

doi:10.6342/NTU202401726

unused time slots is greater than or equal to the number of unscheduled frames. In the second round, the repeat frames that are scheduled in the first round can at least fit in the same time slots. Besides, the number of repeated time slots of those unused time slots in the first round is greater than or equal to the number of unscheduled frames in the first round. Therefore, the number of unscheduled frames after the first round is non-increasing. Since the unscheduled frames after the first round are neither decreasing nor increasing, the numbers of unscheduled frames after the first round and the second round are the same. Therefore, the transmission pattern of the second round and the following round are the same.

Next, we will prove that this theory also applies to scenarios with multiple queues. Since independent queues will have the same transmission pattern after the second round, it can be concluded that the highest priority queue, which does not share its time slot with any other queue, can satisfy this condition and have the same transmission pattern after the second round in every round.

For the second highest priority queue, the transmission pattern of the highest priority queue will be the same after the second round in every round. The time slots that will be occupied are determined by the highest priority queue since the system is preemptive. Therefore, the schedule pattern is also determined. With the schedule pattern and arrival pattern determined, it will also remain the same transmission pattern after the second round.

This logic applies to all queues, the available time slot of every queue is unilaterally determined by the queues with higher priority, as shown in Figure 3.2. The lower-priority queue will never affect the higher-priority queue because of preemption. Therefore, the schedule pattern of all queues can be determined. As a result, their transmission pattern will be the same after the second round in every round.

doi:10.6342/NTU202401726

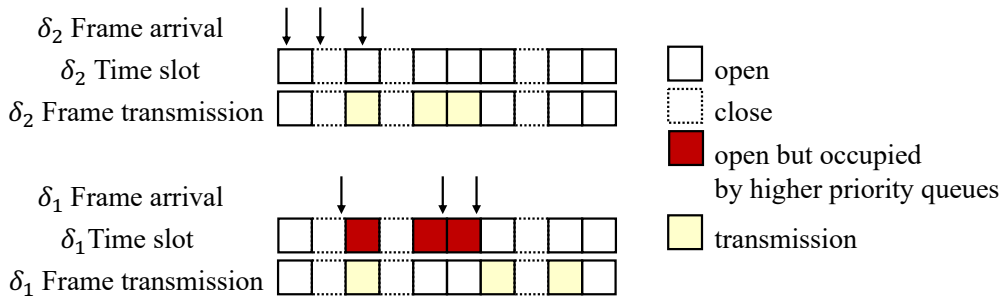


Figure 3.2: A transmission in a higher priority queue will block the same time slots in the lower priority queue.

3.2 Asynchronous Flows

We consider two situations to analyze the maximum queuing latency of asynchronous flows: only one queue is allowed to be open at the same time and multiple queues are allowed to be open at the same time. Both scenarios can be analyzed using methods that allow multiple queues to be open at the same time, however, faster solutions exist for the former. The analysis of asynchronous flows can be viewed as an analysis of a finite set of synchronous flows.

3.2.1 One Queue for a Time Slot

One time slot corresponds to one queue. Since there is no need to consider the priority issue when gates of different queues are opened at the same time, this situation can also be regarded as a simple TDMA analysis problem. According to [10], the worst-case latency of the frame arrival pattern is

$$\max_{1\le h\le m}\left(\max_{1\le k\le n}(s_{g,k+h}-s_{g,k})-\min_{1\le i\le m}(a_{g,j+h-1}-a_{g,j})\right)+L. \tag{3.1}$$

To reduce complexity, the transmission time is set to L. g is the index of the queue. The worst case happens when a frame $\tau_{i,j}$ is assigned to time slot $s_{g,k}$, and the frame $\tau_{i,j-h}$ just misses the time slot $s_{g,k-h-1}$, and there is no unused time slot between $s_{g,k}$

doi:10.6342/NTU202401726

and $s_{g,k-h-1}$. Consider the worst-case scenario for each flow ϕ_i and τ_i, j is always the last frame when entering the queue simultaneously. The worst case happened when the densest set of $h-1$ consecutive message arrivals and the loosest set of h slots, h is a number between 1 and m.

3.2.2 Multi-Queues for a Time Slot

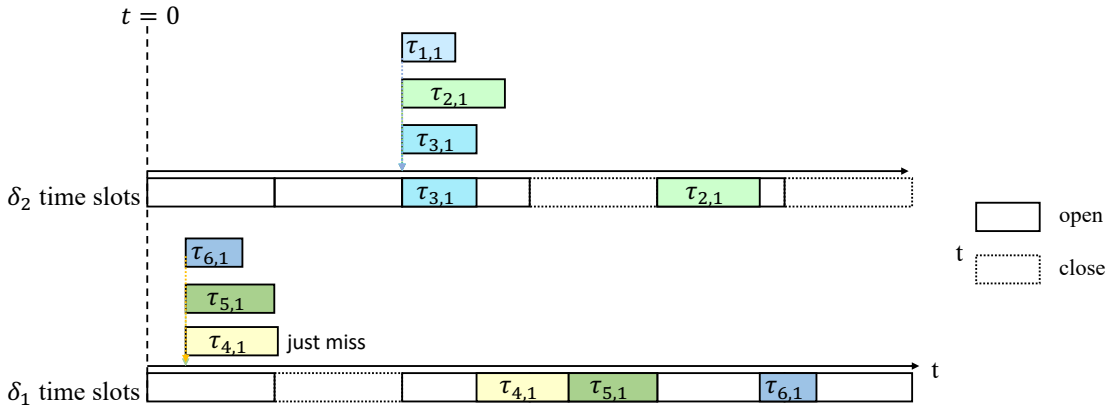
When multiple queues open simultaneously, one queue can have more than one flow. The queuing latency of a frame $\tau_{i,j}$ is the combination of (1) latency from being blocked by the gate close, (2) latency from being blocked by transmissions of frames in the same queue δ_g, (3) latency from being blocked by transmissions of higher-priority frames, and (4) just miss from an open time slot. Since the system is preemptive, only the frames in queues with a priority greater than or equal to g need to be considered when analyzing $\tau_{i,j}$. We find the worst case by assigning different offset combinations to the asynchronous flows. When analyzing a frame $\tau_{i,j}$, where $\phi_i \in \Gamma_g$, we consider latencies caused by:

1. **Just Miss and Gate Close**: Gate close latency can be analyzed with the just miss scenario. When the queue containing $\tau_{i,j}$ has one or more periods of gate closure, the worst case occurs when a just miss happens to $\tau_{i,j}$ or to a frame before $\tau_{i,j}$.

