
Politecnico di Torino

Politecnico di Torino
Master's Degree in Computer Engineering
Academic Year 2023-2024

# TSN scheduling algorithm for real-time applications in a heterogeneous network

IEEE 802.1Qbv time-aware shaper scheduling algorithm for optimising the estimated end-to-end quality of service to guarantee real-time applications in a heterogeneous network.

**Supervisor**
Prof. Stefano Scanzio

**Co-Supervisors**
Prof. Gianluca Cena
Dott. Gabriele Formis

**Candidate**
Gianluca Graziadei

This master's thesis is part of a double degree project between Politecnico di Torino and Universidad Politécnica de Cataluña Barcelona Tech. It was conducted at the Optical Communications Group under the supervision of Prof. Luis Domingo Velasco Esteban and the co-supervision of Prof. Marc Ruiz Ramírez.

**Abstract**

Time Sensitive Networking (TSN) covers the need for real-time applications, focusing on the Industrial Internet of Things (IIoT), of a low latency deterministic synchronous affordable network. IEEE 802.1Qbv working group defines a guideline to develop a Time-Aware Shaper (TAS) to guarantee Time-division multiple access (TDMA) for different time-critical requests to the network. Starting from the literature, in which the TSN class of traffic is limited to a homogeneous local network, this work aims to develop a TAS scheduler to support heterogeneous networks with interfaces with varying characteristics, such as throughput and propagation delay. This thesis treats TAS scheduling as an optimization problem. It produces two conceptual models, respectively an ILP model and a heuristic one, to maximize the number of accepted requests and the determinism of the network and minimize the latency. The goal is achieved by estimating the jitter and delay of each time-critical transmission. To support the specification of a heterogeneous network, this work implements transformation functions to solve the space and time division problems and a transmission pipeline pattern to mitigate the effect of high propagation delay of a transportation network. Conceptual models are validated with a simulation of a hypothetical scenario where a request traverses both WiFi interfaces and a transport network.

## Acknowledgement

I would like to express my deepest thanks to Prof. Velasco and Prof. Ruiz for allowing me to work on these topics, for their constant and valuable support and for allowing me to join a paper writing work. Additionally, I would like to thank UPC FIB and the GCO research group for hosting me during the final year of my university studies. I also express sincere thanks to Prof. Scanzio for his precious availability to assist me in this work during the period outside Politecnico di Torino.

Thanks to all the people who have supported me during these university years, including my classmates, Erasmus's friends, and lifelong friends. A big thank you goes to my parents, who always believed in me and gave me all possible means to pursue my dreams, and in general to all my family who supported me during these five years. Thanks to Benedetto, whose friendship is always demonstrated to be valid outside the national borders. Finally, thanks to Yajaira, with whom I shared this challenging but beautiful year in Spain, she made me feel at home even in a huge foreign city.

# Ringraziamenti

Desidero esprimere un profondo ringraziamento ai Professori Velasco e Ruiz per avermi offerto l'opportunità di lavorare su questa tesi e per il costante e prezioso sostegno. Inoltre, ringrazio la UPC FIB ed in particolare il team GCO per avermi accolto durante l'ultimo anno del mio percorso universitario. Un sentito grazie anche al Prof Scanzio per la sua preziosa disponibilità nel seguirmi in questo lavoro di tesi al di fuori del Politecnico di Torino. Grazie di cuore a tutte le persone che mi hanno accompagnato in questi anni di università. I miei compagni di corso, i colleghi, gli amici con cui ho condiviso l'Erasmus a Barcelona e gli amici di una vita. Un grosso grazie va ai miei genitori, che hanno sempre creduto in me e mi hanno dato tutti i mezzi possibili per perseguire i miei sogni, ed in generale a tutta la mia famiglia che mi ha sempre supportato. Grazie a Benedetto, compagno di viaggio in questo percorso universitario, la cui amicizia si e' dimostrata valida anche al di fuori dei confini nazionali. Infine, grazie a Yajaira, con la quale ho condiviso questo ultimo, entusiasmante anno in Spagna, facendomi sentire a casa in una metropoli straniera.

# Contents

# List of Algorithms

# List of Tables

# List of Figures

# Acronyms

**AP** Access point. 19

**BE** Best Effort. 17, 97, 109

**BRKGA** Biased Random Key Genetic Algorithm. 22

**CM** TSN Connectivity Manager. 19, 20

**DoE** Design of experiment. 101

**E2E** End-to-end. 19, 33, 49, 61, 71, 81, 95, 96, 105, 106

**GRASP** Greedy randomized adaptive search procedure. 22

**IEEE** Institute of Electrical and Electronics Engineers. 17, 28

**IIoT** Industrial Internet of Things. 1, 17, 81, 89, 105, 108, 109

**ILP** Integer linear programming. 22, 47, 48, 57, 63, 107

**JSSP** Job Shop Scheduling Problem. 47, 48, 107

**LS** Local Search. 22, 77, 100

**LSP** Label-Switched Paths. 17

**MCS** Modulation and Coding Scheme. 88

**MPLS** Multiprotocol Label Switching. 17

**PCE** Path computation element. 19

**QoS** Quality of service. 17, 21, 32, 97, 109

**SF** Super Frame. 18, 21, 25, 27, 31, 70, 72, 75, 79, 86, 88, 97, 99, 106–109, 113

**TAS** Time-Aware Shaper. 1, 14, 18, 20, 22, 23, 27, 38, 47, 48, 69, 70, 76, 82, 87, 89, 105, 107, 108

**TDMA** Time-division multiple access. 1, 18

**TS** Time sensitive, constrained in delay and jitter. 17–19, 23, 31, 42, 59, 70, 72, 76, 79, 81, 87–89, 95, 97, 98, 105–107, 113

**TSN** Time Sensitive Networking. 1, 17–19, 21, 25, 26, 47, 89, 105–109

**UML** Unified Modeling Language. 25, 26

**VLAN** Virtual local area network. 18, 21

**VV&A** Verification, validation and accreditation process. 81

# Chapter 1

# Provisioning of TS, QoS, and BE traffic in transport network

IIoT applications[1] typically have stringent requirements on the dependability and transmission metrics of communication networks. For such applications, it is crucial that the network can provide latency guarantees and maximize the determinism of the transmission. In contrast, legacy Ethernet-based networks can only provide a Best Effort (BE) delivery service. To cope with the lack of determinism in traditional Ethernet and the obscurity of proprietary protocols, a set of amendments about TSN in IEEE 802.1Q standard Ethernet has been specified [3].

End-to-end TSN services entail the support of operators' transport networks that are currently carrying traffic from users, businesses, and datacentres, just to mention a few on a BE basis; such traffic is commonly encapsulated into Multiprotocol Multiprotocol Label Switching (MPLS) Label-Switched Paths (LSP) at Layer 2 for traffic engineering purposes [21].

As soon as operators' transport networks will provide support to TSN traffic to guarantee the committed quality of service of Time sensitive, constrained in delay and jitter (TS) flows and allow for the coexistence of TS, Quality of service (QoS) and BE traffic on the same network infrastructure.

## 1.1 IEEE 802.1Qbv

The Time-Sensitive Networking Task Group is part of the Institute of Electrical and Electronics Engineers (IEEE) 802.1 Working Group. The charter of the TSN group is to provide deterministic services through IEEE 802

[1] Such as industrial surveillance, smart manufacturing, real-time process optimization, predictive maintenance, energy distribution, and infrastructure optimization.

networks[10], thus providing mechanisms that enable low and predictable transmission latency and high availability for demanding applications such as real-time audio/video streaming, automotive, and industrial control.

To provide the required guarantees, TSN integrates different traffic shaping[2] mechanisms including **802.1Qbv**, **802.1Qch**, and **802.1Qcr**, allowing for the coexistence of different traffic classes with different priorities on the same network.

Focusing on IEEE 802.1Qbv TAS, it is designed to separate the communication on the Ethernet network into fixed-length, repeating time cycles, defined Super Frame (SF). The basic concept is a TDMA [3]. Establishing virtual communication channels for specific periods, in the following **scheduling windows**, can separate time-critical communication from non-critical background traffic. The definition of the scheduling window is demanded by each specific protocol implementation, in section 2.6.2 more details about it, the general idea is of an atomic quantum of time in which a network interface gate is assigned to a specific resource, which is identified by a Virtual local area network (VLAN) id.

The IEEE 802.1Qbv time-aware scheduler has to ensure that the Ethernet interface is not busy with the transmission of a frame when the scheduler changes from one time slice into the next, it achieves this by putting a guard band at the edge of every scheduling window that carries time-critical traffic.

### 1.1.1 Frame format

From [10] specifications: 802.1Q header is the 4 bytes field between the source MAC address and the EtherType fields, thus the maximum frame size is extended from 1518 bytes to 1522 bytes. The minimum frame size is 64 bytes. Figure 1.1 shows the schema of the frame. Table 1.1 describes the 802.1Q header fields.

## 1.2 Provisioning of TS and non-TS flows

Paper [20] studies the combinability of multiple TS flows while leaving resources available for non-TS flows. Given the realistic heterogeneous scenario in which flow provisioning requests arrive, each of them needs to be accepted or rejected based on the possibility of providing the required performance as well as ensuring that the performance of already established

[2]The process of controlling the flow of network traffic to ensure a smooth and efficient distribution of bandwidth. It helps in implementing Quality of Service policies by prioritizing certain types of traffic over others. This ensures that critical applications, such as time-sensitive flows, receive sufficient bandwidth and low latency.

[3]The channel access method used in digital communication networks, in which the available transmission bandwidth is divided into time slots, and each station in the network is assigned one or more of these time slots for transmitting data

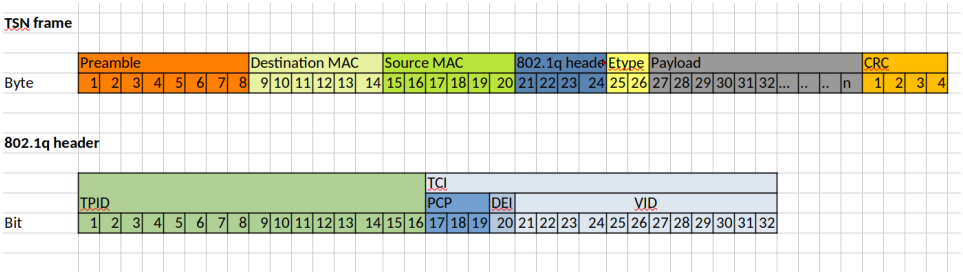



Figure 1.1: IEEE 802.1Q frame

| | |
|---|---|
| **Tag protocol identifier (TPID)** | A 16-bit field set to a value of 0x8100 to identify the frame as an IEEE 802.1Q-tagged frame. |
| ***Tag control information (TCI)*** | |
| Priority code point (PCP) | A 3-bit field which refers to the IEEE 802.1p class of service (CoS) and maps to the frame priority level. |
| Drop eligible indicator (DEI) | A 12-bit field specifying the VLAN to which the frame belongs. |
| VLAN identifier (VID) | If active, this flag marks the frame as eligible to be dropped in the presence of congestion. |

Table 1.1: IEEE 802.1Q header description

TS and non-TS flows is guaranteed. Figure 1.2 shows in section *a* the heterogeneous TSN scenario considered. It includes network nodes with and without TSN capabilities. The network supports both TS and non-TS packet flows, which are mixed in some of the network interfaces, TS flows are exclusively supported through TSN-capable devices. An illustrative example is presented in section *b*, which includes two TSN-capable WiFi Access point (AP), three TSN Ethernet switches, and two non-TSN-capable packet routers. The network connects two robotic arms, two servers, and several users. Two TS flows (denoted TS-1 and TS-2) are routed through a path connected to Server B, although that traffic is considered of low priority, e.g., best effort (BE)

### Control Plane

A TSN Connectivity Manager (CM) provides End-to-end (E2E) control and includes:

- Path computation element (PCE) implementing algorithms with dif-

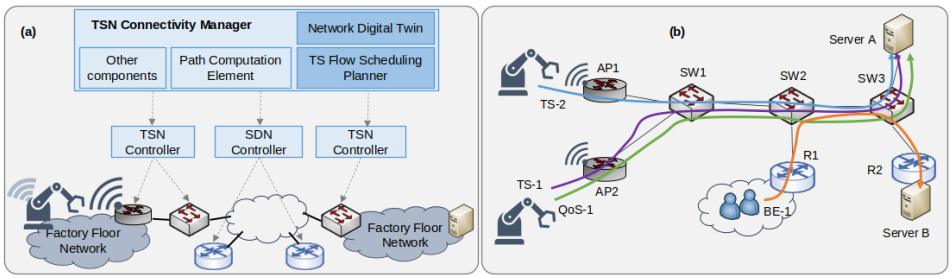



Figure 1.2: Heterogeneous network scenario for TS and non-TS packet flows from L. Velasco, G. Graziadei, Y. El Kaisi, J. Villares, O. Munoz, J. Vidal, and M. Ruiz. Provisioning of time-sensitive and non-time-sensitive flows: from control to data plane. IFIP Networking 2024, TENSOR, 2024.

ferent policies computes the path of a new request

- TAS in charge of producing scheduling for the TS flows to be deployed in the network

- Network digital tween that evaluates a set of KPIs of non-TS flows before new (TS or non-TS) flows are deployed.

Figure 1.3 shows the CM decision flowchart. When a new TS flow request arrives at the TSN CM, a provisioning process is followed that includes path computation, scheduling planning (in the case of a TS flow), and performance evaluation. In the case that the flow request is accepted, the TSN CM uses SDN controllers' north-bound interfaces to send them precise instructions for the new flow. Specifically, in the case of a TS flow, the TSN CM sends the computed network scheduling plan to the TSN controllers that will subsequently provide that plan to the packet schedulers running in the TSN-capable nodes.

This work implements the TAS module of the control plane, the mechanism of propagation of the output of the TAS module is not implemented and it is demanded as an implementation detail.

### Queuing system

To achieve the coexistence of varying classes of traffic over the network each interface has to support different queues:

- **TS-queue**: for each request mapped over the interface there is a FIFO queue which is opened and closed according to the scheduling data plane generated by TAS scheduler.

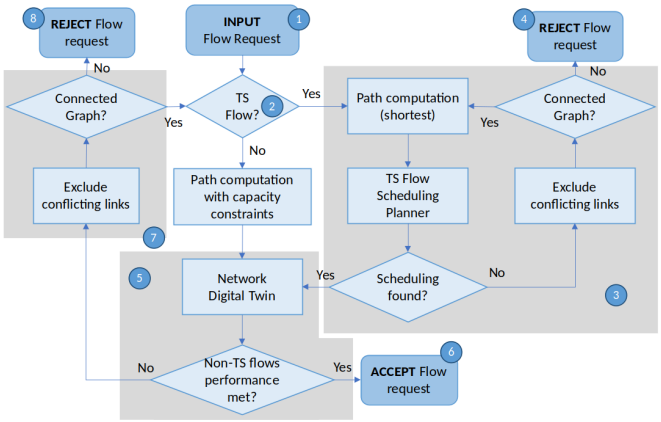



Figure 1.3: Provisioning of TS and non-TS Flows control plane from L. Velasco, G. Graziadei, Y. El Kaisi, J. Villares, O. Munoz, J. Vidal, and M. Ruiz. Provisioning of time-sensitive and non-time-sensitive flows: from control to data plane. IFIP Networking 2024, TENSOR, 2024.

- **QoS-queue**: frames are marked with different priority levels, this queue [4] is implemented according to policies and scheduling algorithm.
- **BE-queue**: FIFO queue.

Figure 1.4 illustrates an example of queuing system of a network interface. The TS-queues are aggregated in a main TS-queue according to the output of the scheduler. Given a timestamp it is possible to define the cycle id, then the corresponding time slot only with bit-shifting module operations:

$$t = 1 + ceil(\frac{time()\ \%\ T}{\tau_e}) \tag{1.1}$$

where T is the duration of SF and $\tau_e$ is the duration of a time slot. Given the data plane it is possible to obtain the TS-request which the time-slot is assigned, thus the main TS-queue opens the TS-queue of the request. A TS-request identifier is included in the TSN-header as VLAN id.

Details about the QoS queue are off-topic.

## 1.3 TAS scheduling related works

In the literature, there are a lot of approaches to solving the TSN scheduling problem summarized in the following categories:

[4]The policies are defined a-priori by the network provider and are included in the control plane. The algorithms for the queue implementation are Weighted Fair Queuing (WFQ), Priority Queuing (PQ), and Class-Based Weighted Fair Queuing (CBWFQ).

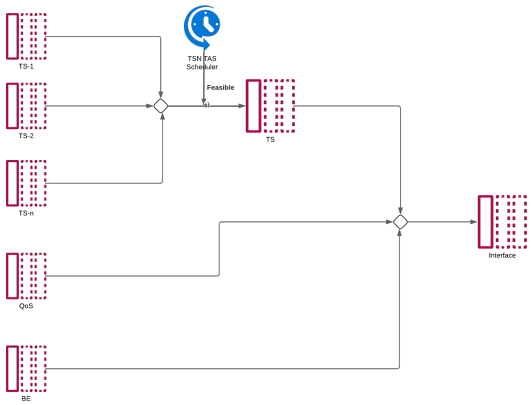



Figure 1.4: Queuing system for the aggregation of multiple traffic classes

- Approximative: find the local optimal, feasible solution
  - Heuristic as Greedy + Local Search (LS)
  - Meta-heuristic as Greedy randomized adaptive search procedure (GRASP)
  - Genetic as Biased Random Key Genetic Algorithm (BRKGA)
  - Machine learning as Deep reinforcement learning
- Exact: find the global optimal, feasible solution
  - Integer linear programming (ILP)
  - SMT
  - Dynamic programming

Table 1.2 from [3] reports papers that discuss the TAS scheduling problem from 2021 up to date.

| **Reference** | **Category** | **Approach** |
|---|---|---|
| [15], [17] | ILP | Job-shop flow |
| [19], [23] | Heuristic | Vector bin packing |
| [13] | Heuristic | Dynamic programming + Greedy |
| [8] | Z3 SMT/OMT | OMNeT++ simulation |
| [12] | Genetic algorithm | BRKGA |
| [1], [22] | Machine learning | Deep reinforcement learning |
| [16] | Machine learning | Reinforcement learning |

Table 1.2: TAS scheduling algorithm literature

This work focuses on developing an ILP model and a Heuristic one.

# Chapter 2

# TSN scheduling in heterogeneous network model

In this chapter the statement of the TAS scheduling problem, the heterogeneous network model, the input parameters and expected output definition, the simplification hypothesis and mathematical restrictions.

## 2.1 Problem statements

Given a heterogeneous time-sensitive network and a set of signal requests constrained in time in jitter and delay, generate the optimal scheduling plan. There are two possible statements of the problem:

- **Scheduling without reconfiguration**: Given a time-sensitive network with a previous scheduled TS-load, accepting a new TS request if feasible.
- **Scheduling with reconfiguration**: Given a time-sensitive network and an incumbent scheduling plan with allocated resources of different TS requests, accepting as input a new TS-request and producing, if feasible, as output the new scheduling plan reducing the number of performed changes between it and the incumbent one.

## 2.2 Optimization objectives

In this section, the optimization objectives are categorized according to [3].

The table 2.1 defines a framework to evaluate each produced model. The first number of the objective ID refers to the group of the objective as described in the following list.

| Objective id | Description |
|---|---|
| **Primary optimization objectives** | |
| 3.1 Maximize the determinism | Minimize the maximum jitter of the network. |
| 3.2 Improve the transmission metrics | Minimize the end-to-end delay in the transmission. |
| **Secondary optimization objectives** | |
| 2.1 Maximize the load balancing | Balance the loading of the TS traffic over the nodes of the network. In the model function performed by the path computation module. |
| 2.2 Minimize the length of the path | Function performed by the path computation module. A lower-size path means using a lower number of resources, thus increasing the number of feasible requests. |
| 2.3 Minimize the fragmentation | Maximize the number of contiguous available time slots over each interface of the network. |
| 3.3 Maximize security | Include additional bands of guards or slots reserved for the verification of checksum of the headers. |
| 3.4 Maximize robustness | Include additional bands of guards or slots reserved for synchronization of the distributed system's nodes. |
| **Thirdly optimization objectives** | |
| 4.1 Minimize data plane update cost | Minimize the cost in terms of transmission performances when a new data plane is installed. Minimize the peak traffic produced by scheduling plan updates. |
| 4.2 Minimize solving time | Guarantee solving time compatible with the SF duration. |
| **Non optimization objectives** | |
| 1.1 Minimize topology cost | How the model scheduling performances are dependent on topology changes. |
| 1.2 Multicasting | Support multicast traffic and the interfaces load improvement produced. |

Table 2.1: Scheduling algorithm optimization objectives

1. Objectives in the first group function more as acceptance criteria for solutions rather than as optimization goals. If a solution meets these criteria, it is considered suitable for a specific use case.

2. The second group of objectives aims to enhance overall system performance rather than solely ensuring the schedulability of time-sensitive

flows. These objectives include distributing spare capacity and balancing link loads.

3. One of the key goals in TSN is to provide bounded latency guarantees to streams, necessitating the reduction of non-determinism caused by factors like variable delay and synchronization errors. Minimizing jitter, particularly at the last hop of the delivery path, is crucial for achieving determinism.

4. Specific problem statement objectives. In scheduling with reconfiguration, the scheduling algorithm aims to reduce the number of changes between the incumbent data plane and the new one generated when a new feasible request is produced. The algorithm has to guarantee a real-time response, compatible with the SF duration.

## 2.3 Heterogeneous network model

In a generic heterogeneous network, it's necessary to consider different interfaces with varying transmission means and different layer 1 and 2 protocols. The first aim is to define an abstract model that can describe a heterogeneous network, software side it means to implement an **abstract factory pattern**.

### 2.3.1 Abstract factory pattern

This pattern provides a way to create families of related objects without imposing their concrete classes. It encapsulates a group of individual factories that have a common theme without specifying their concrete classes. A client software component creates a concrete implementation of the abstract factory and then uses the generic interface of the factory to create concrete objects that are part of the family [24].

This pattern separates the details of the implementation of a set of objects from their general usage and relies on object composition, and this is exactly what is required by this problem: a way to treat different implementations and protocols that describe the same functional processes.

Figure 2.1 shows the Unified Modeling Language (UML) diagram, the implementation of the factory pattern, and the class relationships definition. At the abstract level, the classes SchedulingPlan (in the role of incumbent and new computed), the TimeSlot and its Assignment are required. Given the cardinality of their relationships, in the implemented version they are embedded in the Network class. The diagram contains an example of how it is possible to generalize the algorithm in a heterogeneous network, and how to implement a specific class for an interface typology starting from its abstract factory. The class AP1 to SW1 covers all the interfaces that

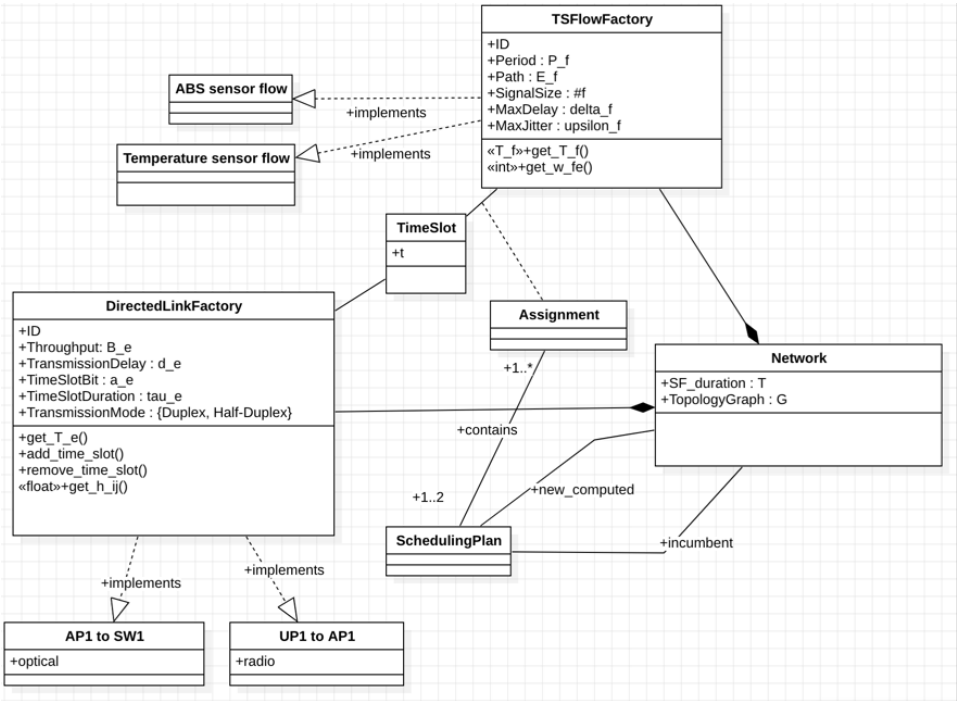



Figure 2.1: TAS scheduling algorithm UML diagram. Abstract factory pattern implementation.

connect the access point of the family AP1 to switches of family SW1, note that in this case the connection is guaranteed by an optical cable which has its characteristics.

Each of the classes in UML implements a factory for a specific actor of the TSN scheduling module. The next subsections discuss the factory classes and their methods.

### 2.3.2 NetworkInterface factory class

The following principal attributes describe a factory class for the object NetworkInterface. Note that if a duplex interface between two hops $a, b$ is represented by two different directed links $e_1 : a \to b$, $e_2 : b \to a$.

#### Throughput

The throughput is a specific metric that measures how many bits can be sent per unit of time for the given interface. In the model, the attribute that describes this information for the interface $e$ is $B_e$. In the case that an interface can offer different speeds [1], the lowest speed is considered.

[1] E.g. commonly, wireless interfaces can adapt their modulation format as a function of the quality of the signal of the receiver

This assumption guarantees the performance of the TS flows even under the worst-case scenario.

There are different definitions of throughput [25]:

- **Maximum throughput** closely related to the channel capacity of the system, is the maximum possible quantity of data that can be transmitted under ideal circumstances. The best-case assumption in terms of throughput is the worst-case assumption in terms of scheduling because a higher quantity of data per time unit means a higher TAS scheduling complexity. In the following, the model will refer to this throughput's definition.
- **Asymptotic throughput** is usually estimated by sending or simulating a very large sequence of data packets through the network and measuring the network path throughput in the destination node.
- **Peak measured throughput** is throughput measured by a real or simulated system over a short period. Given that the system is synchronous and the SF is cyclic, this definition of throughput has to reach the asymptotic throughput for the TS traffic in one cycle.

#### Network interface propagation delay

The worst-case propagation delay $d_e$ over the interface. It is influenced by factors such as the distance between sender and receiver, the medium through which the signal is transmitted (such as wire, fiber optic cable, or wireless transmission), and the speed at which the signal can propagate through that medium. The next section describes how the model manages the end-to-end delay.

