Embedded Reconfiguration of TSN: Dual Reconfiguration with Dropping and Reclaiming

Álex Gracia[1]
Dept. Informática e Ingeniería de Sistemas / I3A, Universidad de Zaragoza, Spain

Alitzel G. Torres-Macías
Dept. Informática e Ingeniería de Sistemas / I3A, Universidad de Zaragoza, Spain
CINVESTAV Unidad Guadalajara, Mexico

Juan Segarra
Dept. Informática e Ingeniería de Sistemas / I3A, Universidad de Zaragoza, Spain

José L. Briz
Dept. Informática e Ingeniería de Sistemas / I3A, Universidad de Zaragoza, Spain

Antonio Ramírez-Treviño
CINVESTAV Unidad Guadalajara, Mexico

Héctor Blanco-Alcaine
Intel Deutschland GmbH, Germany

Abstract

This paper provides a solution to the 37th ECRTS - Industrial Challenge of TSN reconfiguration. The proposed solution is based on applying incremental and global scheduling over the streams under failure, leveraging Mixed Integer Linear Programming (MILP) models. The rationale behind is that the incremental approach can quickly recover critical streams, whereas the global one can yield optimal schedules or find more solutions in tougher cases. The analysis times range from a few seconds to hours, depending on which links are failing, irrespective of the number of affected streams. We also propose an improvement to our main approach in which replicated streams are scheduled in the same network. The scheduling times would be longer, but replicated streams will not be interrupted by single-link failures.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Dependable and fault-tolerant systems and networks; Networks → Network protocols; Networks → Network performance evaluation

Keywords and phrases 802.1Qbv, TSN, TAS, GCL, Scheduling, Real-Time

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding This work was supported by MCIN/AEI/10.13039/501100011033 (grant PID2022-136454NB-C22), and by Dept. of Science, University and Knowledge Society, Government of Aragón (grant to reference research group T58_23R), Spain.
Alitzel G. Torres-Macías: SECIHTI, grant 1141966, Mexico.

1 Introduction

Challenge: Industrial challenge: Embedded reconfiguration of TSN [1].[2][3]
Type of solution: Full solution (not fully implemented).

[1] Corresponding author
[2] https://www.ecrts.org/industrial-challenge-current-challenge-thales/
[3] https://github.com/ecrtsorg/challenge2025-tsn

The challenge involves reconfiguring the stream schedule in the proposed network after port/link failures, so that real-time parameters of the streams are met. We have explored a Mixed Integer Linear Programming (MILP) approach upon the following assumptions:

- Recovery of affected streams is guided by a utility function, which in this proposal corresponds to the *priority* field in the provided stream list. This approach ensures that the most critical streams, as defined by their assigned priorities, are recovered first after a failure. Nonetheless, the proposed method can leverage other utility functions.
- Three out of the 241 streams provided have an invalid path. We have fixed the path by choosing a valid one.
- We assume the following values for unspecified parameters in the challenge:
  - Maximum clock skew between nodes: 500 ns.
  - Propagation delay of links: 0 ns.
  - Processing time of bridges: 1000 ns.

2 Description of the proposal

Our proposal assumes a list of streams, ordered at design time according to their priority. Although streams in this challenge are unique, our proposal supports replicated streams. The latter prevents single link failures from jeopardizing critical streams. Our recovery method can reschedule affected replicas in the same manner as unique streams. This idea is further explained in Sec. 4.

The fault recovery algorithm (Alg. 1) is divided into three stages: pre-scheduling (lines 1 to 1), optimal incremental scheduling on a per-stream basis (lines 3 to 9), and optimal global scheduling of a set of streams (from line 10 on). We leverage MILP models in the scheduling stages, which exploit the available resources in the network —free gaps, as well as the reclaimed transmission windows from the affected streams on the currently operational links—to reschedule the streams affected by the failures.