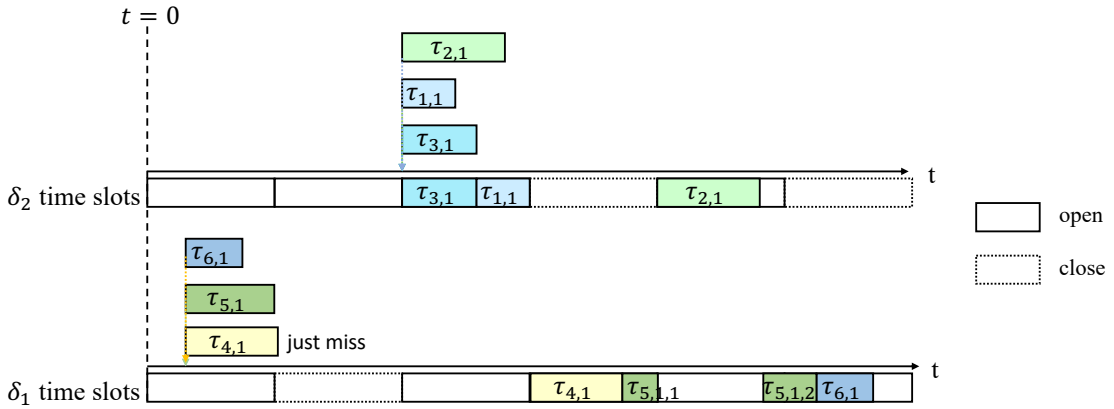
2. **Frames in the Same Queue**: We consider the scenario when every flow in Γ_g have a frame arrive at the same time, and $\tau_{i,j}$ is the last one in the queue.

3. **Frames from Higher Priority Queue**: The transmission of lower-priority frames can be ignored. However, the order of higher-priority frames will affect the latency. During analyzing, when $s_{g,k}$ is open, we open every $s_{g',k}, g < g'$ to

doi:10.6342/NTU202401726



(a)



(b)

Figure 3.3: Different orders of frames in the queue of δ_2 will affect the latency of a frame in δ_1. The transmission order of frames arriving at the same time follows the sequence from bottom to top in the figure.

ensure the order of frames in Γ'_g does not matter and is a pessimistic estimate for $\tau_{i,j}$.

Figure 3.3 illustrates how different transmission orders of frames in δ_2 will affect the latency of frame $\tau_{6,1}$ in δ_1. Figure 3.3a illustrait transmitting a frame $\tau_{3,1}$ from δ_2 first. After the transmission, $\tau_{2,1}$ cannot be transmitted immediately because the remaining time slot is not long enough for it to finish transmission. Next, the time slot $s_{2,4}$ is closed and the time slot $s_{1,4}$ is open so δ_1 transmit $\tau_{4,1}$ and

doi:10.6342/NTU202401726

$\tau_{5,1}$. The transmission of $\tau_{5,1}$ is complete before the gate at δ_2 opens again. At $s_{2,5}$, $\tau_{2,1}$ is transmitted but the remaining time is not enough for $\tau_{1,1}$ to transmit because of the gate close of $s_{2,6}$. Finally, $\tau_{6,1}$ is transmit before $\tau_{3,1}$. In Figure 3.3b, frames $\tau_{3,1}$ and $\tau_{1,1}$ from δ_2 are transmitted first. This different order results in different queuing latencies for $\tau_{6,1}$. δ_1 transmits $\tau_{4,1}$ and $\tau_{5,1}$ after $\tau_{1,1}$. The transmission of $\tau_{5,1}$ is not complete due to preemption, so $\tau_{5,1}$ is fragmented into the transmitted part $\tau_{5,1,1}$ and the untransmitted part $\tau_{5,1,2}$. $\tau_{5,1,2}$ completes transmission after $\tau_{2,1}$ is transmitted. $\tau_{6,1}$ is the last frame to complete transmission.

The worst-case scenario for being blocked by higher-priority queues occurs when $\tau_{i,j}$ is delayed as much as possible by higher-priority frames. In other words, higher-priority frames should be transmitted as early and as densely as possible. Whenever there are gaps between the transmissions of higher-priority frames, it gives frames in δ_g an opportunity to be transmitted. Therefore, the frames in higher-priority queues must be arranged into a sequence that each gate open interval, consisting of consecutive opened time slots, contains the largest possible combination of frame lengths. Selecting the largest possible combination of frame lengths for each gate open interval can be considered a knapsack problem, which is NP-Hard. In Figure 3.3, Figure 3.3b is the optimal order to make $\tau_{6,1}$ delays the most.

In order to perform the analysis in linear time, we open every $s_{g',k}, g < g'$ to delay the $\tau_{i,j}$ transmission as late as possible. Figure 3.5 is an example, Figure 3.5a is the original GCL and Figure 3.5b is the GCL used during analysis. $s_{g'',1}$ and $s_{g'',7}$ are originally closed, but after modification, they will be open because $s_{g,1}$ and $s_{g,7}$ are open. As shown in Figure 3.4, in comparison to Figure 3.3a, opening every $s_{g',k}, g < g'$ can prevent higher priority frames from being interrupted due to insufficient remaining time in that time slot, allowing the frames in δ'_g to be transmitted first. When higher priority queues have more available time slots, the

doi:10.6342/NTU202401726

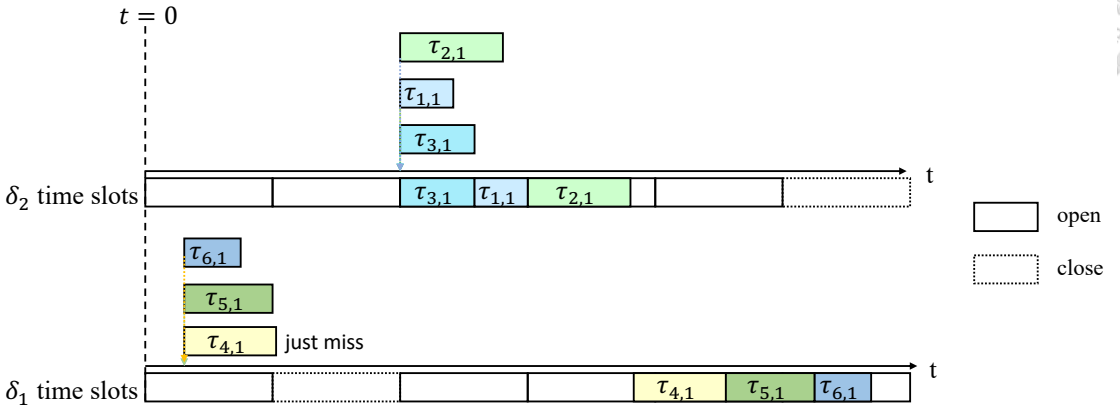


Figure 3.4: Open the time slot $s_{2,4}$ can ensure $\tau_{6,1}$ be postponed as mush as posible.

Figure 3.5: (a) The original GCL. (b) The GCL used during the analysis.

latency of $\tau_{i,j}$ may either increase or remain unchanged, this ensures the estimation is pessimistic for $\tau_{i,j}$.

Combining the above factors will constitute the conditions for the worst case, the worst case for a frame $\tau_{i,j}$ happens when : (1) Every flow in Γ_g has a frame arrive simultaneously and $\tau_{i,j}$ should be the last one to transmit in the queue. (2) The longest frame arrives simultaneously in the same queue except for $\tau_{i,j}$ just misses a time slot $s_{g,k}$. If all time slots are open, consider any time slot. (3) The flows from higher priority queues enqueue at the next open time slot of the queue of ϕ_i after just miss if $s_{g,k+1}$ is closed or enqueue at the same time slot if $s_{g,k+1}$ is opened.