#### Scheduling unit definition

According to the IEEE 802.1Qbv, the time-aware scheduler is designed to separate the communication on the network into fixed-length, repeating time cycles. For each interface $e$, is defined $T_e$ which is the set of the scheduling units. The scheduling unit definition is specific for each interface and depends on its characteristics. In the following, we define it as a time slot of the network interface. Each time slot of the scheduling plan of the interface $e$ has two fixed attributes: the duration in time $\tau_e$ and the number of bits processed $a_e$. The process of defining the scheduling unit and the relationship in time and space between different interfaces with different characteristics is the main problem of scheduling in a heterogeneous scenario. This problem is discussed in section 2.6. For optical interfaces, the propagation delay is evaluated as:

$$d_e = \frac{Distance * n}{c} \tag{2.1}$$

where $c$ is the speed of light travelling through a vacuum and $n$ is the reflective index which is usually equal to 1.5. For WiFi interfaces, it depends not only on throughput and distance but also on the WiFi technologies, interference and congestion. For a factory scenario, the correct dimension of propagation delay is of some us [2].

### Synchronization methods

The IEEE 802.1Qbv time-aware scheduler has to ensure that the interface is busy for the transmission of another TS request. The time-aware scheduler achieves this by putting a guard band at the edge of every scheduling window that carries time-critical traffic. During this guard band time, no new frames transmission may be started, only already ongoing transmissions may be finished. The duration of the guard band is not defined by IEEE802.1Qbv, it could be as long as it takes the maximum frame size to be safely transmitted[2]. $G_e$ is the set of the not available time slots of the network interface $e$, also the guard time slots are included in it.

### Signal processing mode

Different interfaces process the signal differently. The model covers the following processing modes [3].

- **Express Transmission Mode**: data is transmitted immediately as it becomes available, without waiting for the entire message to be received. This mode is characterized by low latency, as data is sent without buffering or waiting for verification of the entire message. It is commonly used in real-time applications where timely delivery of data is crucial, such as voice and video communication and other time-critical flow. In virtual circuit design is not required to wait to receive the header to read the MAC destination, the next hop is already known.

- **Store-and-Forward Transmission Mode**: the entire message is received and stored in a buffer before being forwarded to the next hop on the network. This mode allows for error checking and correction before forwarding the message, improving reliability. It is commonly used in networks where reliability is more important than low latency or scenarios characterised by a higher probability of error in transmission.

[2] In the model, it is designed to protect the transmission in the worst-case scenario, which is when a network interface $e$ sends exactly $a_e$ bits during the time slot. According to this, the band of guard is defined as several contiguous time slots.

[3] Cut-through switching and fragment-free switching are processing modes not implemented by the model. This is not a restriction, to support them is possible to *Overload* the pipeline function discussed in the section *space division problem* 2.6.2.

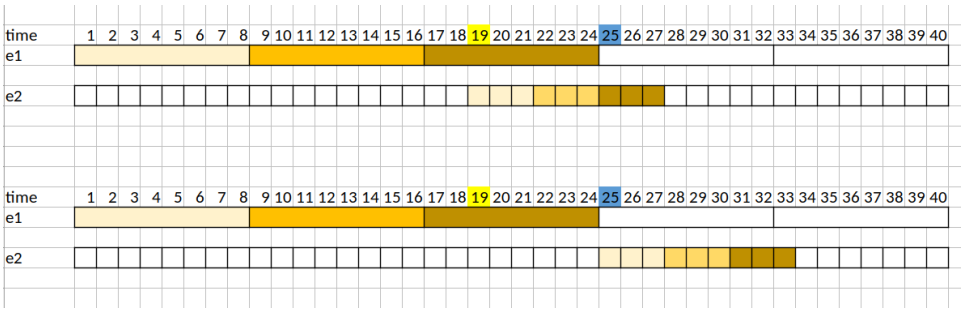



Figure 2.2: Comparison between different processing modes for the same TS-flow in the same network configuration. In the first scenario the express transmission mode, in the second one the store and forward.

- **Adaptive Transmission Mode**: the interleaving of the express and store and forward processing modes for the same directed link in different superframes is the base implementation of the model of the adaptive mode.

Figure 2.2 shows a comparison between express and store and forward transmission modes. The network configuration and TS flow parameters are the same for both examples: $a_{e_1} = 24bit$, $a_{e_2} = 8bit$ and $\tau_{e_1} = 8t.u$, $\tau_{e_2} = 1t.u.$. The propagation delay is approximated to 0.

### Transmission mode

Network interfaces can operate in various transmission modes classified as follows[4]:

- **Simplex**: the communication occurs in one direction only. One device sends data, and the other device can only receive it.
- **Half-Duplex**: communication can occur in both directions, but not simultaneously. Devices can both send and receive data, but not at the same time. Instead, they take turns transmitting and receiving.
- **Full-Duplex**: communication can occur in both directions simultaneously. Devices can send and receive data simultaneously, allowing faster and more efficient communication. This mode is common in modern Ethernet-switched networks and most wired and wireless communication standards.

[4]Following a practise approach, ethernet interfaces typically support full-duplex operation, allowing for simultaneous bidirectional communication. Older Ethernet interfaces and certain wireless standards may support half-duplex operation. Simplex mode is less common in computer networking but is used in scenarios where data only needs to flow in one direction, such as broadcasting or monitoring systems.

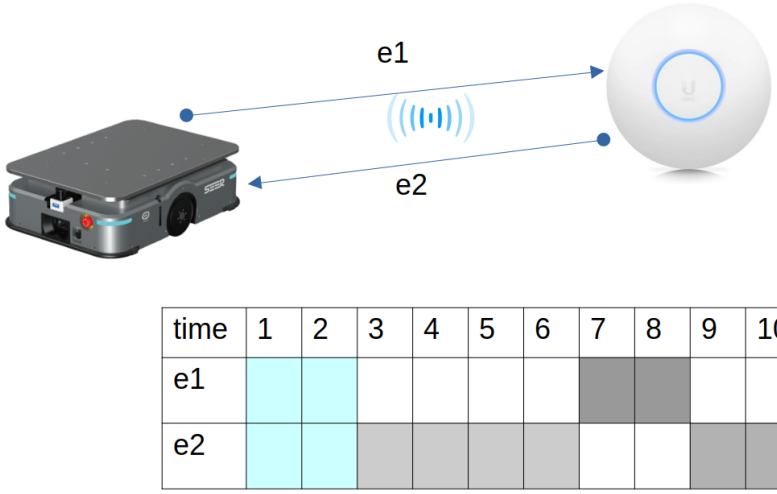



Figure 2.3: Half-duplex radio transmission mode scenario.

In the model, each interface is abstracted by a directed link, thus the interfaces work in Simplex. The network interfaces that work in Full-Duplex should be abstracted by two different directed links that work simultaneously in Simplex. The network interfaces that work in Half-duplex should be abstracted by two network interfaces $e_1, e_2$ that cannot send over simultaneously time slots. According to this, it is required to define for the Half-Duplex interfaces the $UL$ which is the set of time slots dedicated to the up-link and the $DL$ which is the set of time slots dedicated to the down-link. Each interface can only transmit in time slots in $UL$, then the time slots in $DL$ are not available and they are included in $G_e$, the set of not available time slots of $e$. Between $e_1, e_2$ the following relationship is valid:

$$UL_{e_1} = DL_{e_2} \tag{2.2}$$

Figure 2.3 shows a real case example of the half-duplex transmission in a radio scenario. The user equipment is a warehouse robot that sends with the interface $e_1$ a synchronous signal to the access point. The access point sends the command signal with the interface $e_2$. The super frame has a duration of 10 time units, the duration of the time slot is the same for $e_1$ and $e_2$ and it is equal to 1 time unit. In blue the time slots are dedicated to the transmission synchronization (a.k.a. preamble). In grey the DL time slots, thus not available for the transmission.

### 2.3.3 TSFlow factory class

This factory class is the abstract representation of the main attributes and methods of a generic TS-flow $f$.

- **Period** $P_f$ which is the periodicity in time of the TS signal. Starting from $P_f$ is possible to compute an aggregate attribute $T_f$ which is the number of periods (a.k.a iterations) that $f$ has over the $SF$ of period $T$ [5].
$$T_f = \frac{T}{P_f} \tag{2.3}$$

- **Size** $\#f$ which is the number of bits that are transmitted at each period of $f$.

- **Path** $E_f$ which is the sorted list with the $id$ of each directed link $e$ included in the path of the time sensitive $f$ [6].

- The **maximum delay** $\delta_f$ that the flow $f$ can support.

- The **maximum jitter** $\upsilon_f$ that the flow $f$ can support.

### 2.3.4 Network class

We extract the following principal attributes that can describe a factory class for the object Network.

- Duration of the scheduling of SF **T**. This number usually depends on the specific time-sensitive traffic that the network supports and it is $max_{f \in F} P_f$. To guarantee better compatibility with different scenarios, this is considered an input of the algorithm [7].

- **The topology of the network** is represented with a graph $G$ in which each node is a hop and each directed link is an interface of class NetworkInterface. Note that the scheduling problem only required to abstract the interfaces, not each hop. According to the model, the information about each hop is not taken.

- The set $F$ of the objects of the TSFlow class.

- The set $E$ of the objects of the NetworkInterface class.

[5] As better discussed in the paragraph 2.5 mathematical restrictions between the period $P_f$ and the duration of SF $T$ are required. These restrictions guarantee that the division $\frac{T}{P_f}$ always produces an integer result.

[6] As example a path defined as $E_f = [1, 4]$ where $e_1 : a \to b$ and $e_4 : b \to z$ is equal to: - interface 1 from the hop $a$ to the hop $b$ - interface 4 from the hop $b$ to the hop $z$

[7] The characteristics of the classes of time-sensitive flows are well-known a-priori and to schedule all of them it is possible to establish T. It has to be higher than the maximum period and the maximum delay admitted in the TS network.

- The current scheduling plan of the network $NSF = \{SF_e \forall e \in E\}$. Every $SF_e$ defines the allocations of slots to flows. $SF_e = \{s_{ft} \forall f \in F, \forall t \in T_e\}$, where $s_{ft}$ is equal to 1 iff the time slot $t$ over $e$ is assigned to the flow $f$, 0 otherwise.

## 2.4 End-to-end delay and jitter definitions

The model is deterministic and it cannot cover the randomness of factors of a real environment but according to the following simplification hypothesis is possible to treat the main random variables and metrics' contributions that afflict the network scenario.

### 2.4.1 End-to-end delay

The end-to-end delay is the amount of time it takes for data to travel from its source to its destination. It is the additive result of different contributions:

- **Delay of the signal** $d_{signal}$, which is a contribution that depends on the error of the signal period. This is a stochastic dimension.
- **Propagation Delay** $d_{propagation}$, which is the time it takes for the signal to travel through the transmission medium and is approximated to a worst-case deterministic value. [8]
- **Processing Delay** $d_{processing}$. Processing devices, such as routers, switches, or other network equipment, introduce delays as they process the signal. The approximation for the store and forward processing mode is described in the section 2.4.3.
- **Clock Synchronization** differences in clock frequencies or synchronization errors can introduce additional delays. We assume that this problem is out of the scope of this work because, as input, we have a synchronous heterogeneous network.
- **Queuing delay** to guarantee different QoS classes. The queuing delay depends on factors such as network congestion, packet prioritization, and the scheduling algorithm used by the network device. In our model, traffic is allocated in specific time slots with the highest priority for data traffic [9], thus we assume there is no queuing delay.
- **Scheduling delay** $d_{scheduling}$, which is specific to the scheduling problem and depends on scheduling constraints. It is provided in terms of time slots and thus is a deterministic dimension.

[8] In the model, this contribution, specific for an interface $e$ is evaluated as $d_e$ and it is treated as an input parameter.

[9] The highest priority is given to the control plane.

The following equation summarizes the definition of the end-to-end delay:

$$d = d_{signal} + d_{propagation} + d_{processing} + d_{scheduling} \tag{2.4}$$

To estimate the $d_{signal}$, given its nature, means to define a random distribution of probability. According to this, it cannot be optimized, thus it is out of the scope of the optimization model and could be approximated to 0. This is defined as a simplification hypothesis:

- **Simplification hypothesis 1 (SH_1)** $d_{signal} = 0$. The first bit of each flow is transmitted synchronously to the start of the allocated time slot over the first interface of the path.

In the worst case $d_{propagation}$ for a flow $f$ is defined as the sum of the propagation delay at each hop of the path as:

$$d_{propagation} = \sum_{e \in E_f} d_e \tag{2.5}$$

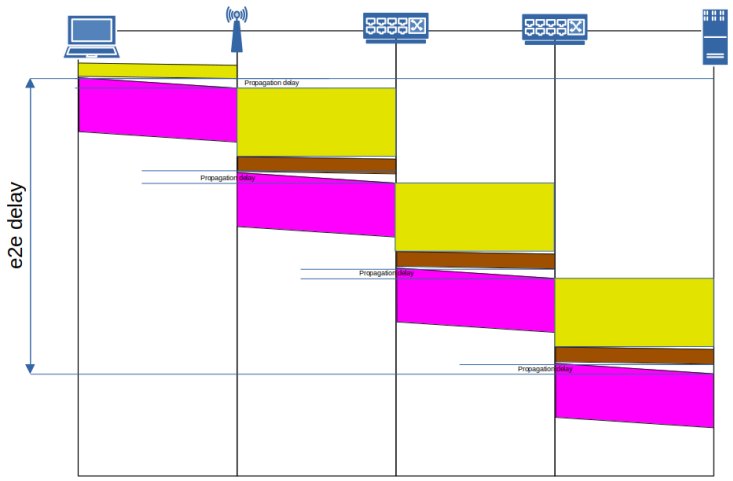
Figure 2.4 shows time graphs with the estimation of the E2E delay in generic scenarios. The time spent by the first bit in the buffer for processing purposes is highlighted in yellow, while the time spent on queuing or scheduling purposes is shown in brown. The purple squares represent the time required to complete the transmission hop by hop. In (a) the scenario involves a store-and-forward processing mode, where the entire frame is processed and checked before being delivered to the next hop. In (b) express processing mode.

### 2.4.2 End-to-end jitter

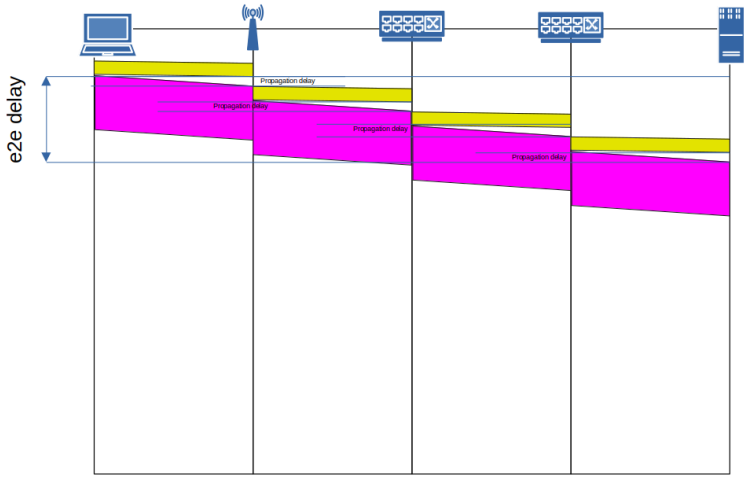
The jitter is the variation in the delay of a transmission theoretically periodical. It is the deviation from the average delay experienced by packets as they travel through a network. Jitter is typically caused by varying network congestion, routing changes, or fluctuations in transmission latency. In real-time time-sensitive communication applications, high jitter can lead to packet loss, degraded signal quality, and overall instability of the communication channel. By the nature of the delay, the jitter is defined as additive components in which there are also stochastic components. Assuming that the delay is a stochastic variable the definition of the jitter is defined by the following equation:

$$j = E\left[(d - \mu_d)^2\right] = E\left[d^2\right] - \mu_d^2 \tag{2.6}$$

Assuming that the mean signal period error is 0 the term $\mu_d^2$ is well evaluated, however, to solve $E\left[d^2\right]$ is required to treat also the propagation and processing delay contribution as a stochastic variable and evaluate the eventual covariance between the different delay contributions.




(a) Store and forward processing mode




(b) Express processing mode

Figure 2.4: Estimation of E2E delay with different processing modes.

To guarantee the determinism of the model the following simplification hypothesis is valid:

- **Simplification hypothesis 2 (SH_2)**. The jitter of each flow depends on the deterministic scheduling component of the delay.

### 2.4.3 Processing time

Each interface could perform the following operations in the processing phase:

- **Reception of Data Packet**, the device captures the entire packet

from the network interface.

- **Buffering**: the device temporarily stores the entire packet in a buffer memory. In store and forward processing mode, buffer memory can hold the complete packet until it's ready to be forwarded.
- **Error Checking**: once the packet is stored in the buffer, the network device performs error checking on the packet. This may involve verifying the integrity of the packet by checking for errors in the packet's header, payload, or any other relevant fields.
- **Transmission**: If the packet passes error checking and the device determines the correct outgoing interface or port, it forwards the packet to the next hop.

Each operation requires time, but it depends on the performance of the interface. The buffering time depends on the processing mode, the transmission of the frame is the time allocated to a scheduling window. Other components of processing time depends on interface and are implementation detail, the following simplification hypothesis is valid:

- **Simplification hypothesis 3 (SH_3)** The processing time for error checking, reception of frame and other actions not listed elsewhere is approximated to 0. If the approximation cannot be valid the time could be evaluated for optimization purposes as an additive component of the propagation delay.

## 2.5 Mathematical restrictions

The scheduling problem is not always feasible. To reach a solution it is required that between the input parameters are valid same mathematical restrictions according to the space and time division problem[10].

- The period of each flow $P_f$ is a divisor of the $T$ duration of the super frame $SF$.

$$MCD\left(P_f, T\right) = P_f \ \forall f \in F \tag{2.7}$$

  This constraint is fundamental because if we try to negate this, then $P_f$ is not a divisor of $T$ and after $n$ superframe the network cannot serve the flow and the request is blocked.

  Figure 2.5 shows an example of the time division problem in the case in which 2.6 is violated. In the example, $T$ is equal to 8 t.u. and $P_f$ to

[10]Usually in each network domain the throughput is given in power of 2, the size of a packet in WiFi transmission is given in power of 2, and the duration of a period of TSN flow is given in a power of 2. The restrictions are specified to complete the model but in the general scenario, all the restrictions are matched.

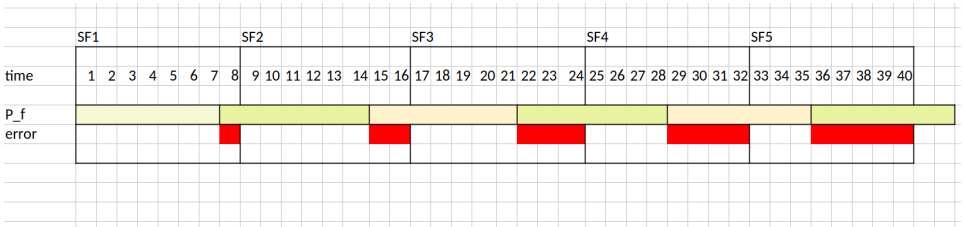



Figure 2.5: Mathematical restriction between flow and super frame periods violation example

7 t.u., thus $P_f$ is co-prime with $T$. With different colours is possible to understand the duration of each iteration of the flow. In red is the error generated by the scheduling at each SF. The error increases at each cycle, thus after 8 SF one iteration of the flow $f$ is lost. The additional negative effect is that the flow is asynchronous with the SF, which means that when the signal is generated it cannot be sent but it is stored in the buffer of the network interfaces. To summarise the restriction is required to not lose transmissions and overload the buffers of the network. The scenario in the figure is not possible, the request is a-priori rejected.

- The number of bits transmitted in one time slot by different directed links $e_1, e_2$ are not co-prime.

$$MCD\left(a_{e_1}, a_{e_2}\right) > 1 \ \forall (e_1, e_2) \in E \tag{2.8}$$

- The duration of each time slot for each directed link $e$ is a divisor of $T$.

$$MCD\left(\tau_e, T\right) = \tau_e \ \forall e \in E \tag{2.9}$$

## 2.6 Heterogeneous transformations

This section treats the problem of heterogeneous transformations. How to synchronize different interfaces with different throughputs without imposing performance limitations. The problem consists of different definitions of the concept of time unit $t$ over each interface $e$ according to its duration and the different number of bits transmitted during it.

In the DirectedLinkFactory class, two attributes are defined such that the throughput definition 2.10 is valid:

- $a_e$ which is the number of bits processed per time slot over $e$
- $\tau_e$ which is the duration of each time slot over the directed link $e$

$$B_e = \frac{a_e}{\tau_e} \ \forall e \in E \tag{2.10}$$

Only two of the three parameters are required in input, the third one is evaluated according to the previous relationship.

According to a top-down analysis, the problem can be divided into two sub-problems: time division problem and space division problem.

### 2.6.1 Time division problem

Each directed link $e$ divides the time of the superframe $T$ into time slots of duration $\tau_e$. The number of time slots is equal to $\frac{T}{\tau_e}$ which according to mathematical restrictions produces an integer result.

The method $get\_T\_e()$ of the NetworkInterface class returns a bit array. Each cell of index $(t-1)$ is equal to 1 if the time slot $t$ is available over the NetworkInterface $e$, 0 otherwise.

It is possible to pre-compute the time slots not available in $SF_e$ [11]. $G_e$ contains all the id of the time slots that for different purposes are not available. The following pseudo-code shows the pre-computation of $T_e$

**Require:** $e \in E$
1: $t_e \leftarrow \frac{T}{\tau_e}$
2: **for** $t \in [1, t_e]$ **do**
3: $\quad T_e[t-1] \leftarrow \neg G_e[t-1] \ \& \ T_e[t-1]$
4: **end for**

Given a heterogeneous scenario, the duration of the time slot is not fixed among different network interfaces. This dimension depends on the throughput of the interface and could be necessary to pass between two different representations of the time between the interface $e_1$ and $e_2$. According to the mathematical restrictions, this transformation is always feasible. It is possible to pre-compute a time transformation matrix $L = \{l_{i,j} \in Q | \forall (i,j) \in E \times E\}$.

**Require:** $E$
1: **for** $e_1 \in E$ **do**
2: $\quad$ **for** $e_2 \in E$ **do**
3: $\quad\quad l_{e_1 e_2} \leftarrow \frac{\tau_{e_1}}{\tau_{e_2}}$
4: $\quad$ **end for**
5: **end for**

[11] As discussed the preamble and guard time slots are not available by the IEEE 802.1Qbv guideline, the Half-duplex cannot assign the downlink (DL) time slots.

A built-in function can solve the time transformation problem. Given a time slot $t$ over $e_1$, it is possible to obtain the complimentary in time $t^*$ over $e_2$.

**Algorithm 1** Time transformation function : timeTransform()

**Require:** $e_1, e_2 \in E, t \in T_{e_1}$

1: $t \leftarrow t * l_{e_1,e_2}$

**return** $< t >$

### 2.6.2 Space division problem

Each time slot $t$ of the interface $e$ transmits $a_e$ bits. Given a flow $f$ with $\#f$ bits transmitted per period, it is required to implement the concept of scheduling window, also defined scheduling slice of TAS.

A scheduling window is a list of contiguous time slots over an interface $e$ that have to be allocated to $f$ to guarantee the transmission of all the $\#f$ bits. Each time slot is an atomic unit in the number of bits transmitted, according to this, it is not possible only to allocate a part of it. A matrix $W$ is defined:

$$W = \{w_{fe} \in N | f \in F, e \in E\} \tag{2.11}$$

where $w_{fe}$ is the scheduling window size for $f$ over each directed link of the path $e \in E_F$ and it is equal to $ceil(\frac{\#f}{a_e})$ [12] if $e \in E_f$, 0 otherwise.

**Algorithm 2** Pre-omputation of the scheduling windows

**Require:** $F, E$

1: **for** $f \in F$ **do**
2:  **for** $e \in E$ **do**
3:  **if** $e \in E_f$ **then**
4:  $w_{fe} \leftarrow ceil(\frac{\#f}{a_e})$
5:  **else**
6:  $w_{fe} \leftarrow 0$
7:  **end if**
8:  **end for**
9: **end for**

Given two different interfaces $e_1$ and $e_2$ that have different throughput, thus different time slot duration $\tau_{e_1}$ and $\tau_{e_2}$ and different number of bits transmitted per time slot $a_{e_1}$ and $a_{e_2}$, and given a time slot $t$ such that is the starting time slot over $e_1$ for the flow $f$, function *firstTimeSlot()* 3 returns the first time slot available over $e_2$ in which the transmission can

[12] As discussed the time slot is an atomic unit. The ceil function guarantees that If at least one bit of $t$ is required by $f$ then the entire time slot $t$ has to be allocated to $f$.

start after the end of the transmission in $e_1$. This scenario covers exactly what happens in store and forwarding transmission mode.

**Algorithm 3** Transformation function : firstTimeSlot()

**Require:** $(e_1, e_2) \in E, t \in T_{e_1}, w_{fe_1}$
1: $t \leftarrow t + w_{fe_1}$
2: $t \leftarrow \text{timeTransform}(e_1, e_2, t)$
**return** $< t >$

### 2.6.3 Pipeline optimization

Given a heterogeneous scenario, the number of bits transmitted per time slot is not fixed, it depends on the throughput of the interface and could be necessary to pass between two different representations of the space between the interface $e_1$ and $e_2$. According to the mathematical restrictions, this transformation is always feasible. It is possible to pre-compute a space transformation matrix $H = \{h_{i,j} \in Q | \, \forall (i,j) \in E \times E\}$.

**Require:** $E$
1: **for** $e_1 \in E$ **do**
2: **for** $e_2 \in E$ **do**
3: $h_{e_1 e_2} \leftarrow \frac{a_{e_2}}{a_{e_1}}$
4: **end for**
5: **end for**

The H matrix solves the space transformation problem, for example, given two interfaces $e_1$ and $e_2$ with $B_{e_1} = 1GBit/s, B_{e_2} = 10GBit/s$ the value of $h_{e_1 e_2} = 10$, when $\tau_{e_1} = \tau_{e_2}$. It means that 10 time slots of $e_1$ transmit the same quantity of bit of one slot of $e_2$, according to this given a path that has $e_1, e_2$ in series, the transmission of over $e_2$ can start in parallel with the 10th time slot of $e_1$. According to this example, the solution of starting after the end of the transmission in $e_1$ implemented by the function *firstTimeSlot()* is not optimal for the **express** transmission mode.

Inspired by the *pipeline concept* [13] it is possible to convert each transmission into a pipeline in which each interface $e$ is a *processing element* in which each of the $w_{fe}$ time slots is a *stage*. The dependencies in our pipeline are only of input-output type and the topology graph of the network describes

[13] A pipeline consists of a chain of processing elements, arranged so that the output of each element is the input of the next; each processing element is arranged into atomic sub-elements. In instructions pipelining *(base, without reordering)* each interaction performs the following stages: fetch, decode, execute and store. The problem is similar but in our case, each interface performs only a fetch stage per each time slot involved in the scheduling window.

them with the direction of the directed link arrow. In this game, the only constraint is to overlap stages of different interfaces of the same path according to the coefficient computed in the space transformation matrix $H$. Table 2.2 defines the pipeline problem.

| | |
|---|---|
| **Objective** | Minimizing the distance in time between the start of scheduling windows of two different contiguous interfaces declared in $E_f$. |
| Constraint 1 | The input-output dependency between two interfaces claims that it is not possible to send a bit without the bit being received by the interface. |
| Constraint 2 | The preemption is not allowed. Each scheduling window is an atomic unit of $w_{fe}$ time slots. |
| Constraint 3 | The lower bound between the starting of two scheduling windows in contiguous interfaces $e_1, e_2$ is the propagation delay of the $e_1$. |
| Constraint 4 | The solution has to cover two different transmission modes: express and store and forwarding. In the store and forwarding mode all the bits in one frame have to be received by the receiver to a check, instead in express mode the transmission doesn't require any check is a stream of bits. |

Table 2.2: Pipeline problem definition

Different approaches are valid to solve the pipelining problem, we focused on the iterative solution given the pre-computed matrix $H$.