In the incremental stage, the algorithm applies a MILP to schedule one stream at a time. Thus, it consumes gaps progressively, reducing their availability to lower-priority streams, and increasing the difficulty of reaching feasible schedules [5] (Sec. 5). In the global optimal scheduling, streams are modeled together and the schedule provided in the solution is optimal for the group of streams [4], i.e., scheduling one stream does not preclude the scheduling of another. If the MILP yields no solution, the stream or group is considered unschedulable. Incremental and global scheduling models optimize similar but different parameters (see details in [5, 4]). The MILP model guarantees frame isolation under certain conditions, which are not met in this challenge. Nonetheless, all critical streams and most other flows accomplish frame isolation in our experiments (Sec. 3). In this use case, frame isolation could be guaranteed by adapting the class-assignment algorithm in [5] so that specific streams are always isolated. Alternatively, our MILP models support additional constraints to ensure that streams using the same queue are spaced enough. Any of these approaches is viable and easy to implement.

Alg. 1 starts when a faulty port/link is identified (line 1). The detection is out of the scope of the challenge [1] (Sec. 2, last paragraph). The algorithm takes a list of streams as input, ordered by utility, generated at design time. Using this list and the failing port, all streams traversing the failing port are added to the disposable stream ordered list (line 1). Optionally, working streams with a lower utility may also be added to the disposable stream

Algorithm 1 Algorithmic description of our proposal

Require: List of streams ordered by utility
1: Fault detection disposableStreamListGet ordered sublist of disposable schedules
2: Purge(disposableStreamList)
3: **for all** *stream* in disposableStreamList **do**
pathListGet ordered list of possible paths to reschedule *stream* **for all** *path* in pathList **do**
5: Generate MILP model to schedule *stream* on *path*
6: Solve model
7: **if** schedulable **then**
8: Deploy schedule for *stream*
9: Exit loop on pathList
10: **while** not satisfied with the schedule **do**
11: Generate MILP model to schedule a set of streams to schedule
12: Solve model
13: **if** schedulable **then**
14: Purge(streamSet) and Deploy
15: **if** scheduled set of streams contains all streams **then**
16: The system is optimal; exit
17: **else**
18: Add more streams to the set
19: **else**
20: Remove streams from the set
21: **if** the new set of streams has already been tested **then**
22: The system cannot be further improved; exit

list to increase the scheduling capacity in the network.[4]

Line 2 purges the disposable streams. We set a DROP policy for all streams in the disposable stream list in all the bridges, and then we remove all GCL windows associated to these streams in all bridges. After a time span equal to the frame transmission time plus link propagation delay plus bridge processing time, no disposable frames exist in the network, except those frames just transmitted from talkers that have not yet arrived at any bridge.

Lines 3 to 9 perform the optimal scheduling on a per-stream basis. For each stream in the disposable list, the algorithm finds all the possible paths for the stream and sorts them by length, shortest first (line 3). Then, it generates its MILP model (Sec. 5) assuming the current network state and the chosen path (line 5), and solves the model (line 6). If schedulable, we now deploy the corresponding GCLs and remove its corresponding DROP policies (line 8). In this case, no more paths for the current stream have to be tested (line 9). The obtained schedule is optimal for the stream, i.e., there exists no better schedule for this stream and path in the current network state. If it cannot be scheduled, try the next path in the path list. If a stream cannot be scheduled on any path (or if there is no valid path from its talker to its listener), we ignore the stream: it is not schedulable in the current network state.

[4] The decision of which working streams can be considered as disposable should be taken at design time, after a statistical analysis of failures and their effects. In this work, only streams traversing the failing link are considered disposable.

Lines 10 to 22 perform the optimal scheduling of *sets of streams*. At this point, some/all of the streams have been rescheduled and deployed.[5] This final stage jointly optimizes the deployed streams and, if possible, adds those that the previous stage was unable to schedule. This is done leveraging a global model describing a set of streams [4]. This model is slower to solve, and it either provides the optimal schedule for the set of streams as a group or fails if it cannot find a schedule for the whole set. Our approach tackles this scheduling by following a sort of binary search. So, a given set of streams is generated iteratively, expanding or contracting the set to converge to the largest schedulable set of streams in each iteration. We leave open the specific policies to build/update the set of streams.