As shown in Algorithm 1 we only need to consider a finite number of trans-

doi:10.6342/NTU202401726

Algorithm 1: Asynchronous Analysis Algorithm

Data: List of time slot in 2 rounds *slots*
Result: *max_latency*

```
max_latency ← 0;
for δ_g in Δ do
    for φ in Γ_g do
        GCL' ← OpenHigherPriorityGates(φ, GCL);
        Γ_g ← shiftToTail(φ, Γ_g);
        for s in slots do
            if isOpen(s) then
                offset_g ← justMiss(φ,s)
                offset_g' ← nextOpenSlot(g, GCL', s);
                sync_max_latency ← synchronousAnalysis(offset_g, offset_g', Φ,
                 φ);
                max_latency ← max(sync_max_latency, max_latency);
```

mission patterns. Consider all possible time slots to just miss for every flow, and use the method of analyzing synchronous flows to find the maximum queueing latency. When analyzing the worst case of frame $\tau_{i,j}$, we only need to consider until $\tau_{i,j}$ is transmitted completely. In other words, there is no need to analyze two rounds as in the method for synchronous flows. This is because when opening every $s_{g',k}$ where $g < g'$, we can ensure that $\tau_{i,j}$ is the last one to be transmitted among frames with the same or higher priority in the current priority level. Therefore, there will be no subsequent frame $\tau_{i,j'}$ with higher latency.

In Figure 3.6, we use two queues for example, and take $\tau_{4,1}$ as the worst case for Φ_4. $\tau_{4,1}$ is the flow transmitted last when multiple frames arrived simultaneously in δ_1. The longest same-queue frame $\tau_{5,j}$ arrives with $\tau_{4,1}$ and just misses, and when the next time slot δ_1 opens, all flow from the higher-priority δ_2, in this example, $\tau_{1,1}$, $\tau_{2,1}$, $\tau_{3,1}$, either arrive simultaneously or with arrival times that allow uninterrupted transmission. Since $\tau_{4,1}$ is the flow transmitted last when multiple frames arrive

doi:10.6342/NTU202401726

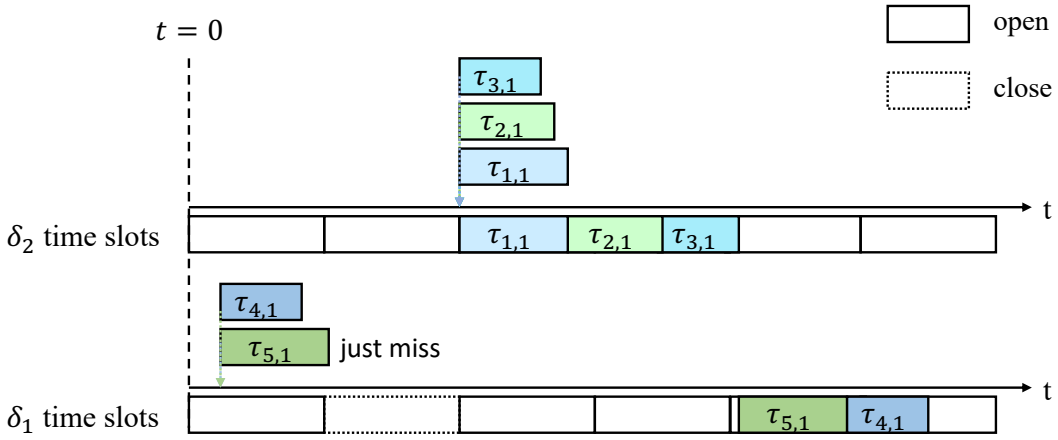


Figure 3.6: Example of a worst-case scenario for asynchronous flows.

simultaneously within the queue, even if there is a sufficiently long time slot available for $\tau_{4,1}$, it cannot transmit when it arrives. $\tau_{1,1}$, $\tau_{2,1}$, and $\tau_{3,1}$ all arrive at $s_{2,3}$, thus blocking $\tau_{4,1}$ and $\tau_{5,1}$, which makes $\tau_{4,1}$ and $\tau_{5,1}$ being the last two frame to transmit.

3.3 Mix of Synchronous and Asynchronous Flows

Analyzing a mix of synchronous and asynchronous flows is similar to analyzing synchronous flows alone. Both involve traversing all flows to analyze worst-case scenarios. The difference of Algorithm 2 lies in considering the fixed offsets of synchronous flows.

When analyzing the worst case for a synchronous flow, it is necessary to consider not just the first frame but all frames within the entire round to consider all the frames from other synchronous flows. Consider the scenario when all the other asynchronous flows in the same queue arrive simultaneously and queue up ahead of this frame. For each frame, two scenarios must be considered: When the frame arrives and has enough remaining time for transmission within the arrival

doi:10.6342/NTU202401726

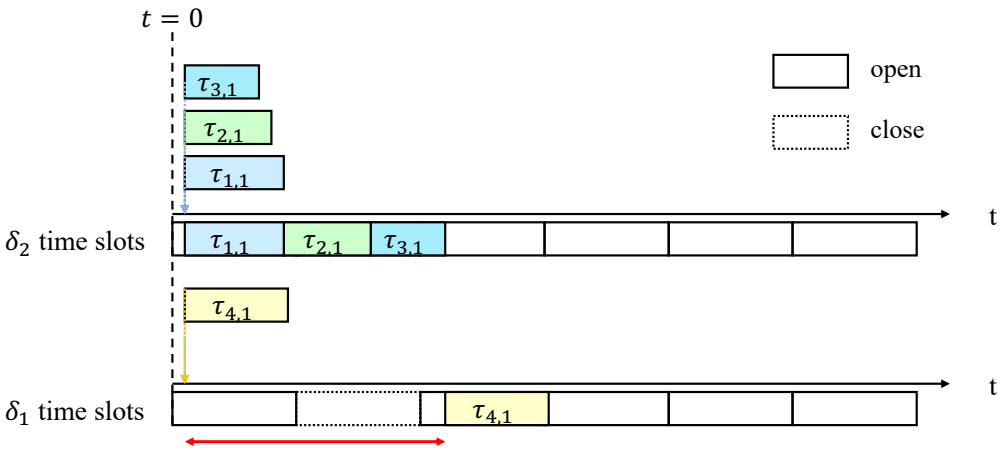


Figure 3.7: $\tau_{4,1}$ have enough remain time in $s_{1,1}$ without considering other queues.