Given two different interfaces $e_1$ and $e_2$ that have different throughput, thus different time slot duration $\tau_{e_1}$ and $\tau_{e_2}$ and different number of bits transmitted per time slot $a_{e_1}$ and $a_{e_2}$, and given a time slot $t$ such that is the starting time slot over $e_1$ for the flow $f$, the function returns the minimum required time between the two starting time slots in two consecutive interfaces of $E_f$. Algorithm 4 solves the pipelining problem.

The example in figure 2.6 covers three scenarios of the pipelining problem from interface 1 to interface 2. In these examples, the propagation delay of transmission is approximated to 0.

- Scenario 1: $a_{e_1} = 24bit$, $a_{e_2} = 8bit$ and $\tau_{e_1} = 8t.u$, $\tau_{e_2} = 1t.u.$, thus $h_{e_1e_2} = \frac{1}{3}$. The windows size are $w_{fe_1} = 3\, w_{fe_2} = 9$. According to the pseudo-code the pipeline() result for the express mode is $first_{e_2} - \tau_{e_2} * (w_{fe_2} - ceil(h_{e_2e_1})) = 18t.u.$, for the store and forward mode $24t.u.$.

- Scenario 2: $a_{e_1} = 8bit$, $a_{e_2} = 8bit$ and $\tau_{e_1} = 1t.u$, $\tau_{e_2} = 1t.u.$, thus $h_{e_1e_2} = 1$. The windows size are $w_{fe_1} = 18\, w_{fe_2} = 18$. According to

**Algorithm 4** Overlapping optimization function : pipeline()

**Require:** $e_1, e_2 \in E, \tau_{e_1}, d_{e_1}, w_{fe_1}, \tau_{e_1}, w_{fe_2}, h_{e_1e_2}, h_{e_2e_1}, l_{e_1e_2}$

$last_{e_1} \leftarrow w_{fe_1} * \tau_{e_1}$
$first_{e_2} \leftarrow timeTransform(e_1, e_2, last_{e_1})$
**if** $e_1.mode == store\&forward$ **then**
    $sigma \leftarrow first_{e_2}$
**else if** $e_1.mode == express$ **then**
    **if** $\tau_{e_2} \leq \tau_{e_1} * h_{e_1e_2}$ **then**
        $sigma \leftarrow first_{e_2} - \tau_{e_2} * (w_{fe_2} - ceil(h_{e_2e_1}))$
    **else**
        $sigma \leftarrow l_{e_1e_2} * \tau_{e_2}$
    **end if**
**end if**
$sigma \leftarrow sigma + d_{e_1}$ // adding propagation delay
**return** $< sigma >$

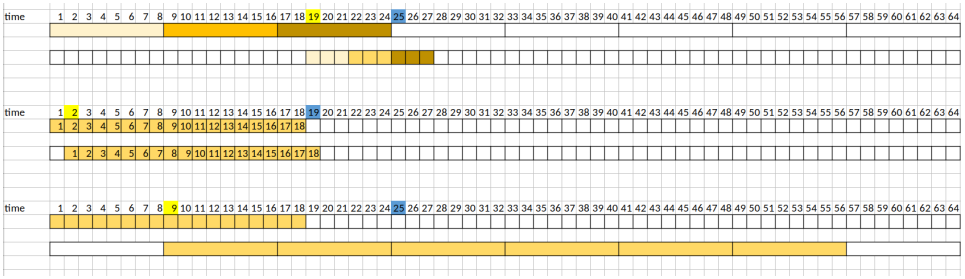


Figure 2.6: Three different scenarios of the pipelining problem from interface 1 to interface 2. The time unit is defined in the first row. The cell in yellow represents the first available time slot for the express mode, and the blu one is the first available in store and forward mode.

the pseudo-code the pipeline() result for the express mode is $1t.u.$, and for the store and forward mode $18t.u.$.

- Scenario 3: $a_{e_1} = 8bit$, $a_{e_2} = 24bit$ and $\tau_{e_1} = 1t.u$, $\tau_{e_2} = 8t.u.$, thus $h_{e_1e_2} = 3$. The windows size are $w_{fe_1} = 18\, w_{fe_2} = 6$. According to the pseudo-code the pipeline() result for the express mode is $8t.u.$, and for the store and forward mode $24t.u.$ as in the first scenario.

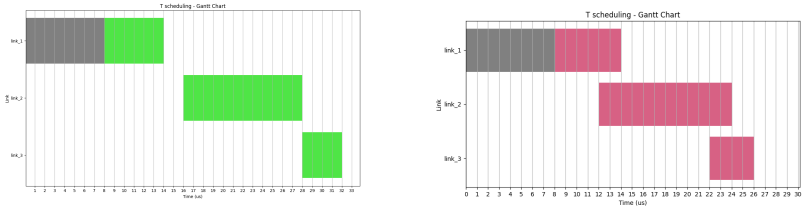
The complexity of this problem, according to this formulation, is constant in time and $O(|E|^2)$ in memory.

The pipeline optimization improves two aspects:

- **Delay and jitter performance**: overlapping in time the scheduling for different interfaces means reducing the amount of time required to transmit in the network the frame.

- **Colors optimization**: according to heuristic criteria involves partitioning $T$ into regions of varying lengths. Each request's period can only be addressed within a single region. By minimizing the duration of these regions, we can maximize the number of available regions, thus enhancing the algorithm's capacity to accommodate more requests. Reducing the latency of individual periods translates to augmenting the probability of accommodating a new TS-request.

Figure 2.7 compares the result of the scheduling plane produced for the same scenario but with the two different implementations discussed. The output is produced with the ILP model which is discussed in the next chapter. Gantt chart (a) is produced with store and forward mode and (b) is a schedule produced with the pipeline optimization in express transmission mode. The scenario is the same: a flow $f$ with a path $E_f : [1, 2, 3]$ at the first period in the $SF$, the three links have respectively a time slot duration of 2us, 4us and 2us, the physical propagation delays for the interfaces are 0.5us, 2us, 0.5us. The grey time slots are reserved as preamble or guard. Over **SH_1** the first bit of the signal is produced to the instant 0, according to this the delay in the first scenario is 28.5 us and in the second scenario is 22.5 us with an expected improvement in performance e2e estimated delay of 21.05%. According to the color heuristic criteria the region to schedule the first period of the flow $f$ has a length of 32us in the first scenario and 26us in the second one with an expected improvement of 18.75%. Note also that in the first scenario, the time between 14us and 16us is lost because $\tau_{e_2} = 4$, thus in $e_2$ it is possible to allocate only in multiple of 4us.




(a) Store and forward (b) Express

Figure 2.7: Comparison of scheduling for varying processing mode.

## 2.7 Parameters

Table 2.3 summarizes the input parameters of the problem. The resolution of the time unit depends on the highest throughput, for example in the case of the presence of 10Gbits interfaces the resolution is 0.8 ns. The resolution in bits processed per time slot has to be homogeneous per all the interfaces $(B_e, a_e)$ and requests $(\#f)$, it could be expressed in bytes or bits.

Table 2.4 summarizes the pre-processed parameters or aggregated class's attributes as described in the previous sections. In this table a new request $r$ is treated as a normal time-sensitive flow because, at the pre-processing layer, the new request is already part of the scheduling problem and the algorithm has to evaluate its feasibility, in other words, $r$ as each already scheduled flow $f$ has to be scheduled.

| **Parameter** | **Description** |
|---|---|
| *SF definition* | |
| $T$ | Duration of the scheduling superframe of the time-sensitive network. |
| $E$ | Set of directed links in the topology; index $e$. |
| $F$ | Set of time-sensitive flows. Index $f$. |
| $SF_e$ | $SF$ for the network interface $e$. $SF_e = \{s_{et}\}$, each component is equal to 1 if time slot $t$ is available for $e$, i.e., both it is not already allocated to an existing TS flow and it can be used for transmission. |
| $NSF$ | Network $SF$ defined as a set of $SF_e \forall e \in E$ |
| *Network interfaces definition, index e* | |
| $G_e$ | Set of fixed not available time slots for the network interface $e$. It can be extended, in the model it covers the guard time slots and the DL time slots of Half Duplex transmissions. |
| $T_e$ | The set of time slot $t$ of the interface $e$ |
| $\tau_e$ | Time duration of each $t \in T_e$ |
| $a_e$ | Bit processed by each $t \in T_e$ |
| $B_e$ | Throughput of the network interface $e$ |
| $d_e$ | Propagation delay of the network interface $e$ |
| $mode_e$ | Frame processing mode for the network interface $e$ |
| *Time sensitive flows definition, index f* | |
| $E_f$ | Sorted list of network interfaces in the path of the flow $f$ |
| $P_f$ | Period of the flow $f$ |
| $\delta_f$ | Maximum allowed delay for $f$ |
| $\upsilon_f$ | Maximum allowed jitter for $f$ |
| *New flow request definition r* | |
| $r = \{E_r, P_r, \delta_r, \upsilon_r\}$ | New scheduling request for the time-sensitive flow $r$. |

Table 2.3: Problem input parameters

| Parameter | Description |
|---|---|
| $T_f$ | Number of iterations (a.k.a. periods) in the $SF$ for the scheduled flow $f$ |
| $t_e$ | Number of time slots in one SF for the network interface $e$ |
| $L$ | Matrix of network interface time division. $L = \{l_{e_1e_2}\}$ where $l_{e_1e_2}$ is the ratio between the duration of the time slot between the network interfaces $e_1, e_2$. |
| $H$ | Matrix of network interface space division. $H = \{h_{e_1e_2}\}$ where $h_{e_1e_2}$ is the different speed coefficient between the interfaces $e_1, e_2$. |
| $W$ | Matrix of scheduling window. $W = \{w_{fe}\}$ where $w_{fe}$ is size of the flow $f$ over the network interface $e \in E_f$. |

Table 2.4: Problem pre-processed parameters

## 2.8 Problem statements comparison

The scheduling algorithm has the following expected output:

- The scheduling window size of the flow $f$ over the directed link $e$ $w_{fe} \forall f \in F, e \in E$
- The new scheduling plan $SF_e \forall e \in E$ with resources allocation for TS traffic, if feasible. In the scheduling plan, the starting time slot per each flow of the scheduling window.
- For the problem of scheduling with reconfiguration: the list of changes performed between the new scheduling plan and the incumbent one.
- Estimated end-to-end delay and jitter, according to simplification hypothesis, for the transmission in the network of each TS flow.

Table 2.5 summarizes the statement of the problem in the case of scheduling without the reconfiguration of the data plane for the already assigned request. Table 2.6 summarizes the statement of the problem in the case of scheduling with reconfiguration.

| *Input problem restrictions* | |
|---|---|
| $F = \{r\}$ | The time-sensitive flows set contains only the new scheduling request. |
| $SF_e = \{s_{et} = 0\}\ \forall e \in E$<br><br>$G_e \leftarrow G_e \cup \{t \mid s_{et}^{-1} = 1\}\ \forall e \in E$ | For each network interface $e$ the scheduling set is empty. The incumbent scheduling plan assignations $s_{et}^{-1}$ are added in the set $G_e$ of the not available time slot. |
| *Output* | |
| $SF_e \forall e \in E_r$ | Data plan (a.k.a scheduling plan) of the new request restricted to the time-sensitive flow $r$. |
| $w_{re} \forall e \in E_r$ | Scheduling window size for each network interface in the path of the new request. |
| $d_i\ 1 \leq i \leq T_r$ | Estimated end-to-end delay for each iteration (a.k.a. period) of the new request $r$. |
| $j$ | Estimated end-to-end jitter of the new request $r$. |

Table 2.5: Scheduling without reconfiguration statement

| *Input problem restrictions* | |
|---|---|
| $F = \{f_1, f_2, ..., f, ..., r\}$ | All the flows scheduled and the new request. |
| *Output* | |
| $SF_e \forall e \in E$ | Data plan (a.k.a scheduling plan) of the network. |
| $w_{fe} \forall f \in F \forall e \in E$ | Scheduling window size for each time-sensitive flow $f$, for each network interface of its path. |
| $d_{fi}\ \forall f \in F,\ 1 \leq i \leq T_r$ | Estimated end-to-end delay for each iteration (a.k.a. period) of each request. |
| $j_f \forall f \in F$ | Estimated end-to-end jitter of each time-sensitive flow scheduled. |
| $\Delta_{SF_e} \forall e \in E$ | Scheduling changes performed to the data plane of each interface of the network. |

Table 2.6: Scheduling with reconfiguration statement

# Chapter 3

# ILP Model

The optimization problem could be solved with ILP. It is required to define a mathematical model with three different elements

- **Decision variables (X)** represent the matrix with the unknown information in a constraint programming problem. In the ILP model, the decision variables are integers: model structure hypothesis 1 (**MSH_1**).
- **Constraints set (A)** are rules defined in the statement of the problem. In the ILP model, the constraints are represented with linear equations: model structure hypothesis 2 (**MSH_2**).
- **Objective function F** evaluates the optimality of the solution. In the ILP model, the objective function is linear: model structure hypothesis 3 (**MSH_3**).

According to these definitions and given the input parameters set (B) as described in the section 2.7 the solution of the model is expressed by

$$AX = B \tag{3.1}$$

where the solution X, if exists, assigns to the decision variable the values that maximize or minimize the evaluation for the objective function F(X). The algorithm is equal in both cases, an F(X) maximization problem could be converted as the minimization problem of the function F(-X). The complexity of the algorithm is *np-complete*.

TSN TAS scheduling problem belongs to the class of Job Shop Scheduling Problem (JSSP) [26]. A classical JSSP can be described as follows: in a job shop environment containing several machines $M = \{M_1, M_2, ..., M_m\}$, there are several jobs $J = \{J_1, J_2, .., J_i, .., J_n\}$, each job, say $J_i$, contains a serial of operations $O_i = \{O_{i1}, O_{i2}, .., O_{ij}, .., O_{in}\}$ which need to be processed in a predefined technological sequence. Each operation is assigned a machine in $M$ to be processed with a given processing time $p_{ij}$. Sequencing needs to be

done for operations in all machines to minimize the maximum completion time of all jobs, i.e., to minimize the makespan.

In TAS scheduling, each network interface is a JSSP machine, and each request submitted to the scheduler module is a JSSP job. Each job is divided into tasks that must be processed in the order defined in the request path $E_f$. The duration of each task over a specific machine is $w_{fe}$. As discussed in the literature there are a lot of articles as [4] that solve the IEEE 802.1Qbv time-aware sharpers problem. The contribution of this work in JSSP field is the allocation of tasks in a real-time heterogeneous scenario in which the duration of the scheduling unit is different between different machines. The solvers used for the model are **Cplex** [9] and **Gurobi** [7], the Python framework to build and optimize the *.lp* file (input solver file) is **pulp** [18]. The source code is available in the Appendix.

## 3.1 Scheduling with reconfiguration

This section implements an ILP model to solve the problem in the statement of the reconfiguration of the network data plane.

### 3.1.1 Decision Variables

In the following section are described the decision variables introduced in the model.

- The starting time slot binary matrix

$$X = \{x_{feti}\} \tag{3.2}$$

  where $1 \leq f \leq |F|, 1 \leq e \leq |E|, 1 \leq t \leq |T_e|, 1 \leq i \leq T_f$. $x_{feti} = 1$ iff the start of the window allocation for the flow $f$ at iteration $i$ in the network interface $e$ starts at the timeslot $t$, 0 otherwise.

- The slot allocation binary matrix

$$Y = \{y_{feti}\} \tag{3.3}$$

  where $1 \leq f \leq |F|, 1 \leq e \leq |E|, 1 \leq t \leq |T_e|, 1 \leq i \leq T_f$. $y_{feti} = 1$ iff the the timeslot $t$ over the network interface $e$ is allocated to the flow $f$ at iteration $i$, 0 otherwise.

- The changes binary matrix

$$C = \{c_{fet}\} \tag{3.4}$$

  where $1 \leq f \leq |F|, 1 \leq e \leq |E|, 1 \leq t \leq |T_e|$. $c_{fet} = 1$ if there is a change between the incumbent and the new data plane in the time slot $t$ over the directed link $e$, 0 otherwise.

- The estimated delay integer matrix

$$D = \{d_{fi}\} \tag{3.5}$$

where $1 \leq f \leq |F|, 1 \leq i \leq T_f$. $d_{fi}$ contains information about the E2E delay for the transmission of the flow $f$ at the iteration $i$ defined according to **SH_1**. The error of the approximation depends on the duration of time slot $\tau_{dest}$ over the last directed link of the path of $E_f[-1]$.

- The estimated jitter integer vector

$$J = \{j_f\} \tag{3.6}$$

where $1 \leq f \leq |F|$. $j_f$ is the jitter for the flow $f$ in the $SF$. It is evaluated on the subset of $D$ where the flow is fixed to $f$ according to **SH_2**.

- A continuous variable W that indicates the maximum estimated delay of the system. It is introduced to balance and minimize the delay.

- A continuous variable Z that indicates maximum estimated jitter. It is introduced to balance and minimize the jitter.

### 3.1.2 Constraints

1. Each time slot $t$ of the network interface $e$ $(e, t)$, could be assigned only once.

$$\sum_{f \in F} \sum_{i \in T_f} y_{feti} \leq 1 \quad \forall e \in E, \forall t \in T_e \tag{3.7}$$

2. Each iteration $i$ of a flow $f$ has to be scheduled once in all the directed links of the flow of $f$ $E_f$.

$$\sum_{t \in T_e} x_{feti} = 1 \quad \forall f \in F, \forall e \in E_f, \forall i \in [1; T_f] \tag{3.8}$$

3. If the scheduling window for iteration $i$ of a flow $f$ starts over $(e, t)$, then $w_{fe}$ contiguous time slot are allocated to $f$ starting from $t$.

$$x_{feti} \leq y_{feki} \quad \forall f \in F, \forall e \in E_f, \forall t \in T_e, \forall i \in [1, T_f], \forall k \in [t; t + w_{fe}] \tag{3.9}$$

Figure 3.1 shows the value of the decision variables X and Y for the first flow iteration over the network interface $e_2$.

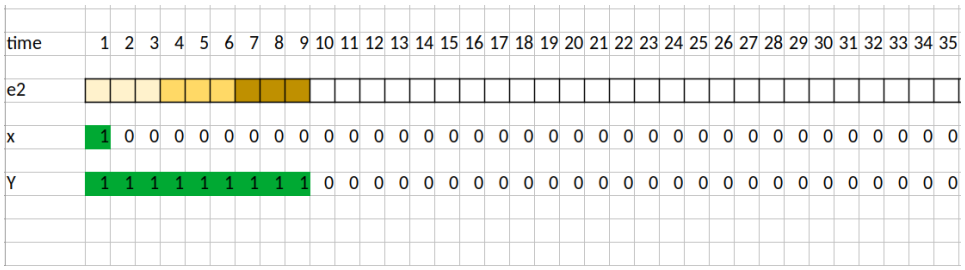



Figure 3.1: Assignment of $w_{fe}$ contiguous time slots for each iteration of a flow $f$.

4. The delay $d_{fi}$ iteration $i$ of a flow $f$ is evaluated as the difference between the time in which the transmission of the frame in the destination hop starts $x_{fe_{dest}ti}$ and the time in which the physical signal produces the first bit. This approximates the transmission delay and evaluates the only contribution the scheduling algorithm gives as explained by **SH_1**. The delay is measured in the number of time slots according to the time granularity of the destination-directed link.

$$d_{fi} = \sum_{t \in T_{e_{dest}}} (t-1) * x_{fe_{dest}ti} - \frac{P_f}{\tau_{e_{dest}}}(i-1) \; \forall f \in F, \forall i \in [1; T_f] \quad (3.10)$$

Figure 3.2 shows in two scenarios the estimation of the end-to-end delay (cells in green). For the first scenario $d_{fi} = 1$ and in the second scenario $d_{fi} = 18$. $d_{fi}$ approximates the delay in the number of time slots of the last network interface, thus it has to be converted into time units. Over the hypothesis that the last interface does not have a propagation delay, tn the first scenario, the approximated end-to-end delay is $d_{fi} * \tau_{e_2} = 8t.u.$, in the second scenario the delay is $18t.u.$

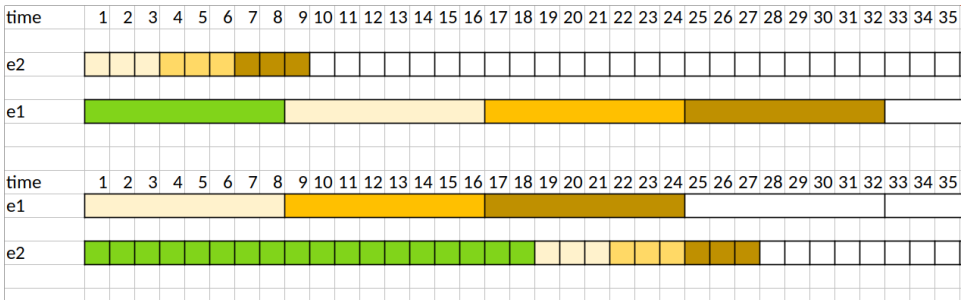



Figure 3.2: End-to-end delay estimation examples

5. The jitter is the deviation from the true periodicity of a presumably periodic signal. According to this and the simplification hypothesis

| row | $\sum_{i=1}^{T_f} x_{feti}$ | $\sum_{i=1}^{T_f} s_{feti}$ | $c_{fet}$ |
|---|---|---|---|
| 1 | True | False | True |
| 2 | True | True | False |
| 3 | False | False | False |
| 4 | False | True | True |

Table 3.1: True table $c_{fet}$ definition

**SH_2**, the jitter for a flow $f$ could be evaluated as the variance of the transmission delay of the $T_f$ iterations over the $SF$. The general mathematical definition of the variance of an aleatory variable X of mean $\mu$ is $E\left[(X-\mu)^2\right]$. This definition is quadratic, not linear against **MSH_2**. Minimizing the variance is equivalent to minimising the distance between the maximum and minimum delay for a flow $f$. For this reason, the jitter could be approximated as the difference between maximum and minimum delay for different iterations $i_a, i_b$ of the same flow $f$, according to the following linear relationship:

$$j_f \geq d_{fi_2} - d_{fi_1} \ \forall f \in F, \forall i_1, i_2 \, s.t. 1 \leq i_1 \leq i_2 \leq T_f, d_{fi_1} \leq d_{fi_2} \quad (3.11)$$

Note that according to the equation, iff $T_f = 1$ then $j_f$ is equal to 0 (**MSH4**).

6. In time-sensitive protocol the delay for each iteration of the flow $f$ is time-constrained according to its maximum acceptable delay $\delta_f$.

$$\tau_{e_{dest}} * d_{fi} + d_{e_{dest}} \leq \delta_f \ \ \forall f \in F, \ \forall i \in [1; T_f] \quad (3.12)$$

7. In time-sensitive protocol the jitter of the flow $f$ is time-constrained according to its maximum acceptable jitter $\upsilon_f$.

$$\tau_{e_{dest}} * j_f \leq \upsilon_f \ \ \forall f \in F \quad (3.13)$$

8. The decision variable $c_{fet}$ evaluates the differences between the new scheduling plan and the incumbent one.

   In particular, it evaluates if there is a *scheduling change* for the flow $f$ over $(e, t)$. True table 3.1 defines the concept of the *scheduling change*. Figure 3.3 shows an example of reconfiguration of the network and the values of the $c_{fet}$ decision variable.

   - Row 1: If in the new scheduling plan $f$ is allocated over $(e, t)$ and $f$ is not allocated over $(e, t)$ in the incumbent configuration there is a change, thus $c_{fet}$ is True.

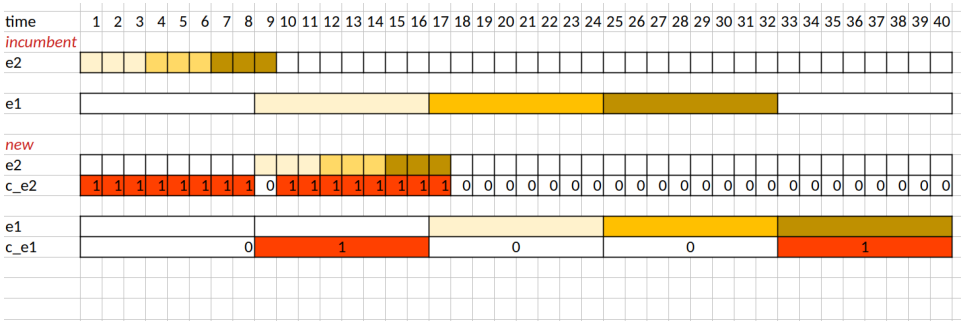



Figure 3.3: An example of reconfiguration of the network between a new data plan and an incumbent one. The cells in red mark a change in the data plane. They are mapped over a true value of the decision variable $c_{fet}$.

$$\sum_{i=1}^{T_f} x_{feti} = 1 \wedge \sum_{i=1}^{T_f} s_{feti} = 0 \Rightarrow c_{fet} = 1$$

$$\neg(\sum_{i=1}^{T_f} x_{feti} = 1 \wedge \sum_{i=1}^{T_f} s_{feti} = 0) \vee c_{fet} = 1$$

$$\neg(\sum_{i=1}^{T_f} x_{feti} \wedge \neg \sum_{i=1}^{T_f} s_{feti}) \vee c_{fet}$$

Applying De Morgan law

$$(\neg \sum_{i=1}^{T_f} x_{feti} \vee \sum_{i=1}^{T_f} s_{feti}) \vee c_{fet}$$

The logical equation can be rewritten as a linear equation

$$[(1 - \sum_{i=1}^{T_f} x_{feti}) + \sum_{i=1}^{T_f} s_{feti}] + c_{fet} \geq 1 \quad (3.14)$$

The final equation is:

$$c_{fet} \geq \sum_{i=1}^{T_f} (x_{feti} - s_{feti}) \quad \forall f \in F, \forall e \in E_f, \forall t \in T_e \quad (3.15)$$

- Row 2: If in the new scheduling plan, $f$ is allocated over $(e, t)$ and it is allocated over $(e, t)$ in the incumbent one there is not a change, thus $c_{fet}$ is False. We proceed as in the previous equation definition, the complete demonstration is available in the Appendix 7.1.1.

The final equation is:

$$c_{fet} + \sum_{i=1}^{T_f}(x_{feti} + s_{feti}) \leq 2 \quad \forall f \in F,\ \forall e \in E_f,\ \forall t \in T_e \quad (3.16)$$

- Row 3: If in the new scheduling plan, $f$ is not allocated over $(e,t)$ and also $f$ is not allocated over $(e,t)$ in the incumbent one there is not a change, thus $c_{fet}$ is False. The demonstration of the constraint definition is available in the Appendix 7.1.1.