3 Experimental results

We have conducted the experiments on an Intel Xeon Gold 5120 CPU (28 cores, 56 total threads, 2017, a relatively old processor). We rely on CBC [2], an open-source solver for MILP problems.

Tab. 1 breaks down the experimental times of rescheduling the streams affected by a link failure (lines 3 to 9 in Alg. 1). The final part of the algorithm has not been applied because all streams were scheduled during the first scheduling stage. Also, they all resulted schedulable using the shortest path, i.e., a single iteration over the path list sufficed when fewer than three consecutive link failures occurred.

The table shows both the time devoted to solving MILP models and scripts to process auxiliary information. The time of scripts would be negligible in a final implementation, since all shell/Python scripts and temporal files could be implemented as optimized code and library calls to solvers.

Solving the MILP rescheduling model exhibits substantial time variability, depending on which link has failed. This is due to the number of frames required to be scheduled in the hyperperiod for each particular stream. Streams with many frames in the hyperperiod (up to 32 frames) are much harder to schedule than streams with just one frame per hyperperiod. In some experiments, the solving time is beyond eight hours. In practice, a solver timeout can be set to return the best schedule found within the time limit, allowing additional streams to be rescheduled

Apart from analyzing scheduling times, we have also tested how many consecutive failing links our system could deal with. We successfully rescheduled all affected streams with up to three failing links. Not all failure combinations were evaluated, as five or more failing links often disconnect some talkers from their listeners, making the corresponding streams unschedulable.

Our current implementation does not guarantee frame isolation, but we have confirmed that the resulting schedules for the highest priority streams in our experiments (allocated to class #7) meet the frame isolation constraint in the proposed use case. Since one of our optimized parameters is talker-to-listener delay, most streams are scheduled in a no-wait way, which guarantees frame isolation.

[5] In our experiments, considering up to three consecutive link failures, all streams have been correctly deployed at this point.

Table 1 Time breakdown, in seconds, for rescheduling affected streams by a link failure.

			Optimal solution	
Faulty link	**Scheduled streams (number)**		**MILP**	**Scripts**
SW1-SW2	First TC7 affected	(1)	16.5	9.8
SW1-SW2	All TC7 affected	(3)	26.1	50.3
SW1-SW2	All affected	(24)	808.8	668.3
SW2-SW3	First TC7 affected	(1)	11.0	164.6
SW2-SW3	All TC7 affected	(3)	49.9	429.4
SW2-SW3	All affected	(22)	1914.3	1188.3
SW2-SW5	First TC7 affected	(1)	8.9	59.2
SW2-SW5	All TC7 affected	(6)	67.3	307.6
SW2-SW5	All affected	(23)	1058.4	971.2
SW3-SW4	First TC7 affected	(1)	4.1	20.8
SW3-SW4	All TC7 affected	(3)	35.6	864.4
SW3-SW4	All affected	(19)	2283.3	1951.3

4 Conclusions and further research

Our approach is based on incremental [5] and global [4] TSN scheduling. The scheduling problem is formulated as a MILP model whose solution directly yields the schedule, from which the TAS GCLs are derived. This approach can provide fast optimal schedules on a stream-by-stream basis, and slower but optimal schedules for sets of streams.

In order to speed up the scheduling of high-priority streams, we can dispose of lower-priority streams so that their corresponding resources can be used to schedule higher-priority ones. We have tested this approach, but in general, it is worthless. Higher-priority streams may be scheduled slightly faster, but the cost of rescheduling these unnecessarily disposed streams is higher than just maintaining their schedule. This approach could be studied for extremely low-priority streams, but not in general.

We have also tested stopping the solver at any feasible schedule, not necessarily the optimal one. However, even though the first schedules are obtained slightly faster, they increase the fragmentation of the gaps in the network, increasing the scheduling time of the following streams.