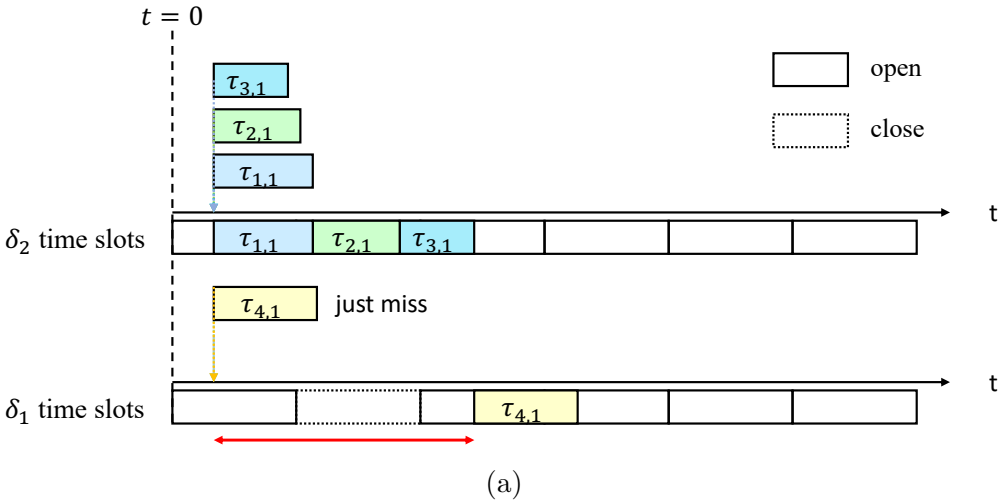
time slot without considering other queues, the worst case of the frame is when synchronous frames arrive simultaneously. Figure 3.7 provides an example.

When the frame arrives and does not have enough remaining time for transmission within the arrival time slot without considering other queues, the worst-case scenario for this frame is as follows: synchronous flow frames from other queues should arrive in the next time slot that this frame can be transmitted, thereby delaying the latency of this frame further. Taking Figure 3.8 as an example, $\tau_{4,1}$ represents a synchronous frame. If, as in Figure 3.8a, it arrives in the same time slot, then a higher priority frame, $\tau_{1,1}$, will be executed when $\tau_{4,1}$ arrives. However, if it arrives in the next time slot when $\tau_{4,1}$ can be transmitted as Figure 3.8b, there will be a gap before $\tau_{1,1}$, $\tau_{2,1}$, $\tau_{3,1}$ arrives, during which no frame can be transmitted. Therefore, it can delay $\tau_{4,1}$ further. When the frame arrives at a closing time slot, asynchronous frames from other queues arrive in the next time slot that the synchronous frame can be transmitted in the worst-case scenario.

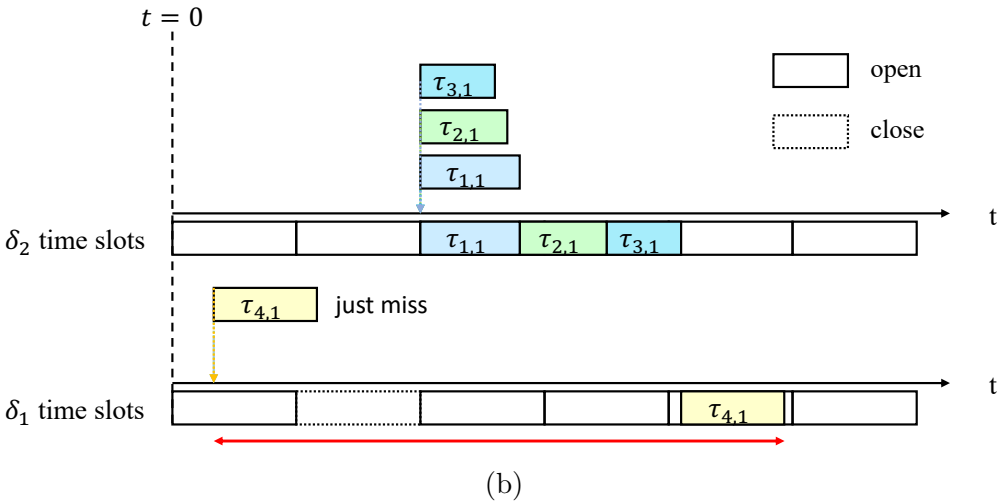
When analyzing the worst case for an asynchronous flow, follow the steps similar to those for analyzing asynchronous flows. In addition to considering all

doi:10.6342/NTU202401726

possible time slots to just miss, also consider the situations when they align with synchronous flows. Figure 3.9 is an example which ϕ_0 is synchronous. The worst case of $\tau_{6,1}$ should consider when $\tau_{4,1}$ just miss $s_{1,2}$, $s_{1,4}$ and when $\tau_{6,1}$ arrive simultaneously with $\tau_{0,1}$.



(a)



(b)

Figure 3.8: $\tau_{4,1}$ does not have enough remaining time at the time slot $s_{1,1}$ without considering other queues.