  The final equation is:

$$c_{fet} \leq \sum_{i=1}^{T_f}(x_{feti} + s_{feti}) \quad \forall f \in F,\ \forall e \in E_f,\ \forall t \in T_e \quad (3.17)$$

- Row 4: If in the new scheduling plan, $f$ is not allocated over $(e,t)$ and $f$ is allocated over $(e,t)$ in the incumbent one there is a change, thus $c_{fet}$ is True. The demonstration of the constraint definition is available in the Appendix 7.1.1. The final equation is:

$$c_{fet} \geq \sum_{i=1}^{T_f}(s_{feti} - x_{feti}) \quad \forall f \in F,\ \forall e \in E_f,\ \forall t \in T_e \quad (3.18)$$

9. Given an iteration $i$ of the flow $f$ and $e_1, e_2 \in E_f$, the scheduling of the window frame over $e_2$ cannot start before the minimum pipeline time according to the heterogeneous space transformation problem discussed in the section 2.6.3.

$$\sum_{t \in T_{e_2}} (t-1) * x_{fe_2ti} \geq pipeline(e_1, e_2) + \sum_{t \in T_{e_1}} (t-1) * x_{fe_1ti}$$
$$\forall f \in F,\ \forall i \in [1, T_f],\ \forall (e_1, e_2) \in E_f \quad (3.19)$$

10. Given two iterations $i_1$ and $i_2$ of the same flow $f$ s.t. $i_1 < i_2$ over a link $e \in E_f$, $i_1$ starts in time slot $t_1$ and $i_2$ starts in $t_2$ s.t. $t_1 < t_2$. This constraint expresses the precedence between iterations of the same flow.

$$\sum_{t \in T_e} t * x_{feti_2} \geq \sum_{t \in T_e} t * x_{feti_1} + 1$$
$$\forall f \in F, \forall i_1, i_2\ s.t.\ 1 \leq i_1 < i_2 \leq T_f,\ \forall e \in E_f \quad (3.20)$$

11. Z is the maximum between the jitter of the flows of the SF.

$$Z \geq j_f * \tau_{last} \ \forall f \in F, last \leftarrow E_f[-1] \tag{3.21}$$

12. W is the maximum between the delay of the flows of the SF.

$$W \geq d_{fi} * \tau_{last} + d_{last} \ \forall f \in F, \forall i \in [1, T_f], last \leftarrow E_f[-1] \tag{3.22}$$

**Constraints optimization and combination**

To improve the performance of the algorithm is possible a-priori restrict the solution space according to the following criteria:

1. Given a flow $f$ and a time slot $t$ over $e \in E_f$, $t$ is a feasible starting point for $f$ over $e$ if it is free and there is a sequence of at least $w_{fe} - 1$ time slots contiguous to $t$.

2. Given a flow $f$ the scheduling of the iteration $i$ has to start after $P_f * (i-1)t.u.$.

According to the previous criteria is possible to precompute the matrix $N = \{\nu_{feti}\}$ where $\nu_{feti}$ is equal to 1 if the time slot $(e, t)$ is a feasible starting time slot for the iteration $i$ of the flow $f$. The algorithm 5 defines the pre-computation for $N$.

**Algorithm 5** ILP required pre-computation : ilp_precomputing()

**Require:** $F, E_f, T_e \ \forall e \in E, W$

```
N ← {0} //init to 0 each element of the matrix
for f ∈ F do
    for e ∈ E_f do
        mask ← [1 * w_fe]
        for i ∈ [1, T_f] do
            lower ← P_f * (i − 1)
            for t ∈ T_e.bitsearch(mask) do
                if (t − 1) * τ_e ≥ lower then
                    ν_feti ← 1
                end if
            end for
        end for
    end for
end for
    return < N >
```

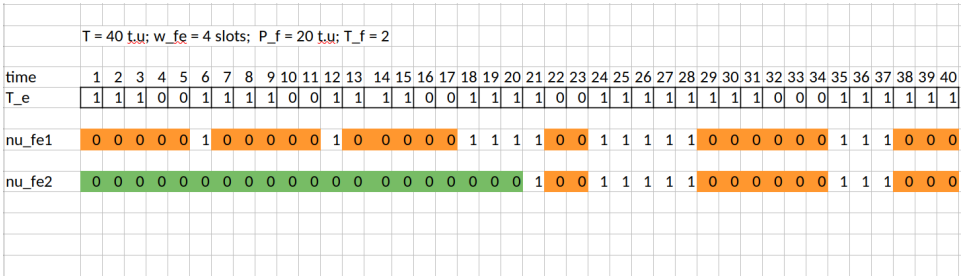



Figure 3.4: The optimization of the space of solution according to the previous criteria. In orange, the space was removed according to optimization criteria 1 and in green the space was removed according to optimization criteria 2.

According to the previous pre-computation, the model is simplified in solving complexity and the following constraint is added [1]

$$x_{feti} \leq \nu_{feti} \quad \forall f \in F, \forall e \in E_f, \forall i \in [1, T_f], \forall t \in T_e \tag{3.23}$$

Figure 3.4 shows an example of space reduction. The SF duration is equal to 40 t.u., the $w_{fe} = 4$, period of the flow $P_f = 10$, thus $T_f = 2$ iterations. The first row refers to the first iteration of the time-critical request, and the second row to the second iteration.

### 3.1.3 Circular shifting optimization

Inspired by the circular buffer, it is possible to treat the SF as a circular frame. In this way, it is possible to achieve the acceptance of a higher number of requests [2]. The optimization is achieved considering the scheduling for two consecutive super frames $SF = \{SF_1 \cup SF_2\}$ [3] and then in post-processing merging them. Figure 3.5 shows an example of a circular shifting optimization. Two slots of different $SF$ of a link $e$ are defined complements if their distance is equal to $t_e$. Two complements have to be mutually allocated to guarantee a correct post-processing:

$$y_{fe(t+t_e)i} = 1 \Rightarrow y_{feti} = 0, \;\; t \in [1, t_e] \tag{3.24}$$

[1] Adding this constraint simplifies the model because set to 0 is the initial value for all non-priori feasible space of solution. This constraint is not included in the problem size evaluation for this reason.

[2] The optimization solves the problem for each flow that starts in the SF but given its duration cannot complete the allocation in the current SF. With circular fashion allocation the remaining scheduling windows are allocated in the next SF, thus in the circular correspondents time slots.

[3] This strategy increases the solving time complexity, thus this is not always a good application. The scenario which is correct for adopting this optimization is the presence in the network of an old or high-loaded interface.

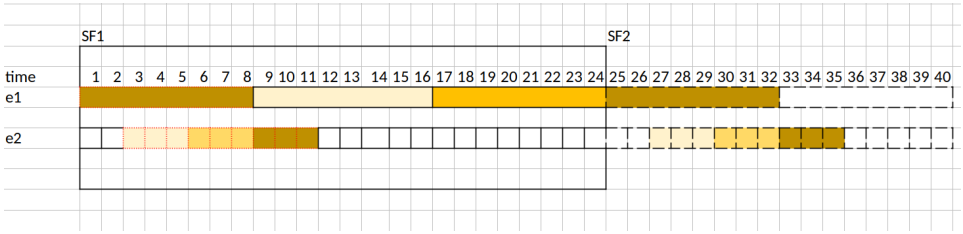



Figure 3.5: Circular shifting optimization example.

The logical condition is expressed by the following additional constraint:

$$\sum_{f \in F} \sum_{i \in [1, T_f]} [y_{feti} + y_{fe(t+t_e)i}] \leq 1 \quad \forall e \in E, \forall t \in T_e \tag{3.25}$$

The algorithm 6 post-processes the data plan to apply the circular shifting optimisation.

**Algorithm 6** ILP circular buffer merging : ilp_circularbuffermerge()

**Require:** $Y$

**for** $f \in F$ **do**
  **for** $e \in E_f$ **do**
    **for** $i \in [1, T_f]$ **do**
      **for** $t \in [t_e, 2 * t_e]$ **do** //considering two contiguous super frames.
        **if** $y_{feti} == 1$ **then**
          $y_{fe(t-t_e)i} \leftarrow 1$
          $y_{feti} \leftarrow 0$
        **end if**
      **end for**
    **end for**
  **end for**
**end for**

Figure 3.5 shows an example of circular shifting optimization. There are two SF of a duration of 24 t.u. The scheduling windows allocated in $SF_2$ with a dashed border are circularly shifted in post-processing in $SF_1$ to the time slot with a dashed red border.

### 3.1.4 Objective function

The objective function is formulated according to the objectives of the optimization problem described in Section 2.2. The following equation is the mathematical formulation

$$minimize \quad p_1 * Z + p_2 * W + p_3 * \sum_{f \in F} \sum_{e \in E} \sum_{t \in T_e} c_{fet} \tag{3.26}$$

where $P = \{p_1, p_2, p_3\}$ is the weights set defined as follows to guarantee the different relevance for the different objectives as defined in the section 2.2.

$$p_1 = \frac{|P|}{max_{f \in F} \upsilon_f} \tag{3.27}$$

$$p_2 = \frac{|P| - 1}{max_{f \in F} \delta_f} \tag{3.28}$$

$$p_1 = \frac{|P| - 2}{\sum_{f \in F} \sum_{e \in E} \sum_{t \in T_e} 1} \tag{3.29}$$

Each weight contains a normalization factor at the denominator which is the maximum value in the domain that the variable can assume and a scaling factor at the numerator that depends on the priority that the objective has to assume. In the model, the priority is linear according to the problem objectives declared.

Table 3.2 evaluates with the framework defined in 2.1 the quality of the ILP model. The following list describes per each objective group how the model evaluates archives the goal.

1. If a request is feasible the integer linear programming produces the optimal configuration, otherwise the request is blocked. Solving an ILP guarantees that the request is blocked only if infeasible. The first objective is already scored by the deterministic approach of the ILP.

2. According to the second group of objectives, the balanced loading between different links is guaranteed by the module that defines the provisioning over the network. The scheduling plane that the model describes is synchronous, thus the probability of overlaps in time and bandwidth waste is reduced.

3. The main objective of section three is to maximize the determinism of the network to guarantee the best performances and affordability of the estimated metrics. Maximizing determinism means minimizing the jitter, and the first additive component of the objective function ensures it. Minimizing the latency means also minimizing the delay, and the second additive component of the objective function ensures it.

4. The last additive component of the objective function ensures the problem-specific objective. The minimization in the number of changes between the incumbent configuration and the new computed. Given two or more possible configurations the algorithm takes the scheduling reconfiguration that reduces the number of required changes.

| **Objective id** | **Scored (Y/N/-)** |
|---|---|
| **Primary optimization objectives** | |
| 3.1 Maximize the determinism | Y (first term of the obj. function) |
| 3.2 Improve the transmission metrics | Y (second term of the obj. function) |
| **Secondary optimization objectives** | |
| 2.1 Maximize the load balancing | - (The input list $E_f$ contains the path of the flow $f$) |
| 2.2 Minimize the length of the path | N |
| 2.3 Minimize the fragmentation | N |
| 3.3 Maximize security | Y (not explicitly, including the required additional guards' time slots in the set $G_e$ per each network interface) |
| 3.4 Maximize robustness | Y (Not explicitly, including the required additional guard time slots in the set $G_e$ per each network interface) |
| **Thirdly objectives** | |
| 4.1 Minimize data plane update cost | Y (third term of the obj. function) |
| 4.2 Minimize solving time | N (integer linear programming is np-complete). |
| **Non optimization objectives** | |
| 1.1 Minimize topology cost | Y (Not explicitly, when the topology changes the minimization of the changes produced in the scheduling data plan limits the peak traffic produced by the update) |
| 1.2 Multicasting | N |

Table 3.2: ILP model with data plane reconfiguration, objectives framework evaluation

### 3.1.5 Model size

According to 3.1 is possible to define the size of the problem. The table 3.3 approximates it. In the worst case, the following hypotheses are valid:

- The maximum number of accepted flows with a load of the network of 100% is $|F| \approx |E|$. Over each time unit one flow is scheduled over one network interface.
- The minimum period of a request must be lower than or equal to the duration of a time unit. According to this $max_{f \in F}(T_f) \leq max_{e \in E}(\frac{T}{\tau_e})$
- The maximum length of a scheduling window has to be lower than or equal to the maximum number of time slots of the interfaces, thus $max_{f \in F, e \in E_f}(w_{fe}) \leq max_{e \in E}(\frac{T}{\tau_e})$.
- The following equation is valid: $max_{e \in E}(\frac{T}{\tau_e}) = \frac{T}{min_{e \in E}\tau_e} = \frac{T * max(B_e)}{min(a_e)}$

To summarize the size of the model is approximated to:

$$size \approx |E|^4 * (\frac{T * max(B_e)}{min(a_e)})^5 \tag{3.30}$$

In conclusion, the size of the model depends on the topology of the network (factor $|E|$), it increases with the increasing of the number of edges, and the granularity of the scheduling. In detail, granularity depends on the number of time units $T$ and the maximum throughput and the minimum quantum of bit transmitted per time slot. A higher granularity is synonymous with lower bandwidth waste, increasing the parameter $a_e$ means increasing the wasted throughput but reducing the complexity of the algorithm.

Given a real scenario with a maximum throughput of 10Gbits and a packet size of 90Bytes, figure 3.6 shows the graph of the size function. The function explodes in complexity with increasing granularity of the time unit, in detail with $T \geq 6 * 10^5$.

## 3.2 Scheduling without reconfiguration

In this section the implementation of the ILP model to solve the problem without the reconfiguration of the network data plane. The problem is treated as a simplification of the previous statement in which the cardinality of the set of the TS-requests is 1.

[5]The main contribution is given by the maximum number of time slots of network interfaces. According to this is possible to proceed with the approximation.

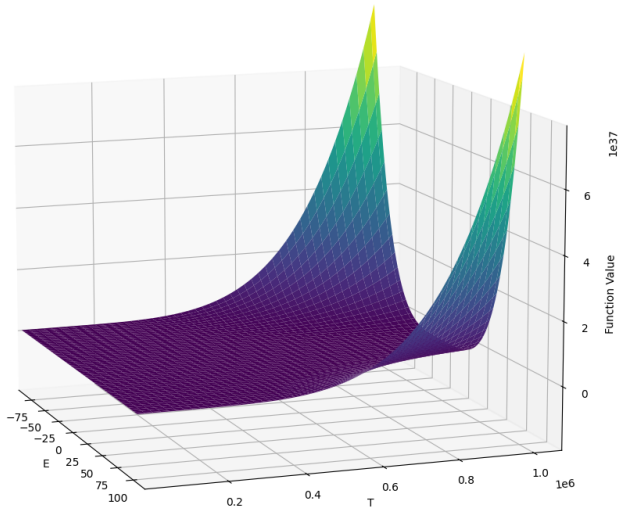



Figure 3.6: Size of the ILP model with a reconfiguration for a real scenario, frame of 90 Bytes and throughput less than or equal to 10Gbits.

| **Decision variables (a.k.a space complexity)** | |
|---|---|
| X | $\|F\| * \|E\| * max_{e \in E}(\frac{T}{\tau_e}) * max_{f \in F} T_f$ |
| Y | $\|F\| * \|E\| * max_{e \in E}(\frac{T}{\tau_e}) * max_{f \in F} T_f$ |
| C | $\|F\| * \|E\| * max_{e \in E}(\frac{T}{\tau_e})$ |
| D | $\|F\| * T_f$ |
| J | $\|F\|$ |
| $\approx O(X)$ | $O(\|F\| * \|E\| * max_{e \in E}(\frac{T}{\tau_e}) * max_{f \in F} T_f)^4$ |
| **Constraints** | |
| C1 | $\|E\| * max_{e \in E}(\frac{T}{\tau_e})$ |
| C2 | $\|F\| * \|E\| * max_{f \in F} T_f$ |
| C3 | $\|F\| * \|E\| * max_{f \in F} T_f * max_{e \in E}(\frac{T}{\tau_e}) * max_{f \in F, e \in E_f} w_{fe}$ |
| C4 | $\|F\| * max_{f \in F} T_f$ |
| C5 | $\|F\| * max_{f \in F} T_f$ |
| C6 | $\|F\| * max_{f \in F} T_f$ |
| C7 | $\|F\|$ |
| C8 | $3 * \|F\| * \|E\| * max_{e \in E}(\frac{T}{\tau_e})$ |
| C9 | $\|F\| * 2 * \|E\| * max_{f \in F} T_f$ |
| C10 | $\|F\| * \|E\| * 2 * max_{f \in F} T_f$ |
| C11 | $\|F\|$ |
| C12 | $\|F\| * max_{f \in F} T_f$ |
| $\approx O(A)$ | $O(\|F\| * \|E\| * max_{f \in F} T_f * max_{e \in E}(\frac{T}{\tau_e}) * max_{f \in F, e \in E_f} w_{fe}{}^5$ |
| $\approx$ | $O(A) * O(X) = \|F\|^2 * \|E\|^2 * max^2_{e \in E}(\frac{T}{\tau_e}) * max_{f \in F} T_f * max_{f \in F, e \in E_f} w_{fe}$ |

Table 3.3: ILP model with data plane reconfiguration, problem size computation

### 3.2.1 Decision Variables

In the following section are described the decision variables defined by the model.

- The starting time slot binary matrix

$$X = \{x_{eti}\} \tag{3.31}$$

  where $1 \leq e \leq |E|, 1 \leq t \leq |T_e|, 1 \leq i \leq T_f$. $x_{eti} = 1$ iff the start of the window allocation for the iteration $i$ in the network interface $e$ starts at the time slot $t$, 0 otherwise.

- The estimated delay continuous matrix

$$D = \{d_i\} \tag{3.32}$$

  where $1 \leq i \leq T_f$. $d_i$ contains information about the E2E delay for the transmission of the flow at the iteration $i$ defined according to **SH_1**. Its granularity depends on the duration of time slot $\tau_{dest}$ over the last directed link *dest* of the path of *f*.

- A continuous variable J that indicates the maximum estimated jitter. It is introduced to balance and minimize the jitter. It is evaluated over the subset of $D$ according to **SH_2**.

- A continuous variable W that indicates the maximum estimated delay. It is introduced to balance and minimize the delay.

### 3.2.2 Constraints

1. Each time slot $t$ of the network interface $e$, $(e, t)$ could be assigned only once.

$$\sum_{i \in T_f} x_{feti} \leq 1 \quad \forall e \in E, \forall t \in T_e \tag{3.33}$$

2. Each iteration $i$ has to be scheduled once in all the directed links of the path of the flow $E_f$.

$$\sum_{t \in T_e} x_{eti} = 1 \quad \forall f \in F, \forall e \in E_f, \forall i \in [1; T_f] \tag{3.34}$$

3. Given an iteration $i$ of the flow $f$ and $e_1, e_2 \in E_f$, the scheduling of the window frame over $e_2$ cannot start before the minimum pipeline time according to the heterogeneous space transformation problem discussed in the section 2.6.3.

$$\sum_{t \in T_{e_2}} (t-1) * x_{e_2 t i} \geq pipeline(e_1, e_2) + \sum_{t \in T_{e_1}} (t-1) * x_{e_1 t i} \quad \forall i \in [1, T_f],\ \forall (e_1, e_2) \in E_f \quad (3.35)$$

4. The delay $d_i$ of the iteration $i$ is evaluated as the difference between the time in which the transmission of the frame in the destination hop starts $x_{e_{dest} t i}$ and the time in which the physical signal produces the first bit. This approximates the delay according to **SH_1**. The delay is measured in the number of time slots according to the time granularity of the destination-directed link.

$$d_i = \sum_{t \in T_{e_{dest}}} (t-1) * x_{e_{dest} t i} - \frac{P_f}{\tau_{e_{dest}}}(i-1) \quad \forall i \in [1; T_f] \quad (3.36)$$

5. W is the maximum between the delay of the flows of the SF.

$$W \geq d_i * \tau_{dest} + d_{dest} \ \ \forall i \in [1, T_f], dest \leftarrow E_f[-1] \quad (3.37)$$

6. In time-sensitive protocol the delay for each iteration of the flow is time-constrained according to its maximum acceptable delay $\delta_f$.

$$W \leq \delta_f \quad (3.38)$$

7. The jitter is the deviation from the true periodicity of a presumably periodic signal. According to simplification hypothesis **SH_2**, the jitter for a flow $f$ could be evaluated as the variance of the transmission delay of the $T_f$ iterations over the $SF$. As discussed for the previous model the jitter could be approximated as the difference between maximum and minimum delay for different iterations $i_a, i_b$ of the flow, according to the following linear relationship:

$$j \geq d_{i_2} - d_{i_1} \quad \forall i_1, i_2 \ s.t. \ 1 \leq i_1 \leq i_2 \leq T_f,\ d_{i_1} \leq d_{i_2} \quad (3.39)$$

8. In time-sensitive protocol the jitter is time-constrained according to its maximum acceptable jitter $\upsilon_f$.

$$\tau_{e_{dest}} * j \leq \upsilon_f \quad (3.40)$$

9. Given two iterations $i_1$ and $i_2$ of the same flow $f$ s.t. $i_1 < i_2$ over a link $e \in E_f$, $i_1$ starts in time slot $t_1$ and $i_2$ starts in $t_2$ s.t. $t_1 < t_2$. This

constraint expresses the precedence between iterations of the same flow.

*This constraint is implicitly satisfied in a scenario in which the scheduler has to handle only one request to minimize the non-determinism.*

#### Constraints optimization and combination

As described for the ILP model with network reconfiguration, it is possible to precompute the matrix $N = \{\nu_{eti}\}$ where $\nu_{eti}$ is equal to 1 if the time slot $(e, t)$ is a feasible starting time slot for the iteration $i$. The algorithm 7 defines the pre-computation for $N$.

**Algorithm 7** ILP required pre-computation in scheduling without reconfiguration : ilp_precomputing2()

**Require:** $F, E_f, T_e \; \forall e \in E, W$
1: $N \leftarrow \{0\}$ //init to 0 each element of the matrix
2: **for** $e \in E_f$ **do**
3: $mask \leftarrow [1 * w_{fe}]$
4: **for** $i \in [1, T_f]$ **do**
5: $lower \leftarrow P_f * (i - 1)$
6: **for** $t \in T_e.bitsearch(mask)$ **do**
7: **if** $(t - 1) * \tau_e \geq lower$ **then**
8: $\nu_{eti} \leftarrow 1$
9: **end if**
10: **end for**
11: **end for**
12: **end for**
**return** $< N >$

According to the previous pre-computation, the model is simplified in solving complexity and the following constraint is included in the model[6]

$$x_{eti} \leq \nu_{eti} \quad \forall e \in E_f, \forall i \in [1, T_f], \forall t \in T_e \tag{3.41}$$

### 3.2.3 Circular shifting optimization

Inspired by the circular buffer, it is possible to treat the SF as a circular frame. In this way, it is possible to accept a higher number of requests. The problem is described in the previous model implementation, in this section it is adapted to work also for the ILP model without reconfiguration. Two

[6] Adding this constraint simplifies the model because set to 0 is the initial value for all non-priori feasible space of solution. This constraint is not included in the problem size evaluation for this reason.

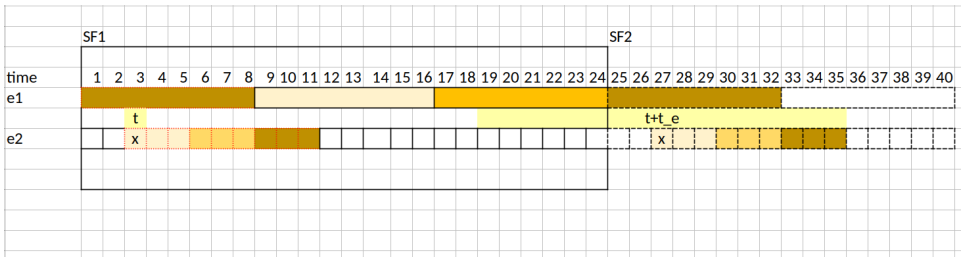



Figure 3.7: Circular shifting optimization example

complements [7] have to be mutually allocated to guarantee a correct post-processing:

$$x_{eki} = 1 \Rightarrow x_{eti} = 0, \;\; t \in [1, t_e] \;\; \forall k \in [t + t_e - w_{fe}, t + t_e + w_{fe} - 1] \quad (3.42)$$

Note that in this scenario the mutual exclusion condition is applied from one time slot to a set of time slots. The following constraint overloads C1:

$$\sum_{i \in [1, T_f]} x_{eti} + \sum_{i \in [1, T_f]} \sum_{k=t+t_e-w_{fe}}^{t+t_e+w_{fe}-1} x_{eti} \leq 1 \quad \forall e \in E, \, \forall t \in T_e \quad (3.43)$$

Figure 3.7 shows an example of circular shifting optimization. The example is the same as figure 3.5: there are two SF of a duration of 24 t.u. The scheduling windows allocated in $SF_2$ with a dashed border are circularly shifted in post-processing in $SF_1$ to the time slot with a dashed red border. To guarantee the mutual exclusion assignation of resources in post-processing, extending the window of not-compatible configuration is required. If the slot $t$ is allocated then the slots marked in yellow have to be available.

The algorithm 8 post-processes the data plane to apply the circular shifting optimisation.

### 3.2.4 Objective function

The objective function is formulated according to the optimization problem objectives described in the section 2.2. The following equation is the mathematical formulation

$$minimize \;\; j \quad (3.44)$$

[7] Comparing the two implementations of the circular shifting optimization, the first developed for scheduling with reconfiguration and the second developed for scheduling without reconfiguration, the definition of complement is different. In both is a time slot with a distance of $t_e$ time slots, however for the scheduling without reconfiguration, this definition is limited to the starting time slot of a scheduling window, thus the optimization has a granularity of $w_{fe}$ contiguous time slots.

**Algorithm 8** ILP circular buffer merging in scheduling without reconfiguration : ilp_circularbuffermerge2()

**Require:** $Y$
1: **for** $e \in E_f$ **do**
2: **for** $i \in [1, T_f]$ **do**
3: **for** $t \in [t_e, 2 * t_e]$ **do** //considering two contiguous super frames.
4: **if** $x_{eti} == 1$ **then**
5: $x_{e(t-t_e)i} \leftarrow 1$
6: $x_{eti} \leftarrow 0$
7: **end if**
8: **end for**
9: **end for**
10: **end for**

Table 3.4 evaluates with the framework defined in 2.1 the quality of the produced ILP model. The following list describes for each objective group how the model evaluates archives the goal.

1. Solving an ILP guarantees that the request is blocked only if infeasible. The first objective is already scored by the deterministic approach of the ILP.

2. As for the previous model: according to the second group of objectives, the balanced loading between different links is guaranteed by the module that defines the provisioning over the network. The scheduling plane that the model describes is synchronous, thus the probability of overlaps in time and bandwidth waste is reduced.

3. The main objective of section three is to maximize the determinism of the network to guarantee the best performances and affordability of the estimated metrics. Maximizing determinism means minimizing the jitter, this is the aim of the objective function. This may be non-optimal because in this model is not possible to reconfigure the data plane (a.k.a. shifting previous allocations).

4. The minimization of the number of changes between the incumbent configuration and the new computed is not achieved, because there is no comparison between the new and incumbent data plane.

