Scheduling in TSN networks extended for class II

Kateřina Brejchová, Marek Vlk
Thursday 14.30
Open informatics – Artificial intelligence
brejcka1@fel.cvut.cz

I. Assignment

A. Problem Statement

My semestral thesis extends our current work in the field of Scheduling in TSN networks. The work currently focuses on the most critical traffic class I and I am extending it for the second most critical traffic class II. As the original problem statement is already written perfectly by Marek Vlk, I only adjusted it a bit instead of rephrasing it:

We model the network as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, where set of nodes $\mathcal{V}$ represents the switches and the end-stations (such as camera, PLC, sensor, actuator), and set of edges $\mathcal{L}$ represents the physical connections (Ethernet links) between the nodes. Each link $(v_a, v_b) \in \mathcal{L}$ is specified by its speed $(v_a, v_b).c$, the number of queues available for time-triggered traffic $(v_a, v_b).q$, and link delay $(v_a, v_b).d$. Each switch $v_a \in \mathcal{V}$ is also characterized by its switching delay $v_a.d$. We assume all the devices in a network are time-synchronized and all links have the same macrotick. The maximum synchronization error between two devices in the network is denoted as δ.

Communication in the network is carried out through the concept of flows. A flow is a periodic data transmission from one end-station to another end-station. The set of flows is denoted as $\mathcal{S}$. Each flow $s_i \in \mathcal{S}$ is specified by talker $s_i.src$, which is the end-station ID that is sending the data, listener $s_i.dest$, which is the end-station ID that receives the data, payload $s_i.pl$ in bytes, period (data cycle) $s_i.Tdc$, deadline $s_i.\tilde{d}$, which is the maximum time till when the listener must receive the data, and release time $s_i.r$, which is the minimum time when the talker can start sending the data. It must hold that $0 \leq s_i.r \leq s_i.\tilde{d} \leq s_i.Tdc$. We assume that each flow does not transmit more data than what can fit into one MTU-sized Ethernet frame of 1542 bytes. The flows can have different periods, which can result in hyperperiod (network cycle) larger than the period of any flow. Consequently, one flow may occur more than once in the hyperperiod. These occurrences of a flow on a link are referred to as frames. Let us denote hyperperiod Tnc as the least common multiple (lcm) of the periods of all flows: $Tnc = lcm(\{s_i.Tdc \mid s_i \in \mathcal{S}\})$. The set of all frames that are routed through link $(v_a, v_b) \in \mathcal{L}$ is denoted as $\mathcal{F}^{(v_a,v_b)}$. Note that this can be an empty set. Further, $f_i^{(v_a,v_b)}$ refers to the first frame of $s_i \in \mathcal{S}$ routed through link $(v_a, v_b) \in \mathcal{L}$ and $f_{i,j}^{(v_a,v_b)} \in \mathcal{F}^{(v_a,v_b)}$ refers to the j-th periodic occurence of the frame of flow $s_i \in \mathcal{S}$ routed through link $(v_a, v_b) \in \mathcal{L}$. We assume that for class I, zero jitter (strictly periodic schedule) is required, i.e., the time distance of every subsequent frame of flow $s_i \in \mathcal{S}$ is always equal to $s_i.Tdc$.

We introduce integer variable $f_i^{(v_a,v_b)}.\phi$, $\forall (v_a, v_b) \in \mathcal{L}, \forall f_i^{(v_a,v_b)} \in \mathcal{F}^{(v_a,v_b)}$, representing the transmission offset of frame $f_i^{(v_a,v_b)}$, and integer variable $f_i^{(v_a,v_b)}.p$, $\forall (v_a, v_b) \in \mathcal{L}, \forall f_i^{(v_a,v_b)} \in \mathcal{F}^{(v_a,v_b)}$, representing the queue ID to which frame $f_i^{(v_a,v_b)}$ is stored. Let $f_i^{(v_a,v_b)}.t$ be the time it takes to transmit frame $f_i^{(v_a,v_b)}$ over link (v_a, v_b), which is referred to as transmission duration and is calculated as $f_i^{(v_a,v_b)}.t = \frac{8 \cdot s_i.pl}{(v_a,v_b).c}$.

We are considering the following constraints for class I:

1) Frame Constraint: This constraint ensures that transmitting a frame on the first link of the routed path cannot start before the release time and must be completed on the last link before the deadline.
2) Link Constraint: The transmissions of two distinct frames on a link must not overlap in time.
3) Flow Transmission Constraint: The transmission of a frame can start only after the previous frame is fully delivered to the switch and processed.
4) Queue Usage Bounds: The number of available queues is bounded.
5) Objective: The objective is to minimize the accrued sum of the number of queues used per egress port. The motivation is to keep as many queues as possible for other classes of (not time-triggered) traffic, achieving thus higher throughput.
6) Frame Isolation Constraint (FIC): To avoid the lost frame problem, the solution proposed by [1] is to add frame isolation constraint (FIC). It isolates two different frames such that one frame can be transmitted to a shared queue only after the other frame is dispatched from the queue.

The following changes for class II:

We introduce integer variable $f_{i,j}^{(v_a,v_b)}.\phi$, $\forall (v_a, v_b) \in \mathcal{L}, \forall f_{i,j}^{(v_a,v_b)} \in \mathcal{F}^{(v_a,v_b)}$, representing the transmission offset of frame $f_{i,j}^{(v_a,v_b)}$. The set of flows belonging to class II is denoted $\mathcal{S}^{[2]}$.