doi:10.6342/NTU202401726

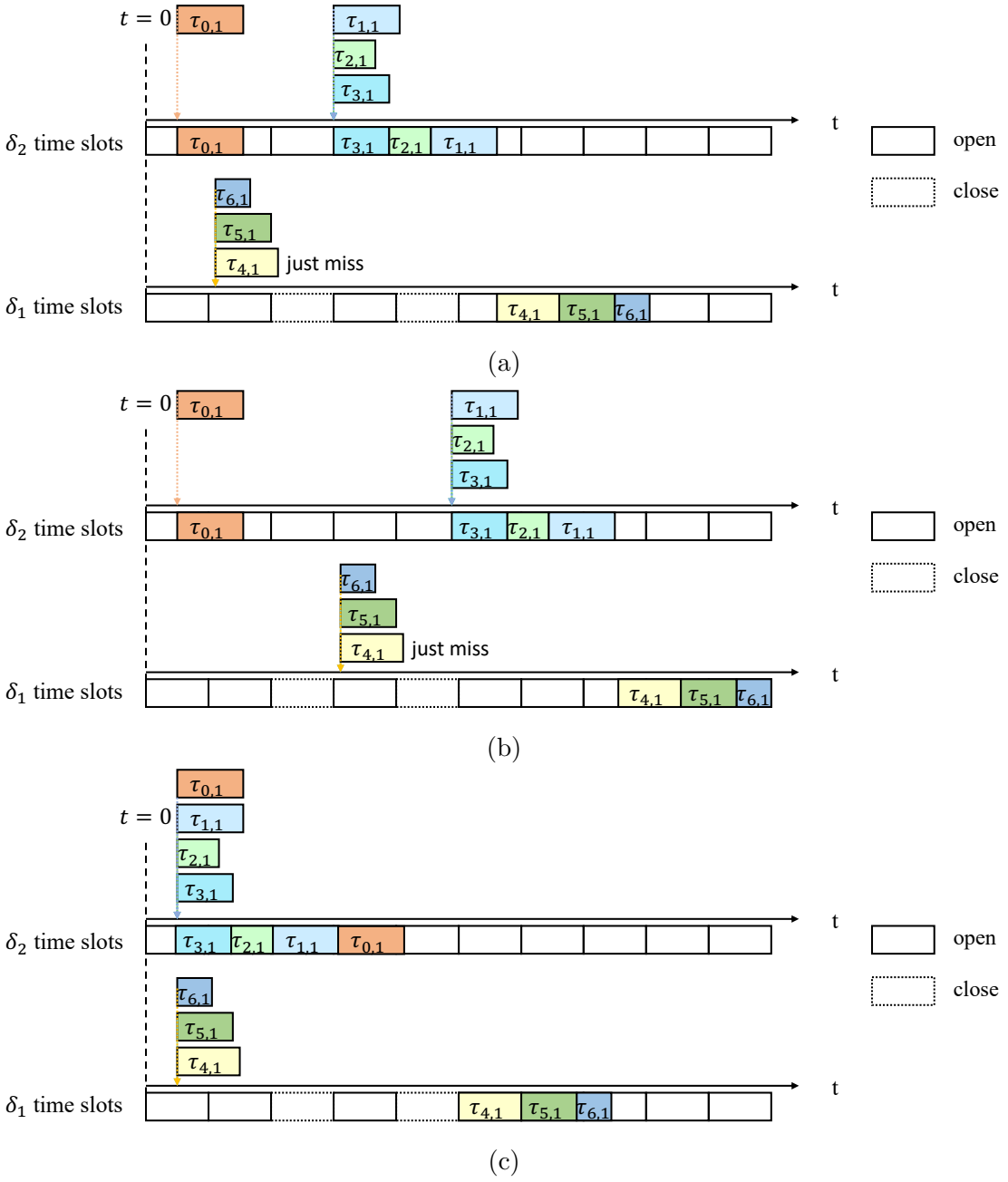


Figure 3.9: The offset combinations to consider when analysing wosrt case for $\tau_{6,1}$. ϕ_0 is synchronous, so $\tau_{0,1}$ has a fix offset.

doi:10.6342/NTU202401726

Algorithm 2: Mixed Asynchronous and Synchronous Flow Analysis Algorithm

Data: List of time slot in 2 rounds *slots*
Data: List of synchronous offsets *offsets*
Result: *max_latency*

```
1  max_latency ← 0;
2  analyzing_time ← 2 × LCM(GCL, Φ);
3  for δ_g in Δ do
4      for φ in Γ_g do
5          GCL' ← OpenHigherPriorityGates(φ, GCL);
6          if isSynchronous(φ) then
7              while offset_g ≤ analyzing_time do
8                  for τ in Γ_g do
9                      if remainTimeEnough(offset_g, GCL') then
10                         offset_g ← φ.offset;
11                         offset_g' ← φ.offset;
12                     else
13                         offset_g ← φ.offset;
14                         offset_g' ← nextOpenSlot(g, GCL', s );
15                     sync_max_latency ← synchronousAnalysis(offset_g, offset_g',
                         Φ, φ);
16                     max_latency ← max(sync_max_latency, max_latency);
17                 offset_g ← offset_g + φ.period;
18         else
19             Γ_g ← shiftToTail(φ, Γ_g);
20             for s in slots do
21                 if isOpen(s, GCL', g) then
22                     offset_g ← justMiss(φ,s)
23                     offset_g' ← nextOpenSlot(g, GCL', s);
24                     sync_max_latency ← synchronousAnalysis(offset_g, offset_g',
                         Φ, φ);
25                     max_latency ← max(sync_max_latency, max_latency);
26             for offset in offsets do
27                 offset_g ← offset;
28                 offset_g' ← nextOpenSlot(g, GCL', offset);
29                 sync_max_latency ← synchronousAnalysis(offset_g, offset_g', Φ,
                     φ);
30                 max_latency ← max(sync_max_latency, max_latency);
```

doi:10.6342/NTU202401726

Chapter 4

Simulated Annealing

To find the configuration of mapping between flows and queues and GCL to minimize the queueing latency, we use Simulated Annealing (SA) [9] as our heuristic method. SA is a technique to approximate the global optimal of a given objective function in the solution space. We anticipate that closely related solutions will produce similar objective values. Capitalizing on this property, we iteratively explore within the solution space. When we come across an improved solution, we keep it and continue to investigate neighboring solutions, while discarding inferior ones. In SA, a worse solution is accepted in certain probabilities, which prevents the exploration process from becoming trapped in a local optimum during the search process and enables us to approach the global optimum as Figure 4.1 shows. This probability decreases

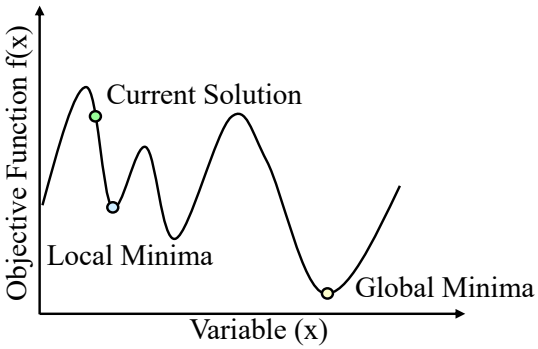


Figure 4.1: When searching for solutions, sometimes it's necessary to temporarily accept suboptimal solutions in order to reach global minima.

doi:10.6342/NTU202401726

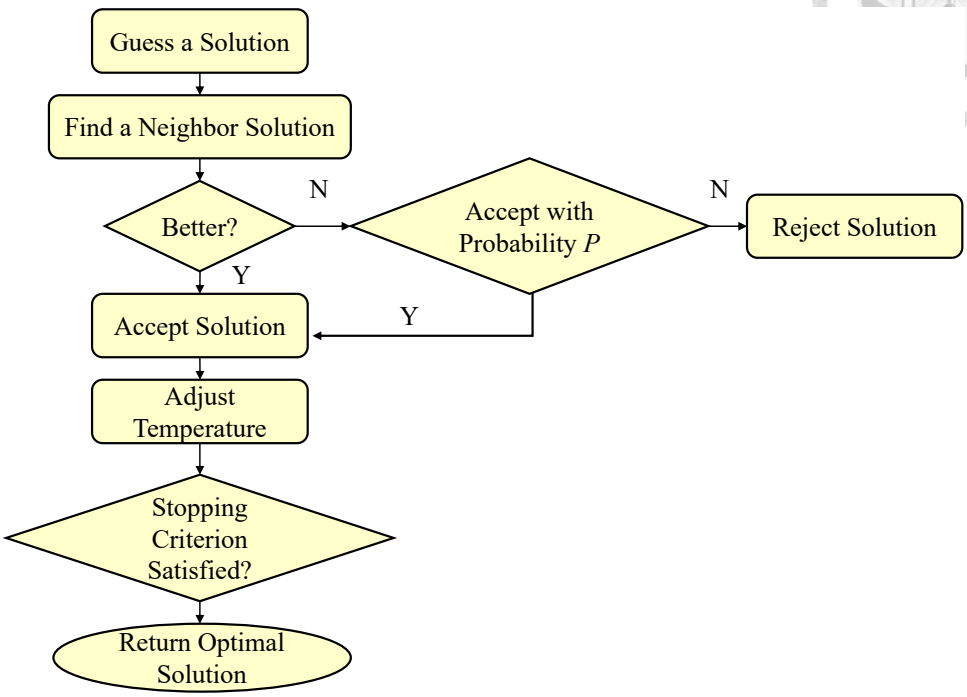


Figure 4.2: The SA process flow chart.

gradually as the number of iterations increases. The process description of SA is depicted in Figure 4.2. The probability P is calculated as

$$P = \exp\left(-\frac{\Delta E}{K_x}\right), \tag{4.1}$$

where K_0 is the initial temperature, and ΔE is the difference in the objective function value between neighboring solutions. The temperature K_x decreases gradually according to

$$K_x = K_0 \cdot A^x, \tag{4.2}$$

where $0 < A < 1$ and x is the iteration number.