### 3.2.5 Model size

According to 3.1 is possible to define the size of the problem. The table 3.5 approximates it. In the worst case, the following hypotheses are valid:

- The minimum period of a request has to be $P_f \leq 1$. According to this $max_{f \in F}(T_f) \leq max_{e \in E}(\frac{T}{\tau_e})$

| Objective id | Scored (Y/N/-) |
|---|---|
| **Primary optimization objectives** | |
| 3.1 Maximize the determinism | Y (first term of the obj. function) |
| 3.2 Improve the transmission metrics | N |
| **Secondary optimization objectives** | |
| 2.1 Maximize the load balancing | - (The input list $E_f$ contains the path of the flow $f$) |
| 2.2 Minimize the length of the path | N |
| 2.3 Minimize the fragmentation | N |
| 3.3 Maximize security | Y (not explicitly, including the required additional guards' time slots in the set $G_e$ per each network interface) |
| 3.4 Maximize robustness | Y (Not explicitly, including the required additional guards' time slots in the set $G_e$ per each network interface) |
| **Thirdly objectives** | |
| 4.1 Minimize data plane update cost | N |
| 4.2 Minimize solving time | N (integer linear programming is np-complete). |
| **Non optimization objectives** | |
| 1.1 Minimize topology cost | N |
| 1.2 Multicasting | N |

Table 3.4: ILP model without data plane reconfiguration, objectives framework evaluation

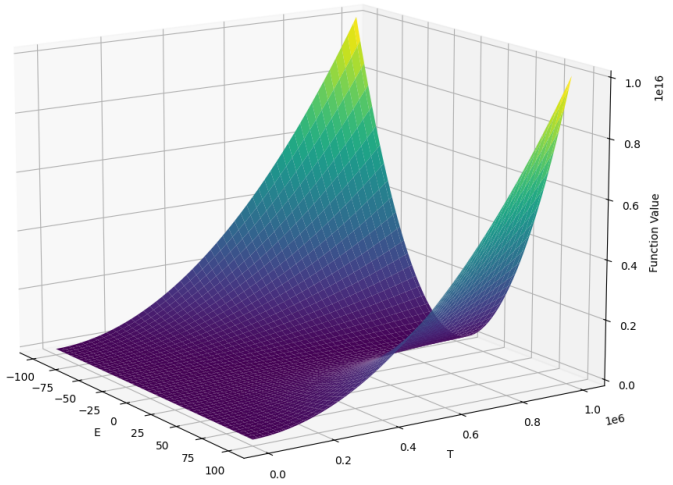



Figure 3.8: Size of the ILP model without reconfiguration for a real scenario, frame of 90 Bytes and throughput less than or equal to 10Gbits.

- The following equation is valid: $max_{e \in E}(\frac{T}{\tau_e}) = \frac{T}{min_{e \in E}\tau_e} = \frac{T * max(B_e)}{min(a_e)}$

To summarize the complexity of the model is approximated to:

$$size \approx |E|^2 * (\frac{T * max(B_e)}{min(a_e)})^2 \tag{3.45}$$

In conclusion, as for the model with the reconfiguration the size of the model depends on the topology of the network (factor $|E|$), it increases with the increasing of the number of edges, and the granularity of the scheduling. In detail, granularity depends on the number of time units $T$, the maximum throughput of a network interface, and the minimum quantum of bit transmitted per time slot.

Given the scenario proposed for the previous model with a maximum throughput of 10Gbits and a packet size of 90Bytes, figure 3.8 shows the graph of the size function. The function explodes in complexity with the increase of the time unit granularity, in detail with $T \geq 4 * 10^5$. Comparing the previous model with the scheduling without reconfiguration the dimension of the size value has a scale of $10^{16}$ instead of $10^{37}$, the simplification of the model has produced an asymptotic reduction of the size of the problem of a factor $10^{21}$.

[8]The main contribution is given by the maximum number of time slots of network interfaces. According to this is possible to proceed with the approximation.

| **Decision variables (a.k.a space complexity)** | |
|---|---|
| X | $\|E\| * max_{e \in E}(\frac{T}{\tau_e}) * \frac{T}{P_f}$ |
| D | $\frac{T}{P_f}$ |
| $\approx O(X)$ | $O(E\| * max_{e \in E}(\frac{T}{\tau_e}) * \frac{T}{P_f})^8$ |
| **Constraints** | |
| C1 | This constraint reduces the solution space, it doesn't increase the size complexity |
| C2 | $\|E\| * max_{e \in E}(\frac{T}{\tau_e})$ |
| C3 | $\|E\| * T_f$ |
| C4 | $\|E\| * T_f$ |
| C5 | $\|E\| * T_f$ |
| C6 | $T_f$ |
| C7 | $\|E\|$ |
| $\approx O(A)$ | $O(\|E\| * 2 * max_{e \in E}(\frac{T}{\tau_e})$ |
| $\approx$ | $O(A) * O(X) = \|E\|^2 * max_{e \in E}(\frac{T}{\tau_e})^2$ |

Table 3.5: ILP model without reconfiguration, problem size computation

# Chapter 4

# Heuristic model

In this chapter, starting from [19] and [13] a Greedy + LS TAS scheduling algorithm. According to the problem statement section, the algorithm is split into two parts:

- Constructive phase: given a set of flows as input and an empty scheduling data plane generate the local best scheduling. This section solves only the scheduling without reconfiguration problem for one or multiple requests in $F$.

- LS phase: given a new request that cannot be allocated in the incumbent scheduling plan, explore the optimal local solutions that can accept the new scheduling closer to the incumbent scheduling plan. This section solves the scheduling with the data plane reconfiguration problem.

The greedy algorithm is characterized by the components described in table 4.1.

## 4.1 Constructive phase

The problem can be split into two sub-problems:

- Find the local optimal starting time slot $t$ over the source network interface of the path.

- Given the starting time slot $t$ define the optimal allocation for balancing and minimizing the scheduling delay.

Starting from the first problem, to find the first allocation time slot $t_{start}$ over the source network interface of the path $E_f[0]$ the following heuristic criteria restrict the searching area in the space of solutions:

| Symbol | Description |
|---|---|
| $\Omega$ | TAS solves a combinatorial problem. $\Omega$ is the space of solution, it contains all the combinations between TS-flows, network interfaces, time slots and iterations. |
| $C = \{c_{fi}\}$ | C is the set of candidates. A candidate $c_{fi}$ for the flow $f$ ad the iteration in SF $i$ is a sorted list of tuples $(e, t)$ where $e \in E_f$ and $t \in T_e$. The order is given by the key $e$ according to the path of $f$. |
| $cost(.)$ | Each candidate has a cost that is represented by $\Delta_j$, which is the variation of the flow's jitter if the candidate is chosen, $delay$, which is the e2e estimated delay and $t_{start}$ which is the first allocated time slot in the source link of the path. |
| $S \subset \Omega$ | The solution of the scheduling problem. Per each iteration of each flow, one candidate is chosen. |

Table 4.1: Heuristic components definition

1. Resources cannot be allocated to the TS-flow before the signal has generated the first bit. According to **SH_1** the following equation declares the heuristic criteria:

$$\tau_{E_f[0]} * (t_{start} - 1) \geq P_f * (i - 1) \tag{4.1}$$

2. The minimum flow latency plus the starting scheduling delay has to be lower than the maximum admitted delay $\delta_f$

$$starting_{delay} + latency_f \leq \delta_f \tag{4.2}$$

   Where the minimum latency is evaluated in the case of express transmission with $h_{e_1e_2} = 1$ [1] according to algorithm 9. The starting delay is the time in which the packet available for transmission has to wait in the buffer of the source interface defined as:

$$starting_{delay} = \tau_{E_f[0]} * (t_{start} - 1) - P_f * (i - 1) \tag{4.3}$$

3. To allocate $T_f$ iterations of the flow $f$ is required that at iteration $i$:

$$T - \tau_{E_f[0]} * (t_{start} + w_{fe}) \geq \tau_{E_f[0]} * w_{fe} * (T_f - i) \tag{4.4}$$

   This criterion guarantees that there is space to allocate all iterations of the flow $f$ over the source network interface.

[1] Reference about the coefficient definition 2.6.2

**Algorithm 9** Min latency for a flow $f$ : latency()

**Require:** $E_f$
1: $latency \leftarrow 0$
2: **for** $e \in E_f$ **do**
3: $\quad latency \leftarrow latency + \tau_e + d_e$
4: **end for**
**return** $< latency >$

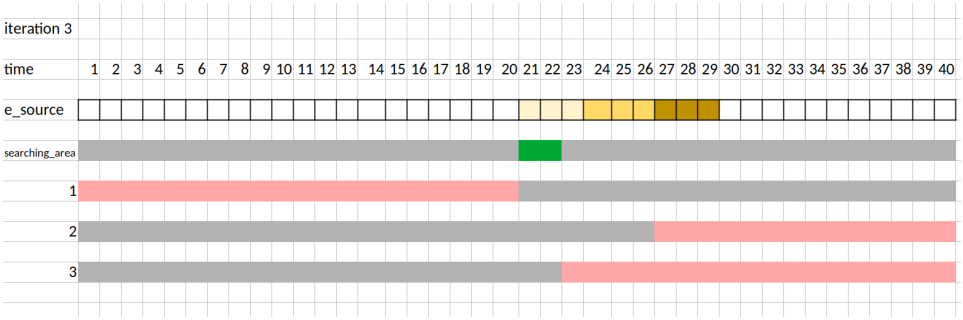



Figure 4.1: Starting time slot searching problem, space reduction example.

Figure 4.1 shows an example of the problem with pruning of the searching space according to the previous heuristic conditions. In the figure $T = 40t.u.$, $P_f = 10\ t.u$, $T_f = 3\ iterations$, $w_{fe} = 9\ slots$, $\tau_{E_f[0]} = 1$, the minimum latency is $3t.u.$ and $\delta_f = 9t.u.$ The final allocation of iteration 3 is marked in the $e_{source}$ row. In the searching area row: in green the possible starting time slots, in grey excluded according to the heuristic criteria. For each criterion, the cells in red are the pruned space. According to 1: $t_{start} - 1 \geq 10 * 2$; according to 2: $starting_{delay} + 3 \leq 9$; according to 3: $40 - (t_{start} + 9) \geq 9 * (4 - 3)$. Applying all the heuristic pruning conditions the problem, in this scenario, is reduced from a searching space of $40t.u.$ to only $2t.u.$ with a pruning ratio of 90%.

Turning on the second problem algorithm 10 implements the constrictive heuristic approach while algorithm 11[2] implements the generation of a candidate given a $t_{start}$, which is the output of the first problem. The *candidate()* function follows the first available allocation strategy which is explained in the next section about the research of patterns in a bit array starting from the time slot $t_{start}$. Line 4 updates the $t_{start}$ time slot according to the transformation problem for the next interface in the path. Overall, the goal is to allocate all the iterations of all requests while minimizing and balancing the estimated E2E delay. As declared in the problem

[2]The coefficient $l_{e,e_{next}}$ is defined in the *time division problem* section, it is a coefficient to pass between different time slots of different network interfaces at the corresponding time. The function $findFirt()$ is defined in the next section: *Time slots pattern search.*

statement section, the variance of the delay is the jitter of the transmission. For optimization purposes, its definition is simplified as the difference between the maximum and minimum delay of iterations of the same flow: model structure hypothesis 5 (**MSH 5**).

Each iteration $i$ of a flow $f$ has a maximum allowed delay $\delta_f$ and a maximum allowed jitter $v_f$. At the first iteration allocation, the objective is to find the best allocation to minimize the delay, for the following iterations incrementally the algorithm has to balance the delay to minimize the jitter. For each flow $f$, two thresholds $f.min\_delay$ and $f.max\_delay$ are defined, thus the space between them is the variance of the delay for $f$ in SF. When a candidate $c$ for the flow $f$ at iteration $i$ is generated the following metrics are computed:

- End-to-end estimated delay which is $cost(c).delay$
- Estimated variation of the space of variance of $f$ which is $cost(c).\Delta_j$.

The algorithm 12 implements the $cost(.)$ function. Algorithm 13 implements the comparator pattern to compare two candidates, thus selecting the best one, which minimizes the cost. When a candidate is selected in lines 22 and 25 the thresholds of the TS-flow are updated.

Figure 4.2 shows an example of the scheduler decisions for a TS request with 10 iterations in SF with randomly allocated network loading of 50%. The graph shows for each iteration the cost of the selected candidate. While points in red have a cost in terms of delay variance, points in green choose a feasible configuration inside the variance area between the two thresholds, thus they don't have a cost. E.g. iterations 6,8,9,10 have better candidates in terms of delay but they are not accepted to evict a jitter additional cost.

### 4.1.1 Time slots pattern search

For each network interface $e$, the array $T_e$ is represented as a bit array, in which the element at the index $(t-1)$ is equal to 1 iff the time slot $t \in [1, t_e]$ is available 0 otherwise. This is the heart of the solving time performance of the heuristic algorithm. The array is chunked in words of 8 bytes, in practice it is represented as a list of integers, each integer is represented over 32bits, thus each integer represents 32 time slots. For the implementation in python the library [11] provides all the required API.

E.g. representing with $t_e = 100$ requires a list of 4 integers, according to this the memory complexity has an enhancement of $x32$. To access the time slot $t = 31$, thus at the index 30 is required to access the first integer in the list with the following bit-mask $0x2$.

The algorithm 14 implements the function $findFirst(.)$ which returns the first available time slot over an interface $e$ such that it is followed by $w_{fe} - 1$ available time slots.

**Algorithm 10** Constructive phase for heuristic solution: construct()

**Require:** $F, I$ //Given a set of requests and a set of iterations for the requests

1: $S \leftarrow \{\}$
2: **for** $f \in F$ **do**
3: $f.min\_delay \leftarrow \delta_f$
4: $f.max\_delay \leftarrow 0$
5: $f.jitter \leftarrow 0$
6: **for** $i \in I$ **do**
7: $C \leftarrow \{\}$
8: $lt_{start} \leftarrow \frac{P_f*(i-1)}{\tau_{E_f[0]}+1}$
9: $ut2_{start} \leftarrow lt_{start} + \delta_f - latency(f)$
10: $ut3_{start} \leftarrow t_{E_f[0]} - w_{fE_f[0]} * (T_f - 2)$
11: **for** $t_{start} \in [lt_{start}, min(ut3_{start}, ut2_{start})]$ **do**
12: $C \leftarrow C \cup candidate(f, i, t_{start})$
13: **end for**
14: **if** $C == \{\}$ **then return** INFEASIBLE
15: **end if**
16: $c_{best} \leftarrow argmin_{c \in C} cost(c)$
17: $S \leftarrow S \cup \{c_{best}\}$
18: $assing(c_{best})$
19: $f.jitter \leftarrow f.jitter + cost(c_{best}).\Delta_j$
20: **if** $f.jitter > v.f$ **then return** INFEASIBLE
21: **end if**
22: **if** $f.min\_delay > cost(c_{best}).delay$ **then**
23: $f.min\_delay \leftarrow cost(c_{best}).delay$
24: **end if**
25: **if** $f.max\_delay < cost(c_{best}).delay$ **then**
26: $f.max\_delay \leftarrow cost(c_{best}).delay$
27: **end if**
28: **end for**
29: **end for**
30: $j \leftarrow max_{f \in F}(f.jitter)$
31: $d \leftarrow max_{f \in F}(f.max\_delay)$
**return** $< S, j, d >$

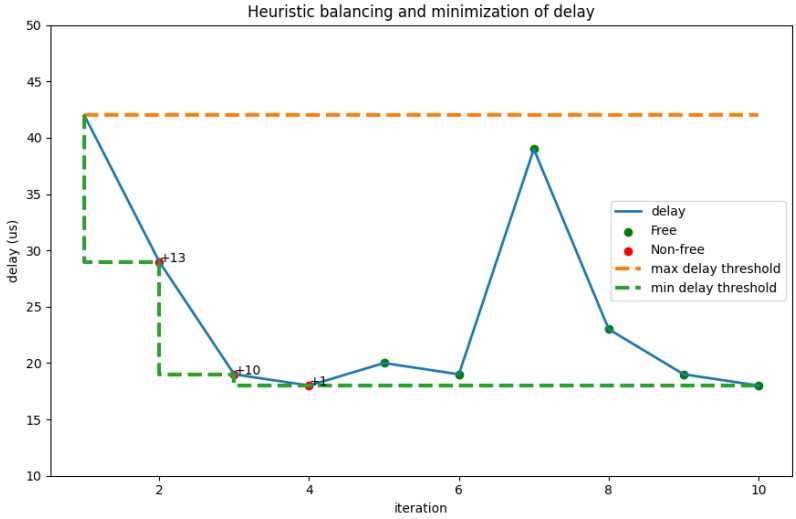



Figure 4.2: TAS heuristic scheduler best candidates example

**Algorithm 11** Candidate definition: candidate()

**Require:** $f, i, t_{start}$

**for** $e \in E_f$ **do**
  $e_{next} \leftarrow e + 1$
  $t \leftarrow findFirst(e, t_{start}, w_{fe})$
  $t_{start} \leftarrow l_{e,e_{next}} * (t * \tau_e + pipeline(e, e_{next}))$
  $c \leftarrow C \cup [(e, t)]$
**end for**
**if** $|c| <> |E_f|$ *or* $cost(c).delay > \delta_f$ **then**
  **return** $< \{\} >$
**end if**
  **return** $< c >$

**Algorithm 12** Cost function implementation: cost()

**Require:** $c_{fi}$

$delay \leftarrow d_{E_f[-1]} + (c_{fi}[-1].t_{start} - 1) * \tau_{E_f[-1]} - P_f * (i - 1)$
**if** $delay > f.max\_delay$ **then**
  $\Delta_j \leftarrow delay - f.max\_delay$
**else if** $delay < f.min\_delay$ **then**
  $\Delta_j \leftarrow f.min\_delay - delay$
**else**
  $\Delta_j \leftarrow 0$
**end if**
  **return** $< \Delta_j, delay, c[0].t_{start} >$

**Algorithm 13** Candidates comparator implementation : compare()

**Require:** $c_1, c_2$
1: **if** $c_1 == \{\}$ **then**
  **return** $c_2$
2: **end if**
3: $best \leftarrow c_2$
4: **if** $cost(c_1).\Delta_j \leq cost(c_2).\Delta_j$ **then**
5:  $best \leftarrow c_1$
6: **else if** $cost(c_1).delay \leq cost(c_2).delay$ **then**
7:  $best \leftarrow c_1$
8: **else if** $cost(c_1).t_{start} \leq cost(c_2).t_{start}$ **then**
9:  $best \leftarrow c_1$
10: **end if**
  **return** $best$

This function implements also the circular buffer optimization as discussed in the ILP model. In the heuristic solution is not required to solve the problem for two contiguous SFs, if there are no available time slots in the set $[t, t_e]$, then the function searches for the first available in the set $[1, t-1]$.

**Algorithm 14** Find the first pattern matching in bitarray: findFirst()

**Require:** $e \in E, t \in [1, t_e], w_{fe}$
1: $mask \leftarrow bitarray([1 * w_{fe}])$
2: **for** $idx \in [t-1; t_e - 1]$ **do**
3:  **if** $T_e[idx : idx + w_{fe}]$ & $mask$ **then**
  **return** $idx + 1$
4:  **end if**
5:  $mask \leftarrow mask >> 1$
6: **end for**
7: **for** $idx \in [0; t-2]$ **do**
8:  **if** $T_e[idx : idx + w_{fe}]$ & $mask$ **then**
  **return** $idx + 1$
9:  **end if**
10:  $mask \leftarrow mask >> 1$
11: **end for**

### 4.1.2 Jitter optimizer module

The following module has the aim to optimize the jitter of a request $f$. Given a scheduling solution $S$, the module finds an iteration with the minimum delay for the request $f$, removes it and performs again a constructive phase

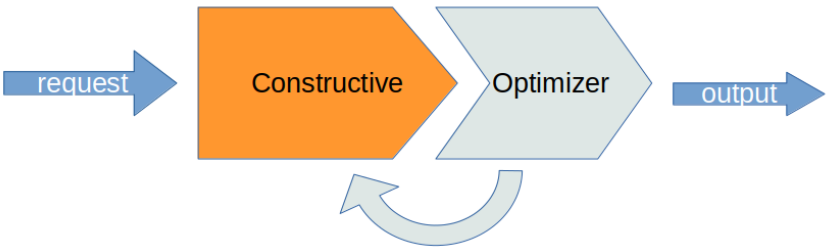



Figure 4.3: Heuristic TAS optimizer feedback scheme

to find the scheduling that better minimizes the jitter of the TS-flow. The optimization process ends when there are no performed optimizations.

Figure 4.3 shows the cycle of the optimization process. The feedback mechanism in the cycle guarantees the end of the process.

**Algorithm 15** Jitter optimization: jitter_optimize()

**Require:** $f, S$

**while** True **do**
  $c_{min} = argmin_{\{c \in S | c.f == f\}} cost(c).delay$
  $S^{'} \leftarrow S \setminus \{c_{min}\}$
  $deallocate(c_{min})$
  $f.min\_delay \leftarrow min_{\{c \in S | c.f == f\}} cost(c).delay$
  $< \{c\}, j, d >= constructive(c_{min}.f, c_{min}.i)$
  $S^{'} \leftarrow S^{'} \cup \{c\}$
  **if** $f.min\_delay < cost(c_{min}).delay$ **then**
    $S \leftarrow S^{'}$
  **else**
    **return** $< S, j, d >$
  **end if**
**end while**

## 4.2 Local search

This section is available for the only problem statement that admits the reconfiguration of the network data plane.

In the scheduling without reconfiguration, when a new request is received if there are no candidates the request is blocked, in the enhanced version the request could be accepted after a reconfiguration of the incumbent data

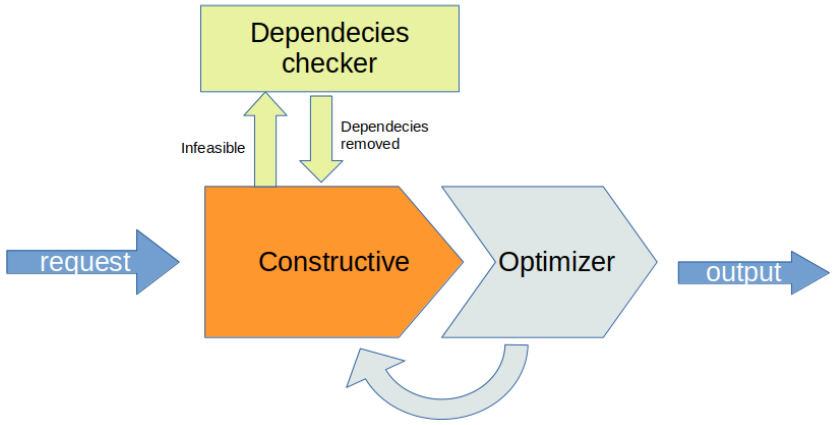



Figure 4.4: Heuristic TAS with data plane reconfiguration

plane. To reconfigure the data plane is required to evaluate the dependencies of the blocked request, remove that, and then run a new constructive phase with the removed dependencies and the blocked request sorted by the level of restriction of the metric constraints. A dependency is an iteration $i$ of a scheduled request $f$ that collides with the hypothetical scheduling of $r$. The LS algorithm 16 describes the process. Figure 4.4 shows how LS with data plane reconfiguration works. A new request is processed by a constructive phase, if it is not feasible, then the dependencies checker removes dependencies and it runs again constructive phase.

## 4.3 Optimization objectives and solving complexity

The function $cost(.)$ defines the optimization objectives. For each flow, at the first iteration, the delay is minimized, and then the delay is balanced to reduce the cost in terms of jitter. Thus, as for the ILP model, the main priority is given to the maximization of network determinism. Table 4.2 evaluates with the framework defined in 2.1 the quality of the heuristic algorithm.

Each module of the heuristic model has a different solving complexity, however, the main contribution to the solving time is given by the constructive phase. For each flow and each iteration are generated $\Delta_{t_{start}}$ candidates according to the $t_{start}$ problem. The number of iterations performed by the *candidate* function is equal to $|E_f|$ and for the $findFirst$ function is at most $t_e$ access in memory.

To summarize the solving complexity is $O(|F| * T_f * \Delta_{t_{start}} * |E_f| * t_e)$ Assuming to schedule the first iteration of a request under the following

**Algorithm 16** LS network reconfiguration: reconfigure()

**Require:** $r, S$

$R \leftarrow \{\}$
$S` \leftarrow S$
**for** $i \in [1, T_r]$ **do**
    $R \leftarrow R \cup \{(r, i)\}$
    $lt_{start} \leftarrow \frac{P_f * (i-1)}{\tau_{E_f[0]}+1}$
    $ut2_{start} \leftarrow lt_{start} + \delta_f - latency(f)$
    $ut3_{start} \leftarrow t_{E_f[0]} - w_{fE_f[0]} * (T_f - 2)$
    $e_{src} \leftarrow E_f[0]$
    **for** $t_{start} \in [lt_{start}, min(ut3_{start}, ut2_{start})]$ **do**
        **if** $T_{e_{src}}[t_{start} - 1] == 0$ **then**
            $c \leftarrow findAssignment(S, e_{src}, t_{start})$
            $S` \leftarrow S` \setminus \{c\}$
            $deallocate(c.f, c.i)$
            $R \leftarrow R \cup \{(c.f, c.i)\}$
        **end if**
    **end for**
**end for**
$R \leftarrow sort(R, f : \frac{\delta_f}{|E_f|}, ASC)$
$S``, j, d = constructive(R.flows, R.iterations)$
**if** $S``$ == INFEASIBLE **then** // The request is blocked
    **return return** $< S, j, d >$
**end if**
$S, j, d = merge(S``, S`)$
    **return** $< S, j, d >$

worst-case assumptions:

- $|F| \leq |E|$: at each time slot only one TS-flow can be assigned to one network interface.
- $T_f \leq min(t_e)$. At most the duration of the period $P_f \leq min_{e \in E_f}(\tau_e)$.
- The maximum admitted delay $\delta_f \geq T$
- The scheduling window size $w_{fe} = 1$
- $|E_f| \leq |E|$. Suppose that at least one flow has a path with all the interfaces of the network.
- The $max(t_e) \leq T$. The number of time units of SF is the maximum allowed number of time slots.
- $\Delta_{t_{start}} \leq t_e - T_f$. According to the maximization of the $\Delta_{t_{start}}$.

The complexity of the heuristic algorithm is $O(|E|^2 * t_e^2 * (t_e - 1)) \approx O(|E|^2 * T^3)$.

The complexity of the heuristic algorithm depends on the topology of the network and the granularity of the scheduling. Comparing this definition and the ILP model with the data plane reconfiguration's size, the heuristic one is not dependent on the maximum network throughput. The $findFirst$ function returns the first available pattern with $w_{fe}$ available time slots. To conclude, the ILP solving complexity is np-complete, and the heuristic algorithm is polynomial.

| **Objective id** | **Scored (Y/N/-)** |
|---|---|
| **Primary optimization objectives** | |
| 3.1 Maximize the determinism | Y (Primary optimization) |
| 3.2 Improve the transmission metrics | Y (the first available time slot is taken by the candidate function, thus the latency is bounded). |
| **Secondary optimization objectives** | |
| 2.1 Maximize the load balancing | - (The input list $E_f$ contains the path of the flow $f$) |
| 2.2 Minimize the length of the path | N |
| 2.3 Minimize the fragmentation | N |
| 3.3 Maximize security | Y (not explicitly, including the required additional guards' time slots in the set $G_e$ per each network interface) |
| 3.4 Maximize robustness | Y (Not explicitly, including the required additional guards' time slots in the set $G_e$ per each network interface) |
| **Thirdly objectives** | |
| 4.1 Minimize data plane update cost | Y (the dependencies checker module limits the dependencies definition to the only source network interface) |
| 4.2 Minimize solving time | Y |
| **Non optimization objectives** | |
| 1.1 Minimize topology cost | N |
| 1.2 Multicasting | N |

Table 4.2: Heuristic model with jitter optimizer and dependencies checker modules, objectives framework evaluation

# Chapter 5

# Verification, validation and benchmarks

A model is built on its underlying hypothesis, and ensuring its correctness and proper application is essential for its accreditation. They represent how the author understands the system, conceptualizes the model, and develops the coding. In the work, the hypothesis and assumptions are declared, in particular, there are structural and simplification hypotheses. Structural hypotheses in ILP are validated by the solver layer, thus by [9] and [7]. Simplification hypotheses are not validated. This is a starting point for future work: demonstrating that stochastic components of the E2E delay have a variance of at most one time slot of the last network interface of the path of the TS-flow[1]. About the data, there are no hypotheses, but mathematical restrictions in 2.5.