Further research

Frame replication on the TSN can be of interest, even in the presence of a replicated network, because it reduces the number of streams to recover. We have been able to schedule replicas of all the critical streams for the use case of this challenge. These replicas are routed along links disjoint from the original stream's path, enabling tolerance to single-link failures. Upon link failures, failing replicas would be rescheduled as any other failing stream. Further experimentation would shed light on viability and actual gains. Alg. 1 can also be applied to a whole replicated system. The conventional FRER would be substituted by a managed FRER, where several replicated streams may be accommodated in different paths of the same network if they cannot be scheduled in their original replicated network. Also, if replicated networks are connected, our schedules could use paths traversing several interconnected

networks.

We have successfully leveraged the MILP approach in industrial use cases, in which the periods of the applications are often designed taking into account a target network cycle. If feasible in the use case of this challenge, targeting a shorter hyperperiod when designing the systems could significantly improve the efficiency of MILP-based approaches.

CBC is highly parameterizable. A more systematic exploration could be of interest. Also, other solvers such as Gurobi [3] could significantly reduce solving times.

5 Effort

We discovered this challenge recently. It has been addressed in 6 working days by 2 full-time students (1 PhD and 1 Master student), 3 academic advisors, and 1 senior industrial expert. Understanding the challenge has required no significant time. We have leveraged our own recent [4] and current on-going work [5, 4].

Appendix: Overview of reference [5], currently under revision

This work proposes a novel incremental scheduling method for TSN traffic that leverages the available gaps (time intervals during which no critical traffic is transmitted) in an existing feasible schedule, regardless of the method used to set it. Upon arrival of a new stream to the network, a MILP model is generated according to the specifications of this stream, the gaps in the schedule, and the network topology. Solving the MILP yields a solution that is compliant with the existing schedule, leading to a new feasible schedule that includes the new and previous streams. Then, the new schedule is deployed in the TSN, reconfiguring the GCLs.

The proposed model introduces a flexible and efficient scheduling approach that overcomes the limitations of fixed-size gaps and no-wait constraints by leveraging complete system information from existing schedules. It guarantees optimality and uses heuristics only to isolate streams on the optimal schedule. Thus, it can still provide non-isolated schedules when isolation is not possible. It leverages IEEE 802.1Qbv queues to optimize the schedule, synthesizing GCL entries, and optimizes jitter and end-to-end latency, scheduling transmissions as soon as possible to ease the scheduling of further streams. It is significantly faster than rescheduling the full set of streams for each new stream arrival, making it suitable for online scheduling scenarios. Additionally, its MILP-based formulation allows for diverse optimization objectives and can be integrated into any framework that separates routing from scheduling to improve overall performance.

References

1 Marc Boyer and Rafik Henia. Industrial challenge: Embedded reconfiguration of TSN. working paper or preprint, July 2024. URL: https://hal.science/hal-04630862.

2 John Forrest, Ted Ralphs, Stefan Vigerske, Haroldo Gambini Santos, John Forrest, Lou Hafer, Bjarni Kristjansson, jpfasano, EdwinStraver, Jan-Willem, Miles Lubin, rlougee, a andre, jpgoncal1, Samuel Brito, h-i gassmann, Cristina, Matthew Saltzman, tosttost, Bruno Pitrus, Fumiaki MATSUSHIMA, Patrick Vossler, Ron @ SWGY, and to st. coin-or/cbc: Release releases/2.10.12, August 2024. doi:10.5281/zenodo.13347261.

3 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

4 Alitzel Galilea Torres-Macías, Juan Segarra Flor, José Luis Briz, Antonio Ramírez-Treviño, and Héctor Blanco-Alcaine. Fast IEEE802.1Qbv Gate Scheduling Through Integer Linear

Programming. *IEEE Access*, 12:111239–111250, 2024. URL: https://doi.org/10.1109/ACCESS.2024.3440828.

5 Alitzel Galilea Torres-Macías, Juan Segarra Flor, José Luis Briz, Antonio Ramírez-Treviño, and Héctor Blanco-Alcaine. Optimal and Fast IEEE802.1Qbv Incremental Scheduling. Submitted, under revision, 2025.