4.1 Initial Solution Selection

In the random initial approach, mapping relationships between flows and queues are randomly assigned initially. We compared the method of initializing

doi:10.6342/NTU202401726

each position of the GCL to 1 at the beginning of SA with the random initialization approach. Setting each position of the GCL to 1 is equivalent to employing a scheduling method with priority and preemptable characteristics. The rationale behind this method is that restricting the available time slots for each queue to achieve a reduction in average queueing latency is our goal, which means the GCL should be configured in addition to setting each bit to 1. However, initializing the GCL randomly at the beginning makes it difficult to ensure feasibility for all flows due to excessive gate closures.

4.2 Two-Stage Method

We consider two methods to select neighbor solutions during the process of SA.

1. **Random Method**: For each iteration of finding a neighbor solution, a flow is randomly selected to replace the corresponding queue or change one of the bits in the GCL.

2. **Two-Stage Method**: The entire SA process is divided into two stages. In the early stage, only the mapping relationships between flows and queues are modified when searching for neighboring solutions, while in the later stage, only the configuration of the GCL is modified. To prevent good mapping relationships from being considered infeasible due to gate closures, we keep the GCL fully open during the first stage. The primary motivation of the Two-Stage method is to simplify the optimization by decoupling the complex interdependencies between mappings and the GCL configuration. By initially focusing on optimizing the mapping relationships, we can lay a solid foundation before fine-tuning the GCL.

doi:10.6342/NTU202401726

4.3 Objective Function Settings

In this work, we consider two types of objective functions. One is the maximum queueing latency of all frames,

$$\min(\max(d_{i_{\phi_i \in \Phi}})). \tag{4.3}$$

The other is the average of the maximum queueing latency of all flows,

$$\min(\overline{d}_{i\phi_i \in \Phi}). \tag{4.4}$$

When the solution is infeasible, based on the observed count of infeasible flows or frames, different degrees of infeasibility are distinguished. We anticipate that with a well-designed objective function, the SA process can gradually converge to a feasible solution. The analysis of asynchronous flows involves two nested loops, exploring each possible offset for synchronous flow analysis. We employ the methods of offset-based counting and frame-based counting to calculate the objective function value. Offset-based counting counts the number of infeasible offsets. Frame-based counting counts the number of infeasible frames in the first occurrence of an infeasible offset, we only consider the first offset because of the computational time constraint.

doi:10.6342/NTU202401726

Chapter 5

Experimental Results

For all the SA experiments of this work, 1000 iterations are conducted for each experiment. In the graph of queueing latency, blue dots represent the queueing latency of the solution found at that iteration, while orange dots represent the lowest queueing latency found so far. If there are no dots, it indicates that the solution for that iteration is infeasible.

5.1 Experimental Settings

We extract 61 flows from the Planning-Simulation example in the Autoware Galactic version. The frame lengths in each flow are set by the maximum frame size that is recorded. These 61 flows have periods ranging from 40,000 microseconds to 10,000,000 microseconds and their lengths vary greatly from 8 bytes to 81,080 bytes. We use randomly selected sets of 10, 20, and 30 flows as test data, with smaller sets being subsets of larger ones to ensure increasing difficulty. The SA algorithm was tested with an initial temperature K_0 set to 5000. The cooling schedule followed an exponential decay with a cooling factor A of 0.99. The algorithm is implemented in Python and the experiments were run on a 3.0-GBz processor with 4GB of RAM.

doi:10.6342/NTU202401726

Table 5.1: Average estimate latency and computation time for 1 iteration in fixed configurations of 30 synchronous flows. *inf.* signifies there is no feasible solution due to the bandwidth load exceeding 1.

Period Bound (ns)	10^5	10^6	10^7	10^8	10^9	10^{10}
Computation Time	*inf.*	0.11	0.11	1.13	12.85	64.70
Maximum Queueing Latency of Flow (ns)	*inf.*	502797.0	502797.0	502797.0	502673.0	502673.0
Average Queueing Latency of Flow (ns)	*inf.*	111101.1	111101.1	111101.1	109235.1	109235.1

5.2 Reduce Execution Time

Performing timing analysis directly on the raw data is ideal, but it consumes a significant amount of runtime. This is because the duration of a round is determined by the least common multiple (LCM) of all periods, and when there are too few common factors among them, the length of a round can result in an excessively long execution time. To reduce the LCM, we keep only the highest digit of each flow's period and set the rest to zero to be pessimistic. Additionally, an upper limit is set for the period, and the values exceeding this limit are capped at the specified upper bound. Underestimating the period implies an overestimation of the number of frames, ensuring that the results are pessimistic. We measured the changes in runtime and the estimated worst-case latency for different period bounds in our test case. In Table 5.1, we fix the configuration of queue mapping, flow offset, and GCL in 30 synchronous flows. The experiment was conducted using 6 different configurations, and the results were averaged. As the period bound increases, the runtime also increases, while the estimated values either decrease or remain constant. In the other experiments conducted in this thesis, a period bound of 10^6 was used.

doi:10.6342/NTU202401726

5.3 Main Experimental Results

When there are a large number of flows, infeasible solutions are more likely to occur. Therefore, we assign scores of different infeasibility levels to gradually converge SA to feasible solutions. Table 5.2 shows the probability of finding a feasible solution during the SA process using different methods of calculating the degree of infeasibility in asynchronous cases. In our experiments, feasible solutions were found for all synchronous cases. Without setting each position of the GCL to 1, the method that has the highest probability of finding a feasible solution occurs offset-based counting. This is likely because the number of infeasible frames is highly correlated with the period, and the number of infeasible offsets better reflects the degree of infeasibility.

In our current experiments, when setting each position of the GCL to 1, we are able to find a feasible solution. This result is reasonable because setting the GCL to fully open represents a reduction in the constraints imposed by the closure of time slots for each queue, which aids in finding the first feasible solution. However, since this is based solely on our current experimental data, there may be cases where setting the GCL to fully open does not directly lead to finding a feasible solution. Therefore, we continue to record the outcomes of the different methods for finding feasible solutions as mentioned above.