According to [5] validation means ensuring the model is correct, while verification means ensuring the model has been correctly coded. Accreditation means that the stakeholders believe in the model and will use it to implement or modify (according to the model results) the system, however, this phase is out of the scope of this work.

The life cycle of a simulation project is based on the iterative process of interleaving verification, validation and accreditation sub-processes phases. The failure of one of them requires a backtrack to the previous steps of the project. Figure 5.1 form [5] defines the Verification, validation and accreditation process (VV&A) phases. There are eight of these (in the figure represented by squares), linked with the references in this work:

[1]Performing this requires the knowledge of specifications of different IIoT-devices. First, it needs to measure the signal's period of a TS-flow lots of time, then evaluate its probability distribution. At the end compare the variance of the obtained distribution with the duration of a time slot in the last network interface of the path of the flow. Iterate this process with different IIoT devices and extract the features to collect the different produced signals. E.g. with a clustering model it is possible to predict for generic time-sensitive flow its period variance, according to the extracted features.

1. Goals, since the clarification of the relevant elements that rule the organization are a product by itself. In the developed model the goals are defined in table 2.1.

2. Problem Entity, which is the definition of the problem analyzed. For the model of this work, the problem statements are defined in table 2.6, 2.5.

3. Conceptual Model, which defines a complete and unambiguous representation of the model. The work defines different conceptual models, two in the ILP section and different combinations in the heuristic section.

4. Scenarios/Configurations to be analyzed, described in the Validation section.

5. Computerized Model, that is the simulator that codes the model. In the validation section the real-case scenario example. In the future work section, there is a reference to discrete network simulators as OMT++/ns-3 which could be a good means to improve this phase of the process.

6. Solutions, verified with unit and integration tests in the Verification section.

7. Accepted Solutions, accepted because of a model-based discussion. At writing time this process is running, thus demanding a future work report.

8. System itself since it will be modified based on the Accepted Solutions.

## 5.1 Verification

The function 17 verifies the input of the TAS scheduling model and the function 18 verifies the validity of a produced solution. Figure 5.2 shows in a high-level flow chart the point at which each verification function is tested.

### 5.1.1 Unit tests

With unit tests, each component of the scheduling algorithm is tested autonomously. Withe-box testing is the chosen strategy, each possible scenario and code branch could be tested. Table 5.1 reports the tests performed with covered scenarios.

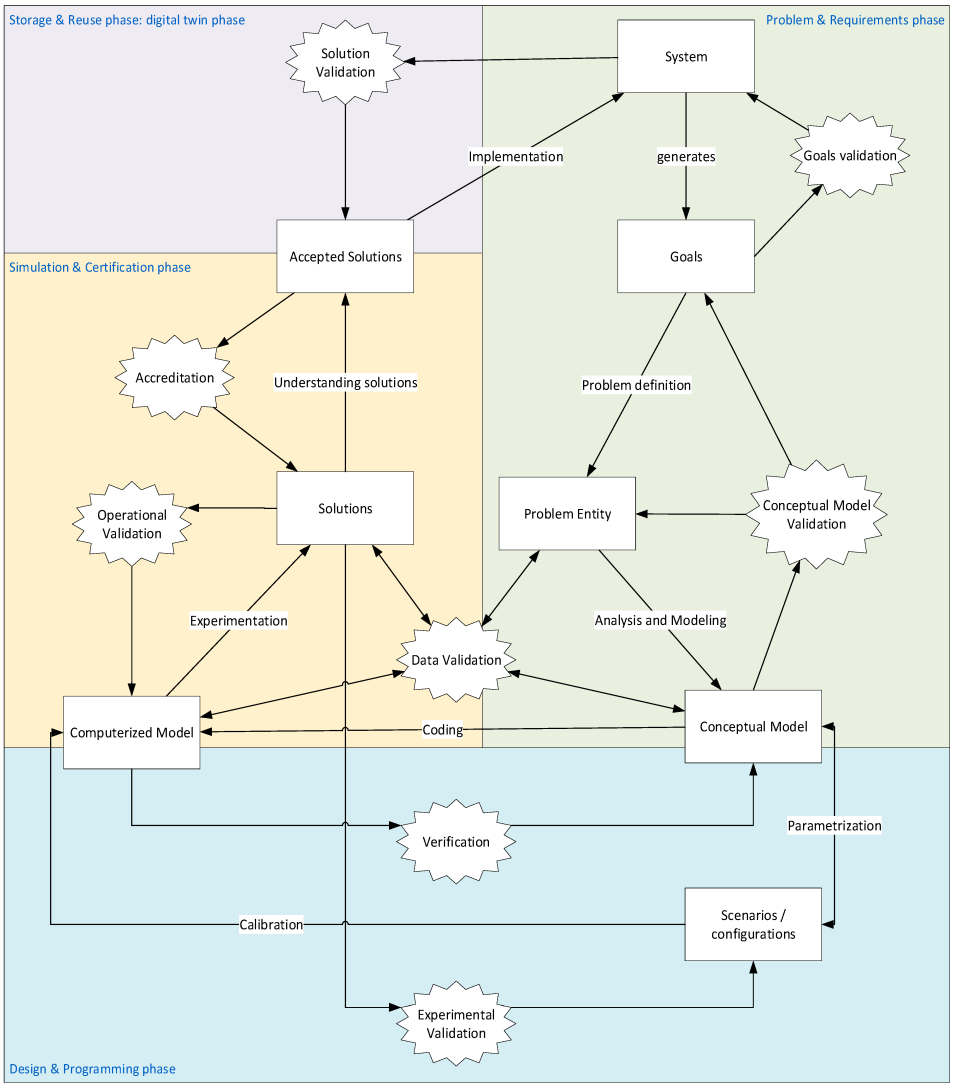



Figure 5.1: Verification, validation and accreditation process for a simulation model from Fonseca i Casas, P. A Continuous Process for Validation, Verification, and Accreditation of Simulation Models. Mathematics 2023, 11, 845.

**Algorithm 17** Input verification : apriori_feasible()

**Require:** $F, E$

1: **for** $e_1, e_2 \in E$ **do**
2: **if** $MCD(a_{e_1}, a_{e_2}) \leq 1$ **then return** $< false >$ // violated math restriction
3: **end if**
4: **if** $MCD(\tau_e, T) \leq 1$ **then return** $< false >$ // violated math restriction
5: **end if**
6: $r_{e_1} \leftarrow |T_e|$ // Available time slots after pre-processing
7: **end for**
8: **for** $f \in F$ **do**
9: **if** $MCD(P_f, T) \leq 1$ **then return** $< false >$ // violated math restriction
10: **end if**
11: **for** $e \in E_f$ **do**
12: $r_e \leftarrow r_e - w_{fe}$
13: **if** $r_e \leq 0$ **then return** $< false >$ // no enough resources
14: **end if**
15: **end for**
16: **end for**
**return** $< true >$

**Algorithm 18** Output verification : valid()

**Require:** $F, E$

$S \leftarrow \{0\}$ // index f
**for** $e \in E$ **do**
    $test \leftarrow T_e$ // Available time slots after pre-processing
    **for** $(f, t) \in SF_e$ **do**
        $S_f \leftarrow S_f + 1$
        **if** $S_f > T_f$ **then return** $< false >$ // flow scheduled twice (or more)
        **end if**
        **for** $j \in [t - 1, t + w_{fe} - 2]$ **do**
            **if** $test[j] == 0$ **then return** $< false >$ // overlapping in resources
            **else**
                $test[j] \leftarrow 0$
            **end if**
        **end for**
    **end for**
**end for**
**for** $f \in F$ **do**
    **if** $S_f \leq T_f$ **then return** $< false >$ // flow not scheduled
    **end if**
    **for** $e1, e2 \in E_f$ **do**
        **for** $i \in [1, T_f]$ **do**
            $t_1 \leftarrow \tau_{e_1} * (SF_{e_1}[f, i] - 1)$
            $t_2 \leftarrow \tau_{e_2} * (SF_{e_2}[f, i] - 1)$
            **if** $t_2 < t_1 + pipeline(e_1, e_2)$ **then return** $< false >$
            **end if**
        **end for**
    **end for**
**end for**
    **return** $< true >$

| Test ID | Coverage | apriori_feasible() | valid() |
|---|---|---|---|
| **Scheduling feasibility**: Empty network with 3 links with time slot duration $\tau$ and $a$ bit processed. | | | |
| U1 | Given a request $f$, the e2e delay of f is constant the jitter is 0 and each iteration $i$ of $f$ precedes the iteration $i+1$ in all the directed links of $E_f$ | true | true |
| U2 | Given two requests $f_1, f_2$, where $E_{f_1} = E_{f_2}$ | true | true |
| U3 | Given two requests $f_1, f_2$, where $E_{f_2} != E_{f_1}$ | true | true |
| U4 | Given two requests $f_1, f_2$, where $E_{f_2} != E_{f_1}$ for the order of the link | true | true |
| U5 | Given a request $f$ where $T_f > t_e$. (not enough time slots) | false | - |
| U5 | Given a request $f$ where $P_f = 3 * T$. (The period of the flow f is higher than T) | false | - |
| U6 | Given a request $f$ where $\delta_f = 1t.u.$ (The flow is not scheduled, too restrictive admitted delay) | true | false |
| U7 | Given a request $f$ where $P_f = 1t.u.$ (The flow is not scheduled, too many iterations) | true | false |
| U7 | Given a request $f$ where $E_f = []$. Expected e2e delay and jitter 0. | true | true |
| **Heterogeneous scenario**: Empty network with 2 links with different time slot duration and processed bits. | | | |
| U9 | Given two requests $f_1, f_2$ | true | true |
| U9.1 | Given two requests $f_1, f_2$ where $a_{e_1} = 3, a_{e_2} = 4, \tau_{e_1} = 3, \tau_{e_2} = 4$ | true | true |
| U9.2 | Given two requests $f_1, f_2$ where $a_{e_1} = 4, a_{e_2} = 3, \tau_{e_1} = 3, \tau_{e_2} = 4$ | true | true |
| U9.3 | Given two requests $f_1, f_2$ where $a_{e_1} = 3, a_{e_2} = 3, \tau_{e_1} = 3, \tau_{e_2} = 4$ | true | true |
| U9.4 | Given two requests $f_1, f_2$ where $a_{e_1} = 3, a_{e_2} = 4, \tau_{e_1} = 4, \tau_{e_2} = 3$ | true | true |
| U9.5 | Given two requests $f_1, f_2$ where $a_{e_1} = 4, a_{e_2} = 3, \tau_{e_1} = 4, \tau_{e_2} = 3$ | true | true |
| U9.6 | Given two requests $f_1, f_2$ where $a_{e_1} = 3, a_{e_2} = 3 \tau_{e_1} = 4, \tau_{e_2} = 3$ | true | true |
| U9.7 | Given two requests $f_1, f_2$ where $a_{e_1} = 3, a_{e_2} = 3, \tau_{e_1} = 4, \tau_{e_2} = 4$ | true | true |
| **Circula buffer optimization** | | | |
| U10 | Given two requests $f_1, f_2$ where the last iteration of $f_1$ ends in the second SF. Check the correctness of the e2e estimated delay. | true | true |
| U11 | Given two requests $f_1, f_2$ where the $P_{f_1} = T - 1$ | false | - |
| **Space reduction matrix N** | | | |
| U12 | Given two requests $f_1, f_2$ | true | true |
| U13 | Given two requests $f_1, f_2$ where $E_{f_2}[-1]$ is full | true | false |
| **Jitter optimizer module** for heuristic in empty network of section 1. | | | |
| U14 | Given a request $f$ where $T_f = 5, T = 5 * P_f$ e2e jitter expected 0. | true | true |
| U14.1 | Given a request $f$ where $T_f = 10, T = 5 * P_f$ e2e jitter expected 0. | true | false |
| U14.2 | Given a request $f$ where $T_f = 1, T = 5 * P_f$ e2e jitter expected 0. | true | true |
| **LS data plane reconfiguration module** for heuristic in 90% load network. | | | |
| U15 | Given a request $f$ where $T_f = 5, T = P_f$ | true | true |
| **ILP data plane reconfiguration** for ILP with model reconfiguration. | | | |
| U16 | Given a request $f$ and empty network. Expected 0 changes | true | true |
| U17 | Given a request $f$ and 95% load network. Expected reconfiguration. | true | ?(true) |

Table 5.1: Unit tests

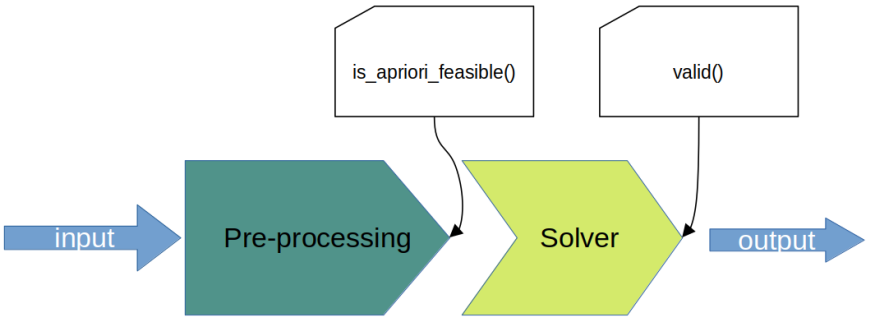



Figure 5.2: Verification high-level flow-chart

### 5.1.2 Integration tests

Integration tests aim to detect interface errors, inconsistencies, and interaction issues between modules that have already been unit-tested. Integration tests examine the behaviour of multiple components when they are combined, each of them treated as a black box. Given the smallest instance with 50 requests described in the Benchmark section, at the start empty, the following configurations are tested:

1. ILP scheduling with data plane reconfiguration + circular shifting optimization

2. ILP scheduling without data plane reconfiguration + circular shifting optimization

3. Heuristic + jitter LS optimization + LS data plane reconfiguration

When a request is accepted the state of the network is stored and the next request is processed. The statements that accept data plane reconfiguration can perform changes at the already accepted request allocations. The expected output data plane can differ for different configurations however the results of the functions *apriori_feasible()* and *valid()* have to be true in all configurations.

## 5.2 Validation

In this section the validation of the model with the simulation of the TAS scheduling algorithm in a heterogeneous network scenario with different TS-flows and a transport network between two user equipment. The simulations run over a VM with 4 vCore and 32GB of RAM.

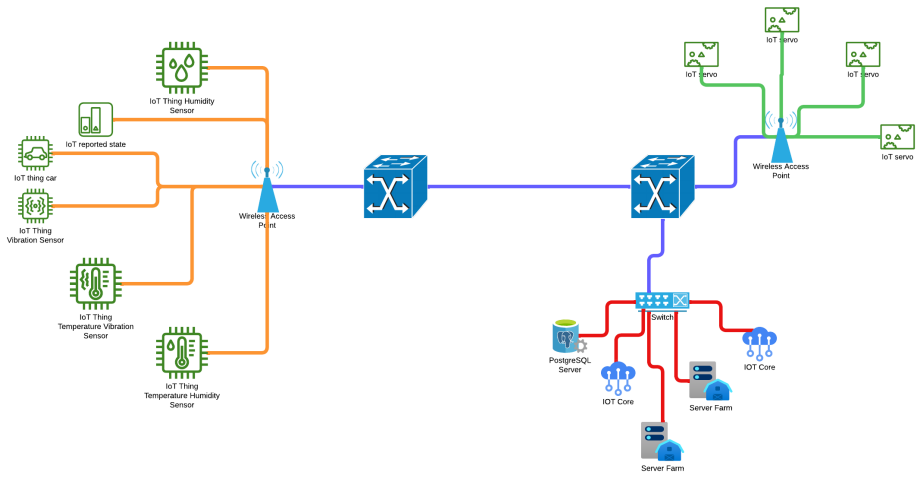



Figure 5.3: Network simulation model. In orange are the radio interfaces that produce TS traffic as IoT sensors connected to an access point. In blu are the interfaces of the provider transport network optical wired. In red are the destination DC interfaces for real-time processing. In green the destination interfaces reached with a radio connection as IoT servos which perform undefined functions.

### 5.2.1 Simulation network configuration

The duration of SF is 10ms. The transport network is optical-wired and the mean distance between different hops is approximated to 20 km. The mean propagation delay is 100us for each interface. The time slot duration per each optical interface is 0.8 ns and its capacity in bit processed is 1Byte, thus the theoretical throughput is 10Gbits. Each user equipment, which is the source of a TS-flow is connected to an access point with a WiFi6 interface with Modulation and Coding Scheme (MCS)-5[2]. The duration of the time slot is 0.5us and the capacity in bits processed is 3 Byte, thus with a throughput of 48Mbits, the propagation delay is 4us. The last network interface could be optical-wired or WiFi; for optical-wired, the propagation delay is evaluated over 100m, thus equal to 500ns.

Figure 5.3 shows an example of a network model for simulation purposes.

### 5.2.2 Simulation statement

To evaluate different performances for different destination interfaces (optical wired vs WiFi6) and different processing modes (express vs store and

[2](MCS)-5 defined by 64-QAM modulation and a coding ratio of $\frac{2}{3}$. To represent 64 symbols is required to transmit 6 bits but only 4 of them transmit data according to the coding ratio.

forward), four different simulation configurations are defined in table 5.3. The TS-traffic is randomly generated between two different classes of application, which are described in table 5.2. The label *wifi2wifi* means a TS flow generated from an IIoT interface to a WiFi IIoT interface, while the label *wifi2wired* means a flow from a WiFi IIoT interface to the data center, thus the destination interface is optical-wired. The simulation tries to allocate 3000 requests over the TSN described heterogeneous network, for each request if feasible, thus allocated, the new network status is installed. The number of the source and destination user equipments is 200.

| **Application 1** | |
|---|---|
| $P_f$ : Period of the flow | 1ms |
| $\delta_f$ : Maximum admitted delay | 1ms |
| $\upsilon_f$ : Maximum admitted jitter | 100 $\mu$s |
| $\#f$ : Bit transmitted by each iteration | random $[90, 120]$ bytes |
| $E_f$ : Path | randomMST(src,dest) |
| **Application 2** | |
| $P_f$ : Period of the flow | 10ms |
| $\delta_f$ : Maximum admitted delay | 10ms |
| $\upsilon_f$ : Maximum admitted jitter | 1ms |
| $\#f$ : Bit transmitted by each iteration | random $[900, 1200]$ bytes |
| $E_f$ : Path | randomMST(src,dest) |

Table 5.2: Simulation applications

### 5.2.3 Simulation results

Given the size of the model, the TAS scheduler is implemented with the heuristic model with the jitter optimization module. Table 5.3 reports the feasibility of each simulation configuration, in other words, the percentage of accepted requests over 3000. For each simulation configuration are reported the following graphs:

- **Solving and pre-processing time**. The solving time is interpolated with a cubic polynomial, according to the estimated solving complexity of T in 4.3.

- **KPI extracted for application**. Estimated jitter and delay for each request. Maximum and mean over a dynamic window from the first request so far. Mean of estimated delay over a mobile window of 100 values.

- **Cumulative network performance**. Network performance cumulative of Application 1 and 2 traffic increasing the loading.

| Path \ Processing mode | express | storeAndForward |
|---|---|---|
| wifi2wifi | C1: 71.1% | C2: 69.86% |
| wifi2wired | C3: 80% | C4: 77.43% |