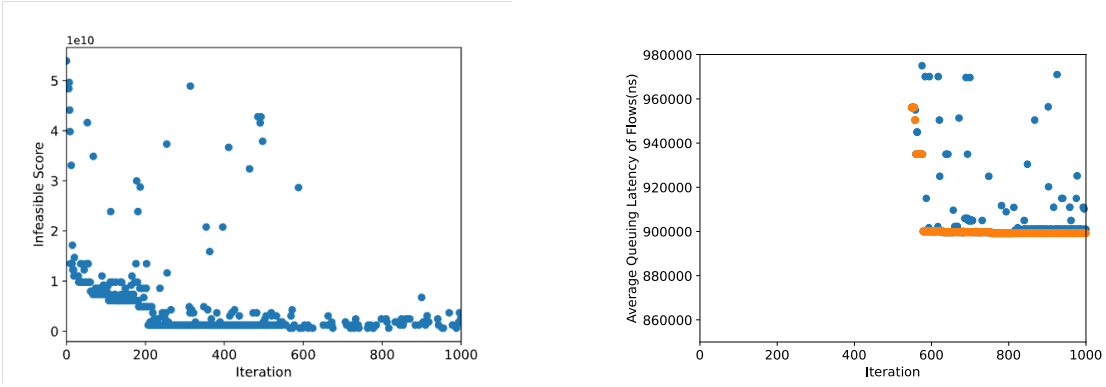
Figure 5.1 illustrates the simulated annealing (SA) process with 30 flows using the offset-based method, where the GCL is initialized randomly. In Figure 5.1a, the infeasible score continuously declines and reaches a low point at around 550 iterations, Figure 5.1b shows the average queueing latencies of feasible solutions, and feasible solutions are found starting from about 550 iterations. This illustrates that after reaching a certain level of reduction, a feasible solution may be found. However,

doi:10.6342/NTU202401726

Table 5.2: Probability of finding feasible solutions.

	Offset-Based	Frame-Based	Open GCL
10 Flows	93%	93%	100%
20 Flows	73%	67%	100%
30 Flows	38%	38%	100%

it is worth noting that achieving a low score does not guarantee the discovery of a feasible solution.



(a) (b)

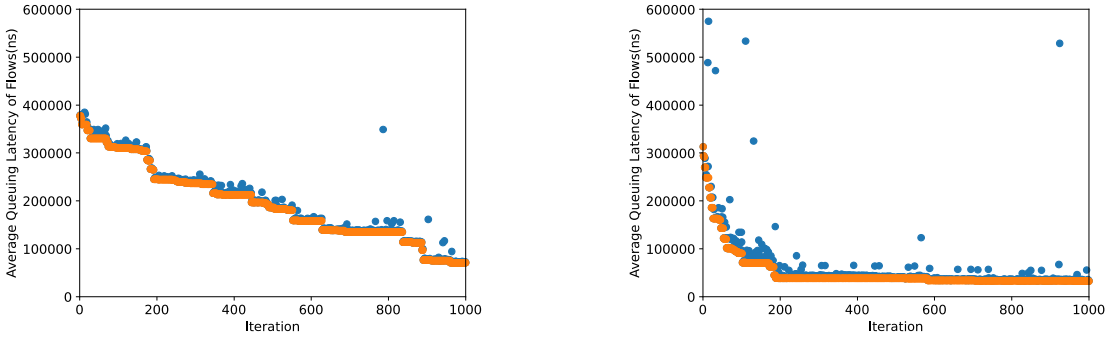
Figure 5.1: (a) Results of the SA process (b) Feasible scores in the SA process

5.4 Results of Two-Stage Method

We compared the queuing latency of the best solutions obtained using the random method and the Two-Stage method during the SA process. Table 5.3 are based on the method of setting each position of the GCL to 1 and using the offset-based method, all SA results have feasible solutions. In our experiment, out of a total of 1000 iterations, the first 200 iterations involve setting the bits of GCL to 1 to search for the optimal flows-queue mapping configuration, constituting the first stage. In this table, "Average" represents the objective function as the average queueing latency of flows, while "Maximum" represents the objective function as

doi:10.6342/NTU202401726

the maximum queueing latency. The subsequent 800 iterations involve adjusting the GCL based on the optimal mapping configuration obtained in the first stage. The Two-Stage method significantly aids in reducing the maximum queueing latency, especially with the objective function of average queueing latency of flows. This suggests that configuring the GCL based on good mapping is more effective than randomly changing the mapping and GCL at each step. Figure 5.2 compares the difference between whether to use the Two-Stage method or not when the objective function is the average queueing latency of flows with 30 flows. While the queueing latency decreases gradually with each iteration in both Figure 5.2a and Figure 5.2b, it is noteworthy that convergence is faster when using the Two-Stage method. The lowest point is reached almost within the first 200 iterations. The modifications to the GCL provide only a small portion of the improvement. This indicates that without using the Two-Stage method, a significant portion of the simulated annealing (SA) process is still spent searching for similar solutions. The Two-Stage method effectively constrains and guides the direction of searching for neighboring solutions during the SA process.



(a) (b)

Figure 5.2: (a) SA Process without Two Stage. (b) SA Process with Two Stage.

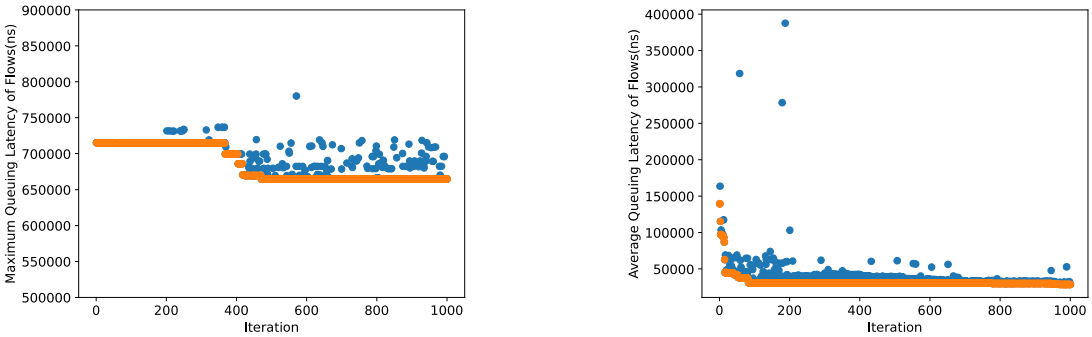
doi:10.6342/NTU202401726

Table 5.3: Experiment result with Two-Stage method on reducing maximum queueing latency.

	Maximum Two-Stage	Maximum w/o Two-Stage	Average Two-Stage	Average w/o Two-Stage
10 Flows	12049.0 (μs)	37955.9 (μs)	3041.9 (μs)	13838.9 (μs)
20 Flows	80047.5 (μs)	103647.4 (μs)	11255.9 (μs)	26995.9 (μs)
30 Flows	609217.2 (μs)	658751.7 (μs)	134375.9 (μs)	162265.0 (μs)

5.5 Results of Objective Function Settings

We employed two objective functions: average queueing latency of flows and maximum queueing latency. Among these, the average queueing latency of flows exhibits a more evident and continuous convergence during the SA process. The primary reason for this may be that using average latency as the objective function increases the likelihood of modifying all flows, as opposed to focusing solely on specific flows when seeking to minimize maximum latency. When using the maximum queueing latency method, which focuses only on one flow, making small changes to either the flow or the GCL may not lead to significant variations. This can make it difficult for each iteration's value to provide a clear direction for SA.



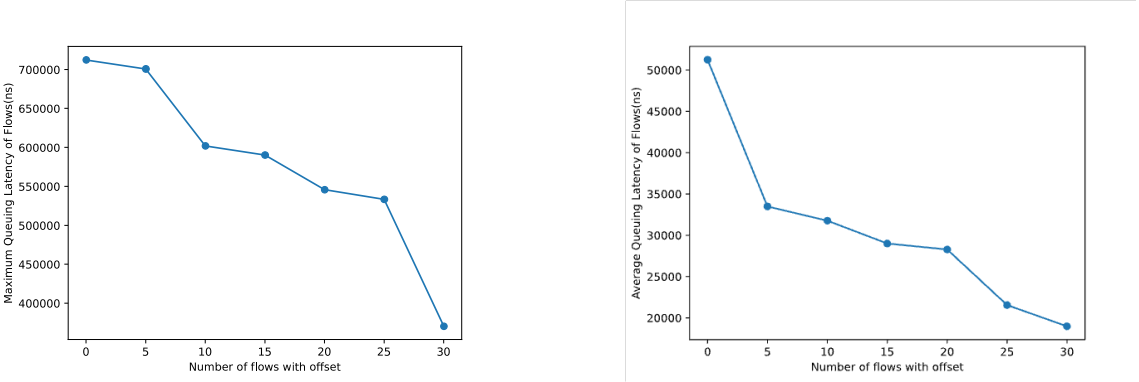
(a) (b)

Figure 5.3: (a) SA Process when the objective function is maximum queueing latency. (b) SA Process when the objective function is average queueing latency.

doi:10.6342/NTU202401726

5.6 The Impact of Number of Synchronous Flow on Estimated Offset Latency

We conducted a mixed analysis of synchronous and asynchronous flows by fixing different numbers of flows with random offsets. Figure 5.4 presents the results of analysis for fixed numbers of synchronous flows among 30 flows. When analyzing 30 flows, where n of them are synchronous flows, each time we randomly select n synchronous flows and assign them random offsets. We perform 1000 iterations of SA and take the best solution. This process is repeated 15 times, and the average result is calculated. The analysis of 30 synchronous flows is conducted using the synchronous analysis method. The results indicate that as the number of flows with fixed offsets increases, the estimated values decrease. This trend suggests that the inclusion of more synchronous flows, with their fixed offsets, contributes to reducing overall latency. This aligns with expectations because with each flow having a fixed offset, there's a significant chance of avoiding simultaneous arrivals with other flows, thereby reducing queueing latency. Therefore, if we can determine the offsets for a subset of flows, we can estimate latency more precisely.



(a) (b)

Figure 5.4: As the proportion of synchronous flows increases, the estimated latency decreases accordingly. (a) The objective function is the maximum queueing latency of flows. (b) The objective function is the average queueing latency of flows.

doi:10.6342/NTU202401726

Chapter 6

Conclusions

In this thesis, we present a timing analysis method for synchronous flows, asynchronous flows, and cases involving a mixture of both in the scenario of TSN. The system model in this thesis is based on the IEEE 802.1Qbv and IEEE 802.1Qbu standards, which respectively define the Time-Aware Shaper (TAS) and frame preemption. In Chapter 5, we demonstrate that analyzing two rounds is sufficient for synchronous flows. Since the duration of analysis is determined by the least common multiple (LCM) of all flow periods, it is necessary to underestimate periods by taking integer values to control computation time. Asynchronous flows, with their variable offsets, can result in an infinite number of frame arrival combinations. To address this challenge, we propose a timing analysis method that constructs the worst-case scenario within linear time.

We employed SA as a heuristic method to search for solutions minimizing queueing latency within the solution space. We utilized both the average queueing latency of flows and maximum queueing latency as objective functions and compared their differences. The initial solutions obtained through SA had a high probability of being infeasible. To guide the SA process toward feasible solutions, we assigned different degrees of infeasible scores. In our case, the offset-based counting method for calculating infeasible scores had the highest probability of finding feasible solutions. Additionally, initializing all bits of GCL as 1 in the initial solution led to

doi:10.6342/NTU202401726

feasible solutions being found in almost all our test cases within 1000 iterations.

Comparing the two methods of altering SA iterations—randomly changing either a flow's corresponding queue or a bit in GCL at each iteration versus dividing the SA process into two stages—we found that the latter approach yielded superior results on average. Therefore, our conclusion is that establishing mapping relationships before fine-tuning GCL is more efficient.

Due to runtime considerations, we narrow down the periods during analysis. In future work, we can improve the timing analysis method to reduce the runtime further. For instance, in asynchronous cases, precluding certain offset combinations beforehand could potentially expedite the process. This can help in achieving more precise estimates.

doi:10.6342/NTU202401726

Bibliography

[1] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks," in *International Conference on Real-Time Networks and Systems*, pp. 183–192, 2016.

[2] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *International Conference on Real-Time Networks and Systems*, pp. 203–212, 2016.

[3] F. Heilmann and G. Fohler, "Size-based queuing: An approach to improve bandwidth utilization in TSN networks," *ACM SIGBED Review*, vol. 16, no. 1, pp. 9–14, 2019.

[4] D. Hellmanns, J. Falk, A. Glavackij, R. Hummen, S. Kehrer, and F. Dürr, "On the performance of stream-based, class-based time-aware shaping and frame preemption in TSN," in *IEEE International Conference on Industrial Technology (ICIT)*, pp. 298–303. IEEE, 2020.

[5] IEEE Standards 802.1, "802.1Qbv—Enhancements for Scheduled Traffic," Draft 3.1, Institute of Electrical and Electronics Engineers, 2016. [Online]. Available: http://www.ieee802.org/1/pages/802.1bv.html

[6] ——, "802.3br-specification and management parameters for interspersing express traffic," Institute of Electrical and Electronics Engineers, 2016. [Online]. Available: https://standards.ieee.org/ieee/802.3br/5814/

doi:10.6342/NTU202401726

[7] ——, "802.1Qbu-frame preemption," Institute of Electrical and Electronics Engineers, 2017. [Online]. Available: http://www.ieee802.org/1/pages/802.1bu.html

[8] M. Kim, J. Min, D. Hyeon, and J. Paek, "TAS scheduling for real-time forwarding of emergency event traffic in TSN," in *International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1111–1113. IEEE, 2020.

[9] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[10] C.-W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, "Security-aware mapping for TDMA-based real-time distributed systems," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 24–31. IEEE, 2014.

[11] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1 Qbv gate control list synthesis using array theory encoding," pp. 13–24, 2018.

[12] S. Thangamuthu, N. Concer, P. J. Cuijpers, and J. J. Lukkien, "Analysis of Ethernet-switch traffic shapers for in-vehicle networking applications," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 55–60. IEEE, 2015.

[13] D. Thiele, R. Ernst, and J. Diemer, "Formal worst-case timing analysis of Ethernet TSN's time-aware and peristaltic shapers," in *IEEE Vehicular Networking Conference (VNC)*, pp. 251–258. IEEE, 2015.

[14] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus," *IEEE Access*, vol. 6, pp. 41 803–41 815, 2018.

doi:10.6342/NTU202401726