Table 5.3: Simulation configurations with feasibility percentage

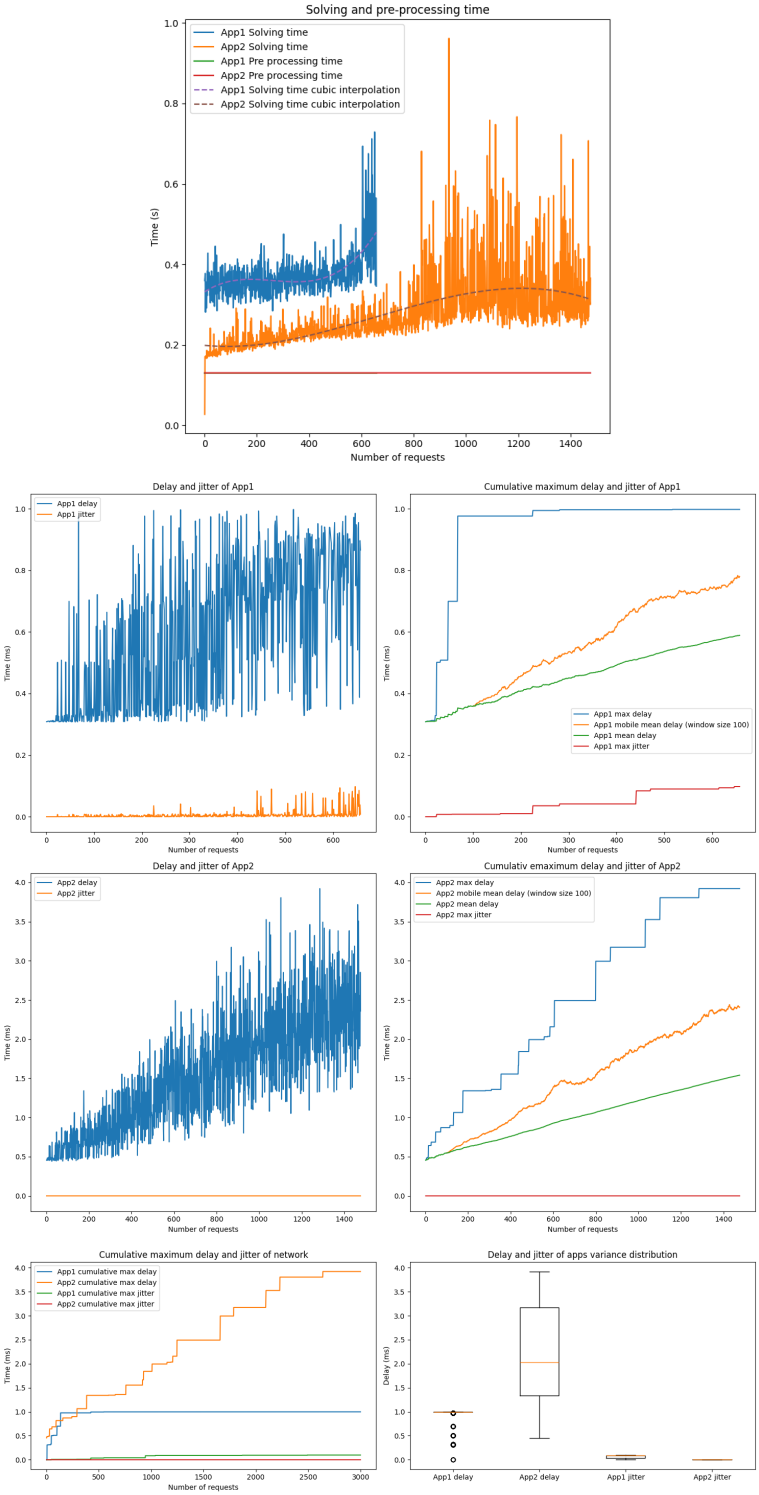



Figure 5.4: C1 simulation results

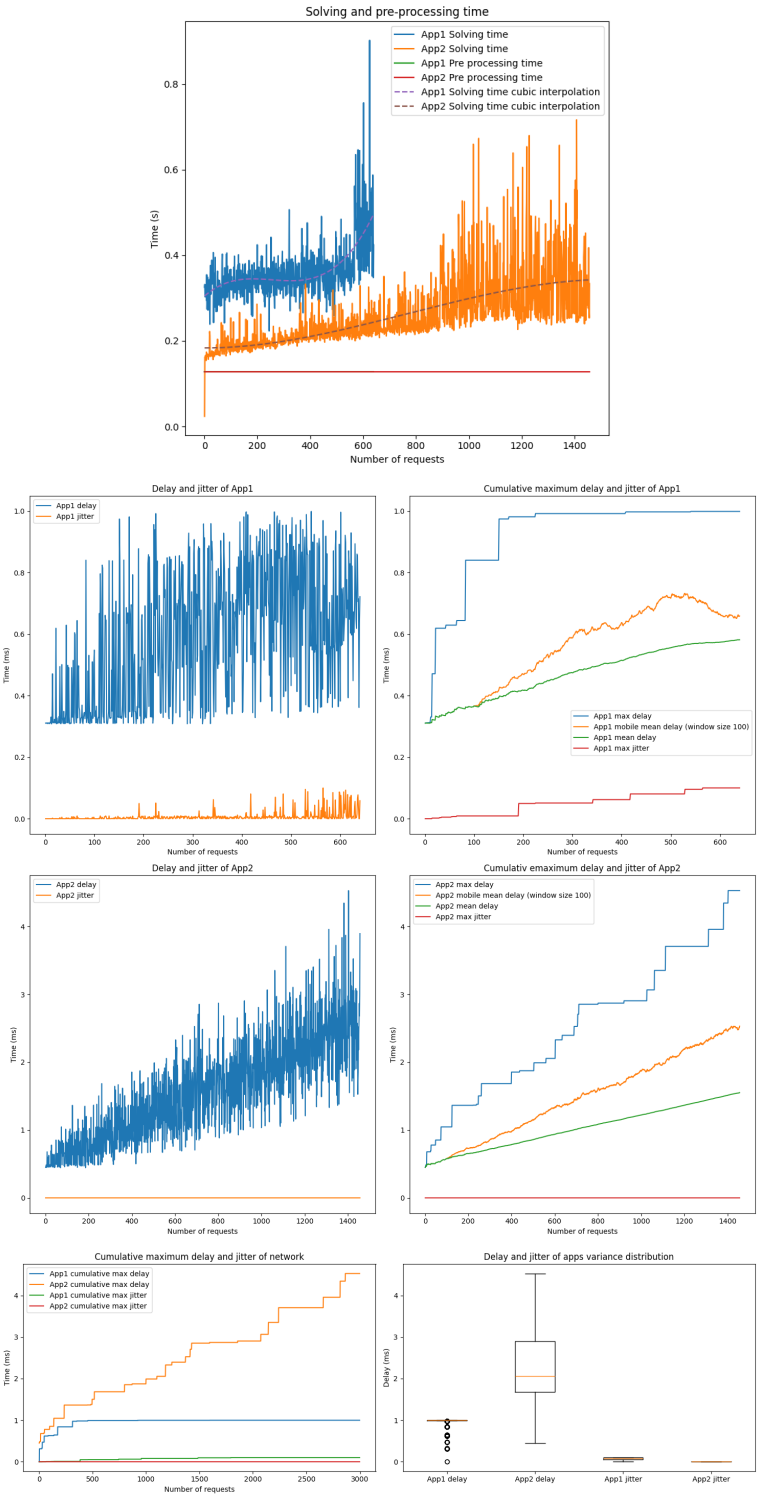



Figure 5.5: C2 simulation results

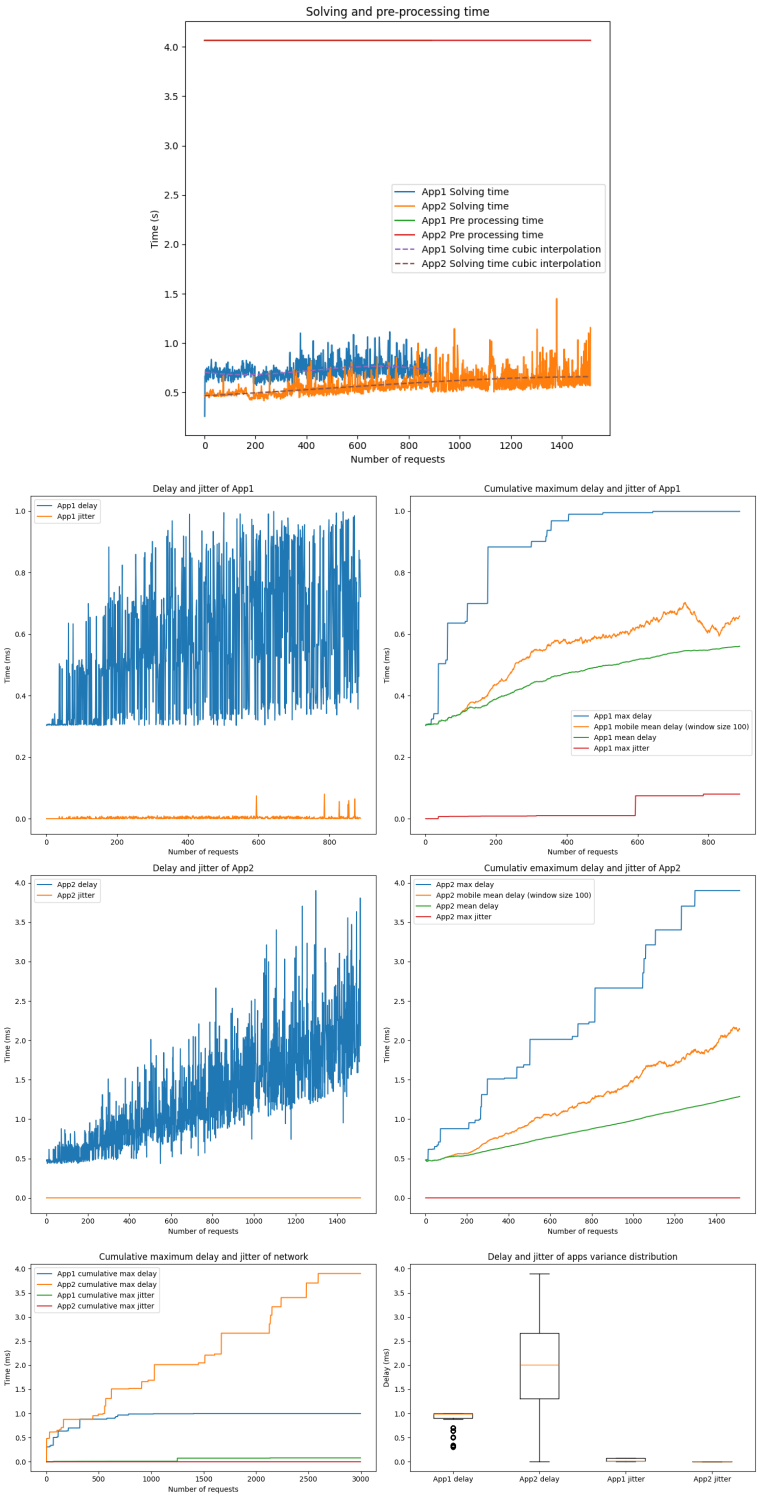



Figure 5.6: C3 simulation results

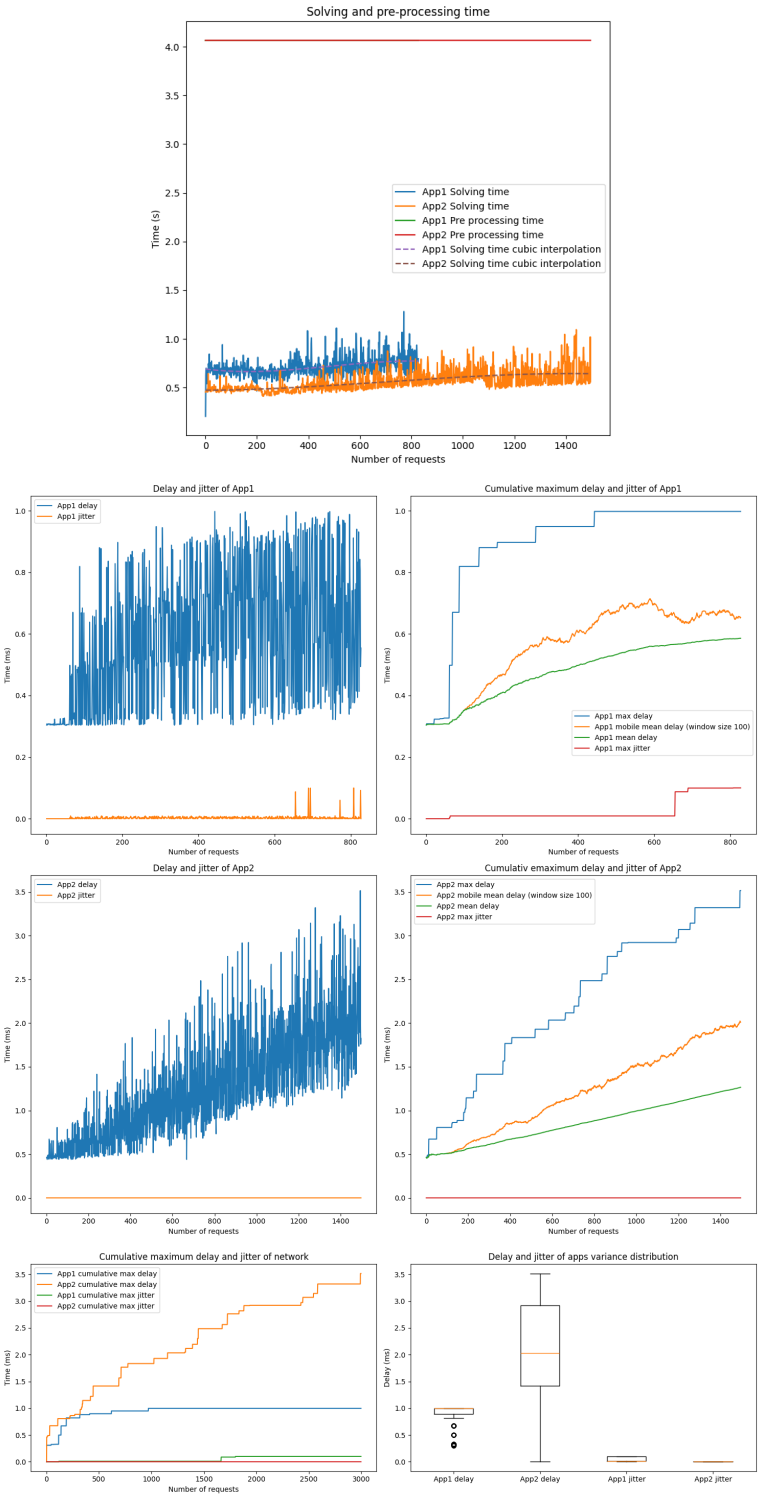



Figure 5.7: C4 simulation results

**Observations**

Observing the graphs 5.4, 5.5, 5.6, 5.7 the following claims are formulated:

1. Each request is independent of the others of the same application class. The E2E estimated jitter of each request of the Application 2 class is 0 ms because for each cycle there is only one iteration.

2. For all the configurations, about 90% of the requests of class Application 2 are satisfied; however, the percentage of the feasibility of Application 1 traffic depends on different configurations (C1 and C2 about 60%, C3 and C4 about 70%).

3. The solving time follows a cubic polynomial interpolation. In particular for Application 1, which is composed of 10 iterations, is between 300ms and 500ms and for Application 2 which is composed of only 1 iteration is between 180ms and 300ms.

4. The pre-processing time is constant and depends on the network configuration. It is about 150ms for C1 and C2 and about 4s for C3 and C4. The substantial difference is given by the memory operation and the pre-computation of transformation matrices.

5. The maximum cumulative delay and jitter of the system are step functions that increase with increasing loading of the TS-network.

6. The boxplot graphs confirm that the objective function is reached, the jitter has a minimum variance for both applications and the delay has a higher variance to balance it for different iterations of the same request.

7. Variation of the slope of the delay mobile mean over a window of 100 values is justified by the increasing of the maximum allowed jitter for Application 1.

8. Application 1 reaches the saturation point [3] after 200 requests in C1 and C2, when 600 requests are accommodated in C3, and after 500 requests in C4. The jitter for different requests of this class is between 0us and 100us.

9. For Application 2 the maximum reached delay is 4ms if the destination is wired, and 5ms if the last is a WiFi interface. The saturation point is not reached.

[3]The point in which the maximum allowed delay is reached by a request of the application class.

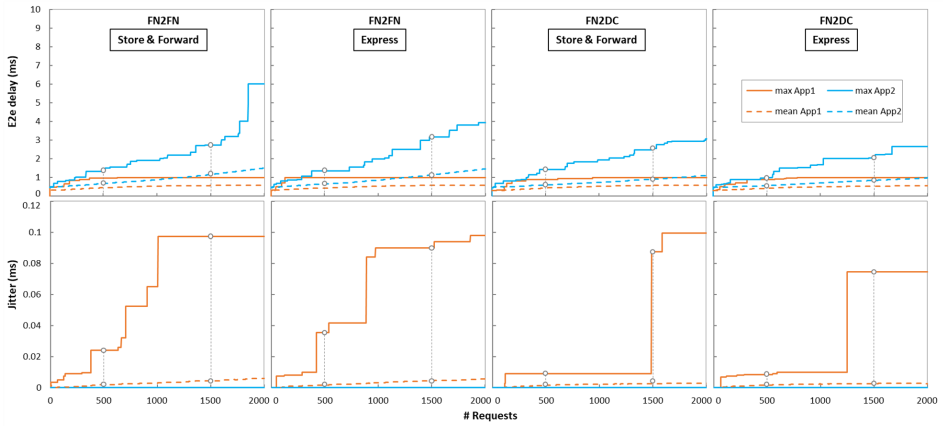



Figure 5.8: Metrics comparison for different configurations.

Figure 5.8 compares E2E estimated metrics for different configurations. The label *FN2FN* is mapped over *wifi2wifi* and *FN2DC* is mapped over *wifi2dc*. Focusing over 1500 requests accommodated:

1. Overall, the estimated jitter and delay have better performance with express processing mode.

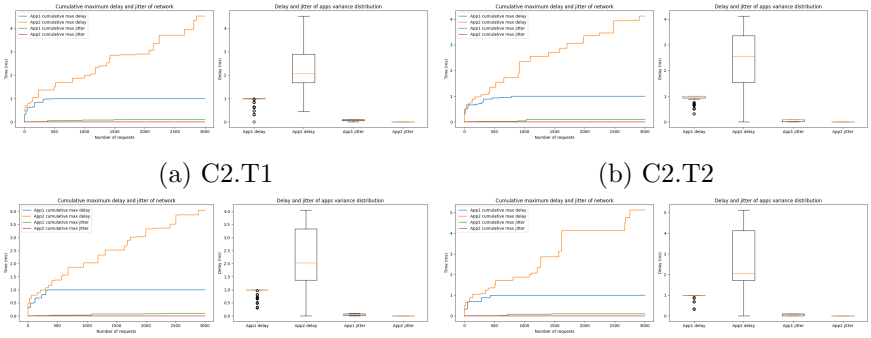
2. The main contribution to the delay estimation is given by the propagation delay. This explains why, when comparing scenarios in *wifi2dc*, the improvements in delay estimation for the express mode are not significant. However, when comparing the maximum jitter, Application 1 in express processing mode shows a lower jitter of 30us.

3. The metrics for *wifi2dc* are better than *wifi2wifi*. This is explained by different factors:

    - Throughput of the optical wired is higher than WiFi6, thus its transmission granularity is better.
    - The propagation delay for optical-wired is lower than WiFi6 (4us for WiFi6 and 0.5us for optical connections in data center).

### 5.2.4 Throughput

In this section throughput analysis given the previous network configuration C2, labelled with *wifi2wifi* and *storeAndForward*, varying the number of bits processed by interfaces, mixing 3000 TS-requests randomly generated by Application 1 and Application 2. Table 5.4 reports the simulated configurations and figure 5.9 illustrates the cumulative metrics for each configuration.



(a) C2.T1 (b) C2.T2

(c) C2.T3 (d) C2.T4

Figure 5.9: Cumulative metrics for throughput simulations.

| Optical Processing [Bytes] / WiFi Processing [Bytes] | 1 | 2 | 5 | 10 |
|---|---|---|---|---|
| 3 | C2.T1 | - | - | - |
| 6 | - | C2.T2 | - | - |
| 15 | - | - | C2.T3 | - |
| 30 | - | - | - | C2.T4 |

Table 5.4: Simulation configurations throughput evaluation

### Throughput waste

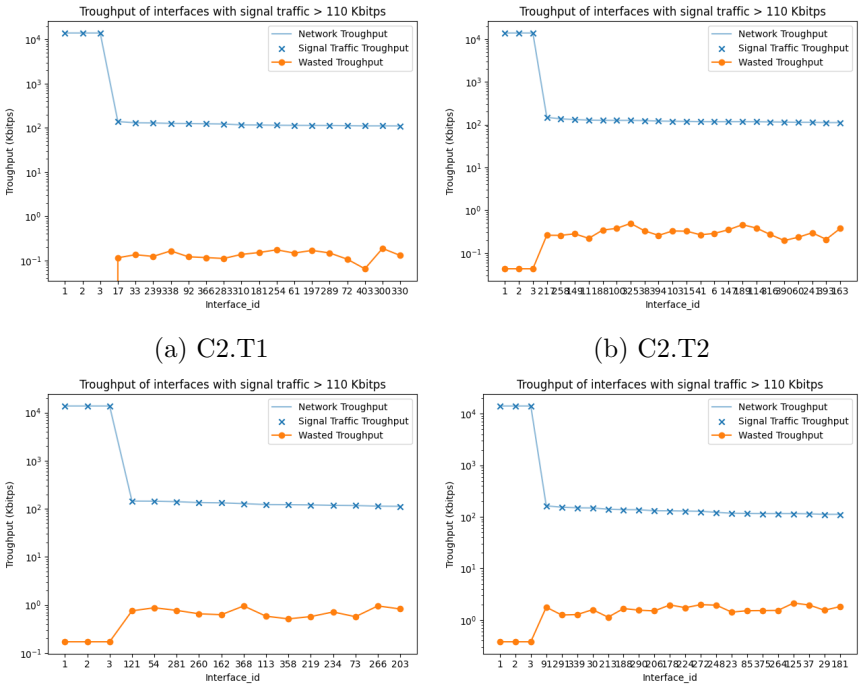
Starting from the definition of the time slot of the interface $e$ as an atomic unit of duration $\tau_e$ in which are proceed $a_e$ bits it is possible to define the wasted throughput of a network interface $e$ as:

$$B_e^w = \frac{\sum_{f \in F} \sum_{t \in t_e} \sum_{i \in [1,T_f]} a_e * x_{feti} - \sum_{\{f \in F | e \in E_f\}} \#f}{T} \tag{5.1}$$

At the numerator, the difference between the amount of allocated processed bits in the network interface $e$ (a.k.a used capacity) and the number of bits transmitted by the TS-flows allocated over $e$, at the denominator the number of time units of the SF. The contribution given by the band of guards is not clear and depends on future work, it may be approximated to 5% of traffic throughput. This analysis doesn't treat bands of guard contribution.

### Simulation results

Figure 5.10 shows the simulated TS traffic throughput (signal traffic throughput), the network used capacity and the wasted throughput as defined. The coexistence of different classes of traffic TS, BE, QoS over the provider's




(a) C2.T1

(b) C2.T2

(c) C2.T3

(d) C2.T4

Figure 5.10: Throughput for the interfaces with allocated more than or equal 110 Kbits of TS-traffic. Interfaces 1,2,3 are transport network switches, others are WiFi6 interfaces. With x, the cumulative signal throughput is allocated over the interface. With the orange line the wasted throughput. The y-scale is logarithmic to summarize the data.

transport network is feasible. In all scenarios the used capacity of the transport network is about 14Mbits, thus the remaining throughput can be allocated to the other traffic classes. The wasted throughput is approximated to 0 over the transport network, and it increases according to the increase in the number of bits processed over each network interface. The wasted traffic in the WiFi interfaces varies in different configurations up to 1% of traffic processed.

Focusing on the wasting of throughput, the experiment is repeated for non-real scenarios described in table 5.5. In this section 100 requests mixed from Application 1 and Application 2 are accommodated. As expected increasing the the number of bits processed increases the wasted throughput.

### 5.2.5 Network loading and scheduling fragmentation

In this section, the network loading is analysed for configurations C2.T1 and C2.T4, which are defined in table 5.4. For each transport network's

| *Transport network* | | | | | | |
|---|---|---|---|---|---|---|
| Id | Path | Optical [Bytes] | WiFi [Bytes] | Network [Kbps] | Signal [Kbps] | **% Wasted** |
| C2.T5 | wifi2wifi | 10 | 30 | 448 | 446 | **0.46** |
| C4.T5 | wifi2dc | 10 | 30 | 482 | 480 | **0.41** |
| C2.T6 | wifi2wifi | 20 | 60 | 500 | 495 | **1** |
| C4.T6 | wifi2dc | 20 | 60 | 499 | 495 | **0.8** |
| C2.T7 | wifi2wifi | 50 | 150 | 479 | 462 | **3.55** |
| C4.T7 | wifi2dc | 50 | 150 | 486 | 467 | **3.9** |
| C2.T8 | wifi2wifi | 100 | 300 | 451 | 412 | **8.65** |
| C4.T8 | wifi2dc | 100 | 300 | 513 | 471 | **8.18** |
| *Lan network* | | | | | | |
| C2.T5 | wifi2wifi | 10 | 30 | 250 | 244.5 | **2.2** |
| C4.T5 | wifi2dc | 10 | 30 | 278 | 274 | **1.44** |
| C2.T6 | wifi2wifi | 20 | 60 | 288 | 279 | **3.1** |
| C4.T6 | wifi2dc | 20 | 60 | 264.5 | 255 | **3.6** |
| C2.T7 | wifi2wifi | 50 | 150 | 301 | 273 | **9.3** |
| C4.T7 | wifi2dc | 50 | 150 | 260 | 250 | **3.84** |
| C2.T8 | wifi2wifi | 100 | 300 | 292 | 324 | **23.3** |
| C4.T8 | wifi2dc | 100 | 300 | 309 | 287 | **7.12** |

Table 5.5: Wasted throughput increasing the bits processed per time slot

interface, the SF is divided into 40 sections of 250us. For each section, the sequences of contiguous available time slots are evaluated, and their lengths are determined. This measurement is repeated throughout the simulation at intervals of 800 requests.

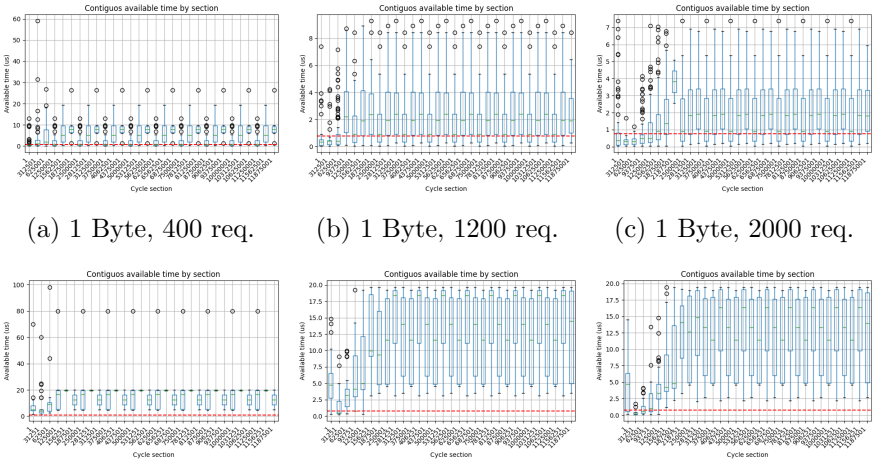
### Simulation results

Data are summarized with the boxplot graphs by section id[4] in figure 5.11 for link 1, 5.12 for link 2, and 5.13 for link 3. This analysis shows the increase in the loading and the scheduling fragmentation. A higher number of requests is equal to a section with a lower number of contiguous time slots available. For processing a 90B TS-request, which is the minimum TS-size in the simulation, a scheduling window of at least 0.768us is required with a throughput of 10 Gbps. The red line marks this lower-bound. If a section has not this time availability the request is accommodated in the next section with a delay cost of 250us. Fragmentation afflicts all the sections with a mean contiguous availability under the red line. Comparing C2.T1 and C2.T10, the fragmentation has a lower negative effect when the size of processed Bytes and the duration of each time slot are higher. The box plots also show the effect of the circular buffer optimization, the load is higher over the starting sections of the SF. Comparing different transport network's interfaces the load is homogeneous and well-balanced, validating the first availability allocation strategy.

## 5.3 Benchmarks

In this section the comparison of performances of two models:

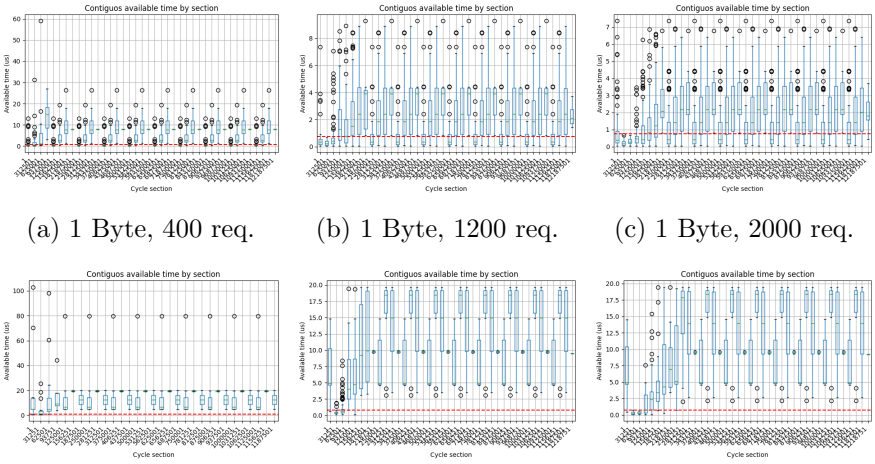
[4]The section id is the id of the first time slot of the section.




(a) 1 Byte, 400 req. (b) 1 Byte, 1200 req. (c) 1 Byte, 2000 req.

(d) 10 Byte, 400 req. (e) 10 Byte, 1200 req. (f) 10 Byte, 2000 req.

Figure 5.11: Loading and scheduling fragmentation transport network interface 1 increasing accommodated requests. In the first row 1 Byte processed per time slot, in the second row 10 Bytes processed per time slot.

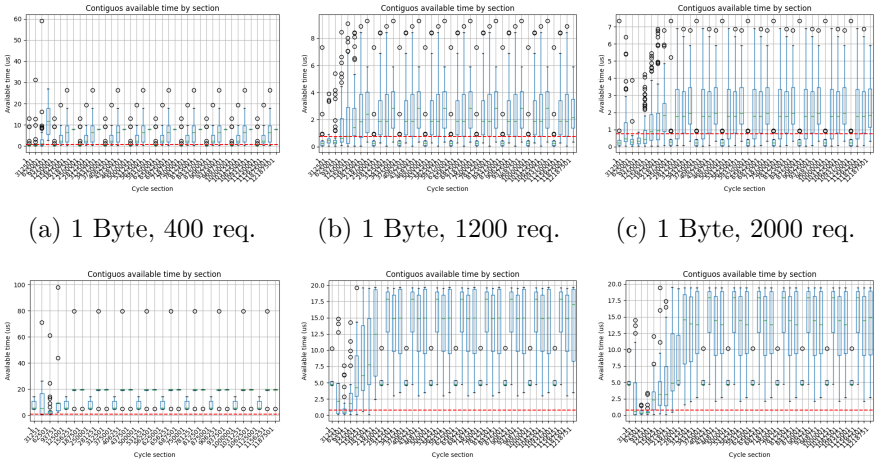



(a) 1 Byte, 400 req. (b) 1 Byte, 1200 req. (c) 1 Byte, 2000 req.

(d) 10 Byte, 400 req. (e) 10 Byte, 1200 req. (f) 10 Byte, 2000 req.

Figure 5.12: Loading and scheduling fragmentation transport network interface 2 increasing accommodated requests. In the first row 1 Byte processed per time slot, in the second row 10 Bytes processed per time slot.

- ILP model without data plane reconfiguration with circular buffer optimization solved with Gurobi [7].
- Heuristic model without data plane reconfiguration with jitter optimization LS.




(a) 1 Byte, 400 req. (b) 1 Byte, 1200 req. (c) 1 Byte, 2000 req.

(d) 10 Byte, 400 req. (e) 10 Byte, 1200 req. (f) 10 Byte, 2000 req.

Figure 5.13: Loading and scheduling fragmentation transport network interface 3 increasing accommodated requests. In the first row 1 Byte processed per time slot, in the second row 10 Bytes processed per time slot.

The experiments are conducted according to fractional factorial Design of experiment (DoE) [5]. According to the model's size, the factors are $T$ which is the granularity of the scheduling and $max(B_e)$ which is the maximum throughput of an interface in the network. The benchmark evaluates the solving time and the optimality gap. For each factor, six levels are evaluated and the benchmark configurations [6] are summarized in table 5.6.The other simulation parameters are fixed: the network topology is the same as figure 5.3, the optical $a_e$ is 1 Byte and the radio $a_e$ is 24 Bytes. The duration of time slots is generated according to the parameter $B_e$. Requests are randomly generated with a path labeled as *wifi2wifi* and $\delta_f = T$, $\upsilon_f = 0.1 * T$ and a period $P_f = T$. The number of bytes for each request is random from 80 to 120. The number of requests generated per instance is 50. As for the validation section, each accepted instance is stored in the data plane and its time slots are not available for the next requests.

Given the number of resources and the time cost required to carry out the ILP experiment, DoE approximates the benchmark of an intermediate configuration $CY$ extracting the contribution of factor $T$ and the contribution of $max(B_e)$ at specific levels.

[5]Experimental design is used to study the effects of multiple factors on a response variable. A full factorial design tests every possible combination of factor levels, instead in a fractional design only a set of experiments is run.

[6]This section aims to evaluate how the factors affect the performance and the quality of the solution in a generic configuration that sure doesn't replicate a real scenario.

| $T$ [ms] \ $max(B_e)$ [Mbps] | 10 | 20 | 30 | 40 | 80 | 100 |
|---|---|---|---|---|---|---|
| 10 | C1 | C2 | C3 | C4 | C5 | C6 |
| 20 | C7 | - | - | - | - | - |
| 30 | C8 | - | - | - | CZ | - |
| 40 | C9 | - | - | - | - | - |
| 80 | C10 | - | - | CX | - | - |
| 100 | C11 | - | - | - | - | CY |

Table 5.6: Fractional design of experiment for benchmark solving time and memory vs optimality gap

### 5.3.1 Execution time

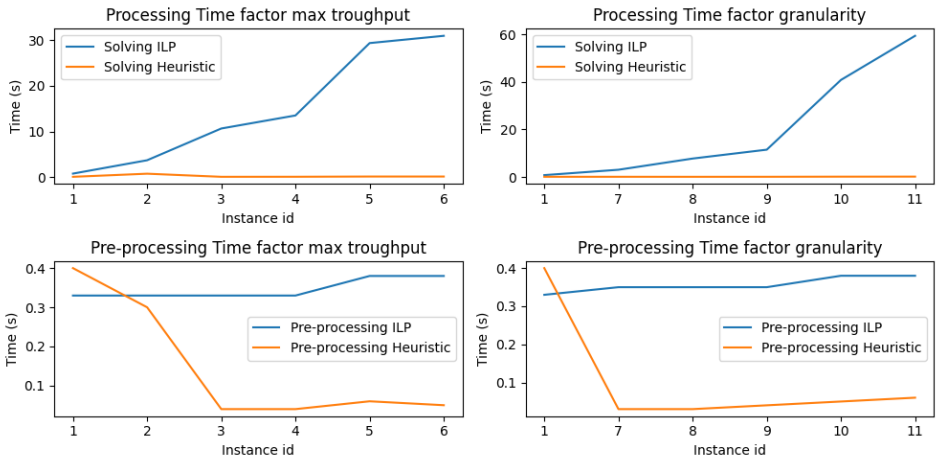



Figure 5.14: Execution time ILP vs Heuristic

Execution time is the amount of pre-processing time and solving time. During the pre-processing phase, the model is instantiated in memory, and optimization structures, such as the circular shifting optimization and the matrix N in ILP, are pre-computed. During the solving time, the solution is built. Figure 5.14 reports the mean solving and pre-processing time for 50 requests of each instance. ILP by definition is np-complete, thus increasing the factor as expected the solving time follows an exponential function. The solving time of the heuristic seems to be constant, however, as demonstrated by the validation section, follows a cubic polynomial with increasing granularity. For the ILP model, the pre-processing time is linear with increasing factors but with a low slope. For the heuristic model, the pre-processing time depends on the network status.

### 5.3.2 Feasibility

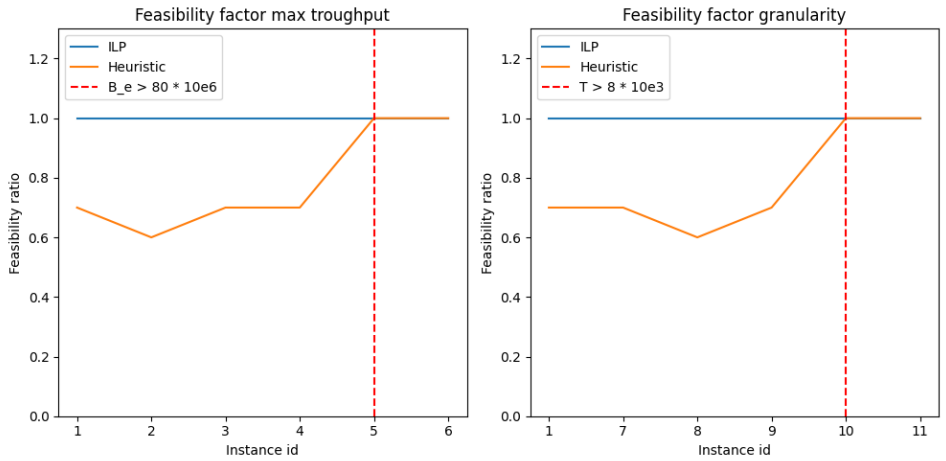



Figure 5.15: Feasibility ILP vs Heuristic

According to the ILP, all generated instances are feasible. Figure 5.15 shows the feasibility ratio for the heuristic, which is the ratio between the number of accepted requests of the instance and the number of requests of each instance (in this case 50). The red lines mean the point at which given the network configuration the feasibility, is comparable [7] between heuristic and ILP. The graph demonstrates that for small instances in which the throughput and the scheduling granularity are low, executing the ILP is the best choice; in big instances with high granularity, given the increasing of required memory by ILP, the heuristic model produces a comparable solution with the global optimal, thus could be the chosen model.

### 5.3.3 Objectives

Figure 5.16 shows how the optimization objectives are achieved by the two different models. All instances have a possible data plane configuration with 0 units of time of estimated jitter. The error of the heuristic model is explained by the Delay opt. gap, which is the difference between the maximum delay for the heuristic and ILP after that the 50th request is evaluated. A negative opt. gap means that the delay has to be higher to reduce the gap over the global optimal jitter. Increasing the granularity and the throughput of the network interfaces, the delay has to increase for two reasons: higher granularity represents the time in a higher number of time units, and increasing the delay is possible to reduce the jitter gap.

[7] In this context comparable means that the feasibility of the Heuristic model has an optimal solution with a gap at most 20%.

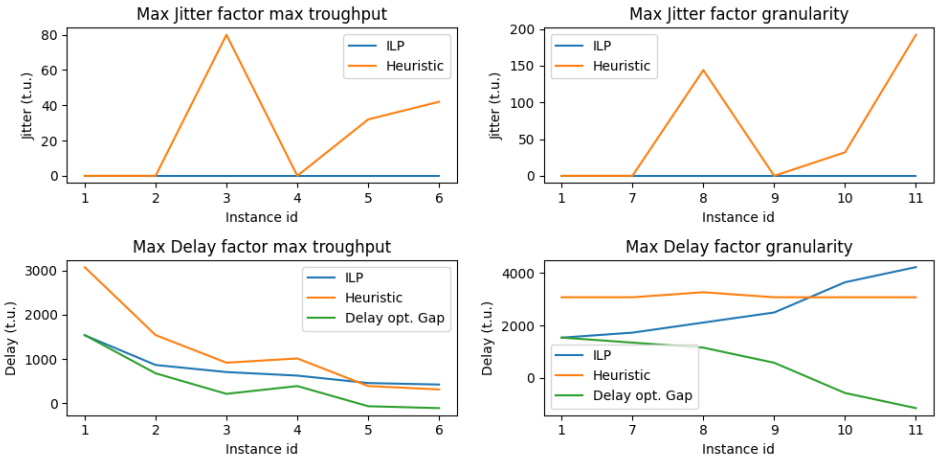



Figure 5.16: Objectives ILP vs Heuristic

This section can be also used to validate the need for a Heuristic jitter optimization module.

# Chapter 6

# Conclusion

Developing the TAS scheduling module for E2E TSN is feasible. According to the benchmark comparison between the two developed models is possible to claim that for small heterogeneous networks with limited throughput, the ILP is the global optimal solution. However, increasing the throughput of the network interfaces, the size of the model explodes and a heuristic approach is required.

If for a simple vehicle or machine network, in which the throughput and propagation delay are limited the ILP guarantees the best performances, in the heterogeneous scenario which involves the transport network of a provider and its integration with IIoT networks the problem is solved by the heuristic model.

Analysing benchmarks in a bigger scenario the quality of the results between ILP and heuristic models are comparable in terms of feasibility, however, an optimal gap is defined as the distance between the local optimal solution and the global one, which is the different level of determinism of the network.

Benchmarks verify that in lots of instances, the price to minimize the estimated E2E jitter of transmission is an increase of the delay. This could be not a real problem, as defined in the objectives declaration, table 2.2, TAS has to guarantee a higher number of accepted TS-requests then to guarantee the maximum determinism of the network as a secondary objective.

Both models have declared errors in metrics estimation at most the duration of a time slot of the destination network interface. They are open points of this work, however, to be pragmatic an error at most of 4us over a signal that has a period of 1ms should be acceptable, and comparable with the same signal period error. To cover these points, a collaboration between producers of network interfaces and IoT devices is necessary. The degree of freedom is adding guard bands in the scheduling as proposed by [10], accepting the increase of throughput waste in the transport network.

This work proposes two different verified scheduling statements, the first

one accepts performing changes over an incumbent data plane, and the second one tries to allocate resources without the reconfiguration of the data plane. The difference between them is, that the first one guarantees higher feasibility, but the practical problem is that reconfiguring allocated flows produces the loss of frames, thus an implicit increase of the jitter. This increase cannot be estimated because depends on the load of the network. In the worst case, the jitter can reach one TSN cycle after one reconfiguration, violating the maximum jitter constraint. Thus each reconfigured flow could be removed for jitter constraint violation, generating an involution of the network status.

The simplest way to proceed is supposing the no reconfiguration statement. This hypothesis seems to be verified by the network throughput graph, which shows that the TS-traffic is limited over the transport network, about 14Mbps with 3000 requests. The throughput, thus the availability of network resources, doesn't seem a problem for the scheduling.

A consistent problem could be the propagation delay of the transport network. This is a physical bound of the transmission mean; however, the pipeline optimization for the express transmission mode can reduce its impact over feasibility. Deleting the effect of the wired propagation delay at the last hop, analysing configurations C1 and C2 of the validation section, the mean mobile delay in a window of 100 values is about the same. The difference between C1 and C2 is the processing mode, for the first one is *express* and for the second one is *store and forward*. The improvements of *express* processing mode over the E2E delay are covered by the high propagation delay, however the estimated jitter for Application 1 is better in C1 and for this reason its feasibility is higher of 2 %. Performing the same comparison the feasibility of C3 (*express* processing mode) over C4 (*store and forward* processing mode) is higher of 3.6%, and the C3 maximum jitter for Application 1 is 30us lower of the maximum jitter of C4. Analysing the configurations C2 and C4, thus excluding the effect of the express processing mode optimization, Application 1 reaches the saturation point [1] in C4 after 400 requests and in C2 after 200 requests. In conclusion, the metrics for wired terminal interfaces are better and the best performance in estimated jitter and delay is reached with the express processing mode optimization.

The period of a request is a crucial point; comparing Application 1 with Application 2, in which the bits transmitted in one SF are about the same because Application 2 has a payload 10 times the payload of Application 1 and Application 1 has a period 10 times the period of Application 2, the performance of the Application 2 are better and it doesn't reach a saturation point. The cost of the real-time is the faster saturation of the network for the specific traffic. In the same duration, for example, a sensor of Application 1

[1] The point in which the estimated e2e maximum delay for a specific application reaches the maximum delay admitted for the application.

has updated its value ten times, and the sensor of Application 2 only once.

## 6.1 Main Contributions

The main contributions of this thesis are:

- In TSN TAS scheduler. This master thesis is in the scope of a project that aims to reach the feasibility of supporting TS-traffic over the providers' transport network and high-performance wireless. The developed TAS scheduling module has to support **heterogeneous network**. In the opposite of the cited papers the developed TAS has to work with interfaces with varying throughputs (from Mbits to Gbits), thus different *inertial frames of reference*. TAS scheduler accepts varying processing modes, store and forward and express. Following a code **pipeline's fashion approach**, the scheduler overlaps the assignation of resources and minimizes the latency. This contribution is fundamental for scheduling TS-traffic over a provider's transport network because mitigates the effect of high propagation delay. This master thesis includes a strategy to accept the **reconfiguration of a TS data plane** maximising the determinism of the network during the installation of a new one. This is achieved by the minimization of the number of changes between the installed and new generated data planes. The TAS admits the possibility of multiple iterations of a flow in SF and implements a **circular shifting scheduling** in SF inspired by the *circular buffer* pattern, to increase the number of accepted requests.

- In Job Shop Scheduling Problem. The developed tools could be extended to a general JSSP problem. The particular contribution of ILP model is for the reconfiguration of a scheduling plan minimizing the number of changes between the new and the incumbent scheduling. Another highlightable contribution is given by the pre-computation of a matrix to prune the space of solution, excluding a-priori non-feasible solutions.

## 6.2 List of Publication

### 6.2.1 Publications in Conferences

- L. Velasco, G. Graziadei, Y. El Kaisi, J. Villares, O. Muñoz, J. Vidal, and M. Ruiz, "Provisioning of Time-Sensitive and non-Time-Sensitive Flows: from Control to Data Plane" accepted in International Workshop on Time-Sensitive and Deterministic Networking (TENSOR), collocated with the IFIP Networking conference, 2024. `https://zenodo.org/records/11393029`

At the writing time, we are targeting to extend this paper to a journal paper.

## 6.3 Future works

While this work has made significant contributions to the field mainly for the heterogeneous network model definition, several avenues for future research could extend the topic. In this section some potential directions for future work.

### 6.3.1 Improvement of the work

The work could be improved by discrete event simulations with OMNET++ environment using the NeSTiNg - Network Simulator for Time-Sensitive Networking (TSN) add-in or ns-3 with a custom TSN add-in. Simulators deliver a real scenario for TSN, thus a good alternative validation method.

To improve the *bitarray pattern search* in the heuristic model it is possible to follow the alternative parallel algorithm proposed in [14] which declares an improvement of $x7$, combined with hardware accelerator and better management of the memory. This section is really interesting because with these two improvements should be possible to solve the scheduling problem for a new request in real-time about the same duration of SF.

Starting from the comparison of application 1 and 2 traffic, defined in the validation section, an additive module can be developed to optimize the scheduling fragmentation. Completing the open point about the estimation of IIoT period error and accepting the scheduling error of at most one time slot of the last network interface of the path. Adding the required guards according to TSN specifications to reach the correct threshold between the wasted throughput and the acceptance of the synchronization error between network and signal.

### 6.3.2 Different approaches

Starting from the definition of the matrix $N$ in the ILP model it is possible to research a dynamic programming algorithm for TAS. Its existence is not sure, e.g. [6] proposes a good and innovative dynamic programming approach, however, as remarked by the paper extension the proof of correctness given it is unfortunately flawed. Another research opportunity is the application of deep reinforcement learning starting from reinforcement learning model as [16], however, this approach could be tricky in the data plane reconfiguration and it is not guaranteed that its accuracy could be better than heuristic iterative algorithms. The developed model statement refers to a cyclic repetitive SF, with DRL could be feasible to develop dy-

namic requests [2] scheduling and guarantee better performances for the other classes of traffic (BE, and QoS).

### 6.3.3 Applications in practice

In this section practical applications of TSN over transport network:

1. **Industrial automation**: given different geographical and functional units and one IIoT control centre, it is feasible centralize the managing and synchronization of them. E.g. in the following scenario there are two productive units with servo robots and one warehouse with autonomous mobile robots. While the servo robots produce something synchronously and with limited latency the warehouse robots can optimize the warehouse space to accept the new input. This could minimize the required warehouse space.

2. **Autonomous Vehicles and Intelligent Transportation Systems**: TSN can facilitate real-time communication between vehicles, road infrastructure, and public transport networks, improving traffic safety and efficiency. E.g. the monitoring of roadway traffic. Each vehicle produces TS-traffic as the speed, the peace, and the path. The vehicle has a radio interface connected to smart mobility access points. The access points are connected to the provider transportation network. The data are processed by the roadway control room for real-time traffic prediction, monitoring speed, and accident notification. The traffic of a vehicle can also sent to other vehicles as the next one or the previous one for the real-time trajectory and the peace update. The real-time updates are the real potentiality of the network as an example for security reasons.

3. **Energy**: Synchronization provided by TSN can enhance the management and control of smart electrical grids, enabling more efficient energy distribution and better integration of renewable sources. E.g. reducing the peaks of energy introduced in the network there is a reduction of wasted energy and also a financial benefit.

4. **Healthcare**: supporting real-time communication for telemedicine and remote surgery applications, ensuring low latency and high reliability for critical medical data transmission.

5. **Financial**: deterministic networking for high-frequency trading platforms, where milliseconds can impact financial transactions significantly.

[2] In different SF a request can transmit the varying amount of bits, or can assume a varying path e.g. according to the network load.

## 6.4 Research Project

### European funded project

This master's thesis is in the scope of NextGeneration UNICO5G Towards a smart and effIcient telecoM Infrastructure meetiNG current and future industry needs (TIMING) (TSI-063000-2021-145).

# Chapter 7

# Appendix

## 7.1 ILP model

### 7.1.1 Constrain definitions

The mathematical demonstration to define the model's constraints is available in this section.

#### Constraint 8

- Row 2

$$\sum_{i=1}^{T_f} x_{feti} = 1 \wedge \sum_{i=1}^{T_f} s_{feti} = 1 \Rightarrow c_{fet} = 0$$

$$\neg(\sum_{i=1}^{T_f} x_{feti} = 1 \wedge \sum_{i=1}^{T_f} s_{feti} = 1) \vee c_{fet} = 0$$

$$\neg(\sum_{i=1}^{T_f} x_{feti} \wedge \sum_{i=1}^{T_f} s_{feti}) \vee \neg c_{fet}$$

Applying De Morgan law

$$(\neg \sum_{i=1}^{T_f} x_{feti} \vee \neg \sum_{i=1}^{T_f} s_{feti}) \vee \neg c_{fet}$$

The logical equation can be rewritten as a linear equation

$$[(1 - \sum_{i=1}^{T_f} x_{feti}) + (1 - \sum_{i=1}^{T_f} s_{feti})] + 1 - c_{fet} \geq 1 \quad (7.1)$$

- Row 3

$$\sum_{i=1}^{T_f} x_{feti} = 0 \wedge \sum_{i=1}^{T_f} s_{feti} = 0 \Rightarrow c_{fet} = 0$$

$$\neg(\sum_{i=1}^{T_f} x_{feti} = 0 \wedge \sum_{i=1}^{T_f} s_{feti} = 0) \vee c_{fet} = 0$$

$$\neg(\neg \sum_{i=1}^{T_f} x_{feti} \wedge \neg \sum_{i=1}^{T_f} s_{feti}) \vee \neg c_{fet}$$

Applying De Morgan law

$$(\sum_{i=1}^{T_f} x_{feti} \vee \sum_{i=1}^{T_f} s_{feti}) \vee \neg c_{fet}$$

The logical equation can be rewritten as a linear equation

$$[\sum_{i=1}^{T_f} x_{feti} + \sum_{i=1}^{T_f} s_{feti}] + 1 - c_{fet} \geq 1 \quad (7.2)$$

- Row 4

$$\sum_{i=1}^{T_f} x_{feti} = 0 \wedge \sum_{i=1}^{T_f} s_{feti} = 1 \Rightarrow c_{fet} = 1$$

$$\neg(\sum_{i=1}^{T_f} x_{feti} = 0 \wedge \sum_{i=1}^{T_f} s_{feti} = 1) \vee c_{fet} = 1$$

$$\neg(\neg \sum_{i=1}^{T_f} x_{feti} \wedge \sum_{i=1}^{T_f} s_{feti}) \vee c_{fet}$$

Applying De Morgan law

$$(\sum_{i=1}^{T_f} x_{feti} \vee \neg \sum_{i=1}^{T_f} s_{feti}) \vee c_{fet}$$

The logical equation can be rewritten as a linear equation

$$[\sum_{i=1}^{T_f} x_{feti} + (1 - \sum_{i=1}^{T_f} s_{feti})] + c_{fet} \geq 1 \quad (7.3)$$

## 7.2 Model implementation

The following GitHub repository reports the model implementation for ILP and heuristic, and the instances generated for the repetition of simulations.

### 7.2.1 Simulations repetition

Script for the validation test. For ILP limit the size of the instance.

#### Generator of instances

Script *instance_generator/generator.py* generates instances for the simulation. Table 7.1 defines the parameters the script accepts. Each new instance $X$ is stored in the folder *instance_generator/instances/instance_X*, which contains three files *description.txt*, *network.json*, and *requests.json*. The description reports the instance configuration parameters as the instance size, number of requests and granularity of scheduling, while the network file contains the network configuration, and *requests.json* contains the requests of TS-traffic as declared in table 2.3.

| **Parameter** | **Description** | **Required** (Y/N) | **Accepted values** |
|---|---|---|---|
| Processing mode (-pm) | Network interfaces processing mode | Y | storeAndForward, fast |
| SF duration (-t) | T parameter of TSN cycle | Y | int |
| Path generation mode (-p) | Random generation of path strategies | Y | wifi2dc, wifi2wifi |
| Number of requests (-n) | Random generation of path strategies | Y | wifi2dc, wifi2wifi |
| Size (-s) | Minimum size for the TS-requests in bytes | Y | int, $\geq 64$ |

Table 7.1: Instances generator parameters

#### ILP validation

Script *validation_ilp.py* runs the simulation given a generated instance folder. It allocates iteratively one TS request, then stores the new network configuration and allocates the next requests. The output of the process is a binary file *out_ilp.pickle*, which is stored in the instance folder. The information stored is about the solving and pre-processing time, the requests that are accepted or blocked, and the maximum jitter and delay after each iteration. Table 7.2 reports the parameters of the script.

| **Parameter** | **Description** | **Required** (Y/N) | **Accepted values** |
|---|---|---|---|
| Instance (-i) | Instance folder generated with instance generator, it contains at least network.json and requests.json files | Y | path |

Table 7.2: Simulation parameters

### Heuristic validation

Script *validation_heuristic.py* runs the simulation given a generated instance folder. It allocates iteratively one TS request, then stores the new network configuration and allocates the next requests. The output of the process is a binary file *out_heuristic.pickle*, which is stored in the instance folder. The information stored is about the solving and pre-processing time, the requests that are accepted or blocked, and the maximum jitter and delay after each iteration. Table 7.2 reports the parameters of the script.

Other scripts in the utility folder take the binary file produced as the output of the simulation, process it and produce a .csv file with cumulative maximum and minimum delay per iteration and graphs to summarize the data. Each script is documented with a script helper which is printed in the terminal with the parameter *-h*.

# Bibliography


[1] Daniel Bezerra, Assis T. de Oliveira Filho, Iago Richard Rodrigues, Marrone Dantas, Gibson Barbosa, Ricardo Souza, Judith Kelner, and Djamel Sadok. A machine learning-based optimization for end-to-end latency in TSN networks. *Computer Communications*, 195:424–440, 2022.

[2] Gianluca Cena, Stefano Scanzio, and Adriano Valenzano. SDMAC: A Software-Defined MAC for Wi-Fi to Ease Implementation of Soft Real-Time Applications. *IEEE Transactions on Industrial Informatics*, 15(6):3143–3154, 2019.

[3] Hamza Chahed and Andreas Kassler. TSN Network Scheduling-Challenges and Approaches. *Network*, 3(4):585–624, 2023.

[4] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. page 183–192, 2016.

[5] Pau Fonseca i Casas. A Continuous Process for Validation, Verification, and Accreditation of Simulation Models. *Mathematics*, 11(4), 2023.

[6] Joaquim A.S. Gromicho, Jelke J. van Hoorn, Francisco Saldanha da Gama, and Gerrit T. Timmer. Solving the job-shop scheduling problem optimally by dynamic programming. *Computers Operations Research*, 39(12):2968–2977, 2012.

[7] Gurobi Optimization, LLC. Gurobi Optimization, LLC. `https://www.gurobi.com`, 2024.

[8] Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, and Saad Mubeen. Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, RTNS '21, page 68–77, New York, NY, USA, 2021. Association for Computing Machinery.

[9] IBM. IBM ILOG CPLEX Optimization Studio. `https://www.ibm.com/products/ilog-cplex-optimization-studio`, 2024.

[10] IEEE Standards Association. IEEE 802.1 Time-Sensitive Networking Task Group. `https://www.ieee802.org/1/pages/tsn.html`. Accessed on: 2024-05-09.

[11] Ilan Schnell. Bitarray. `https://pypi.org/project/bitarray/`, 2024.

[12] Hyeong Jun Kim, Kyoung Chang Lee, and Suk Lee. A Genetic Algorithm based Scheduling Method for Automotive Ethernet. In *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–5, 2021.

[13] Jonatan Krolikowski, Sébastien Martin, Paolo Medagliani, Jérémie Leguay, Shuang Chen, Xiaodong Chang, and Xuesong Geng. Joint routing and scheduling for large-scale deterministic IP networks. *Computer Communications*, 165:33–42, 2021.

[14] Ko Kusudo, Fumihiko Ino, and Kenichi Hagihara. A bit-parallel algorithm for searching multiple patterns with various lengths. *Journal of Parallel and Distributed Computing*, 76:49–57, 2015. Special Issue on Architecture and Algorithms for Irregular Applications.

[15] Yinzhi Lu, Liu Yang, Simon X. Yang, Qiaozhi Hua, Arun Kumar Sangaiah, Tan Guo, and Keping Yu. An Intelligent Deterministic Scheduling Method for Ultralow Latency Communication in Edge Enabled Industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, 19(2):1756–1767, 2023.

[16] Junhong Min, Yongjun Kim, Moonbeom Kim, Jeongyeup Paek, and Ramesh Govindan. Reinforcement learning based routing for time-aware shaper scheduling in time-sensitive networks. *Computer Networks*, 235:109983, 2023.

[17] Zaiyu Pang, Xiao Huang, Zonghui Li, Sukun Zhang, Yanfen Xu, Hai Wan, and Xibin Zhao. Flow Scheduling for Conflict-Free Network Updates in Time-Sensitive Software-Defined Networks. *IEEE Transactions on Industrial Informatics*, 17(3):1668–1678, 2021.

[18] Stuart Mitchell. PuLP. `https://pypi.org/project/PuLP/`, 2024.

[19] Ammad Ali Syed, Serkan Ayaz, Tim Leinmüller, and Madhu Chandra. Dynamic scheduling and routing for tsn based in-vehicle networks. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2021.

[20] L. Velasco, G. Graziadei, Y. El Kaisi, J. Villares, O. Muñoz, J. Vidal, and M. Ruiz. Provisioning of Time-Sensitive and non-Time-Sensitive Flows: from Control to Data Plane. *IFIP Networking 2024, TENSOR*, 2024.

[21] L. Velasco and M. Ruiz. Supporting time-sensitive and best-effort traffic on a common metro infrastructure. *IEEE communications letters*, 24(8):1664–1668, Aug 2020.

[22] Xiaolong Wang, Haipeng Yao, Tianle Mai, Tianzheng Nie, Lin Zhu, and Yunjie Liu. Deep Reinforcement Learning aided No-wait Flow Scheduling in Time-Sensitive Networks. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 812–817, 2022.

[23] Yang Wang, Jidong Chen, Wei Ning, Hao Yu, Shimei Lin, Zhidong Wang, Guanshi Pang, and Chao Chen. A time-sensitive network scheduling algorithm based on improved ant colony optimization. *Alexandria Engineering Journal*, 60(1):107–114, 2021.

[24] Wikipedia contributors. Abstract factory pattern — Wikipedia, The Free Encyclopedia, 2024. [Online; accessed 19-May-2024].

[25] Wikipedia contributors. Network throughput - Wikipedia, The Free Encyclopedia, 2024. [Online; accessed 13-May-2024].

[26] Hegen Xiong, Shuangyuan Shi, Danni Ren, and Jinjin Hu. A survey of job shop scheduling problem: The types and models. *Computers Operations Research*, 142:105731, 2022.