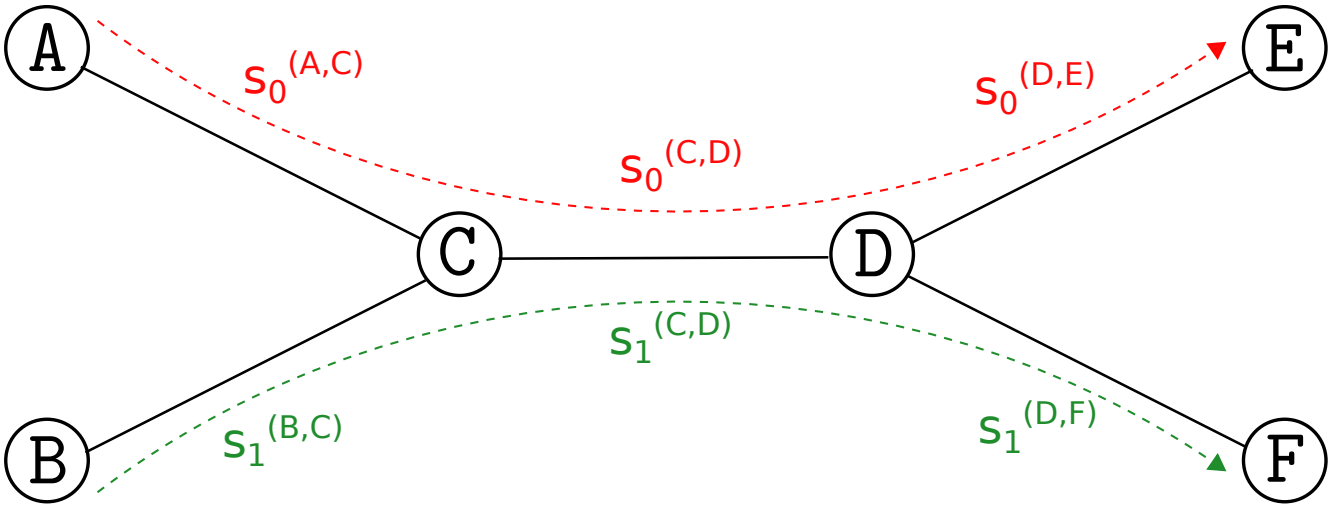


Fig. 1: Sample network with two flows of class II

We introduce notation for jitter $f_i^{(v_a,v_b)}.J$. The jitter says how much can the consequent periodic occurence of the flow instances differ in period. Zero jitter would mean that the consequent occurences are determined by the start time of the first occurence such that

$$f_{i,j}^{(v_a,v_b)}.\phi = j \cdot s_i.Tdc + f_{i,0}^{(v_a,v_b)}.\phi$$

. However, this does not hold when jitter is introduced. For better understanding of jitter and end to end latency, please see constraints (2), (3) and figures (1), (2).

We consider the following constraints for class II:

1) Frame Constraint: This constraint ensures that each $f_{i,j}^{(v_a,v_b)}$ must be send after the j-th period (periodic occurence) starts but before the $(j+1)$-th period starts.

$$\forall s_i \in \mathcal{S}^{[2]}, \forall f_{i,j}^{(v_{s_i.src},v_x)} \in \mathcal{F}^{(v_{s_i.src},v_x)} : (f_{i,j}^{(v_{s_i.src},v_x)}.\phi \geq j \cdot s_i.Tdc) \ \& \ (f_{i,j}^{(v_{s_i.src},v_x)}.\phi \leq (j+1) \cdot s_i.Tdc) \quad (1)$$

2) Link Constraint
3) Flow Transmission Constraint
4) Queue Usage Bounds: The number of available queues is bounded and class II flows use the first available queue with no scheduled traffic.
5) Objective
6) Jitter constraint: The frames $f_{i,j}^{(v_a,v_b)}$ should repeat regurarly with $s_i.Tdc$. However, for class II, the two consequent frames can differ in period by at most $f_i^{(v_a,v_b)}.J$.

$$\begin{aligned} &\forall s_i \in \mathcal{S}^{[2]}, \forall f_{i,j}^{(v_a,v_b)}, \forall j \in \{1,...,\frac{Tnc}{s_i.Tdc} - 1\} : |f_{i,j}^{(v_a,v_b)}.\phi - (f_{i,j-1}^{(v_a,v_b)}.\phi + s_i.Tdc)| \ \leq \ f_i^{(v_a,v_b)}.J \\ &\forall s_i \in \mathcal{S}^{[2]}, \forall f_{i,0}^{(v_a,v_b)}, k = \frac{Tnc}{s_i.Tdc} - 1 : |(f_{i,0}^{(v_a,v_b)}.\phi + Tnc) - (f_{i,k}^{(v_a,v_b)}.\phi + s_i.Tdc)| \ \leq \ f_i^{(v_a,v_b)}.J \end{aligned} \quad (2)$$

7) Latency constraint: The length of the time range that it takes the flow from the start on the first link to the end on the last link is restricted by maxiumum end to end latency.

$$\begin{aligned} &\forall s_i \in \mathcal{S}^{[2]}, \forall f_{i,j}^{(v_{s_i.src},v_x)} \in \mathcal{F}_i^{(v_{s_i.src},v_x)}, f_{i,j}^{(v_y,v_{s_i.dest})} \in \mathcal{F}_i^{(v_y,v_{s_i.dest})} : \\ &\qquad f_{i,j}^{(v_y,v_{s_i.dest})}.\phi + f_i^{(v_y,v_{s_i.dest})}.t + (v_y, v_{s_i.dest}).d - f_{i,j}^{(v_{s_i.src},v_x)}.\phi \leq s_i.L \end{aligned} \quad (3)$$

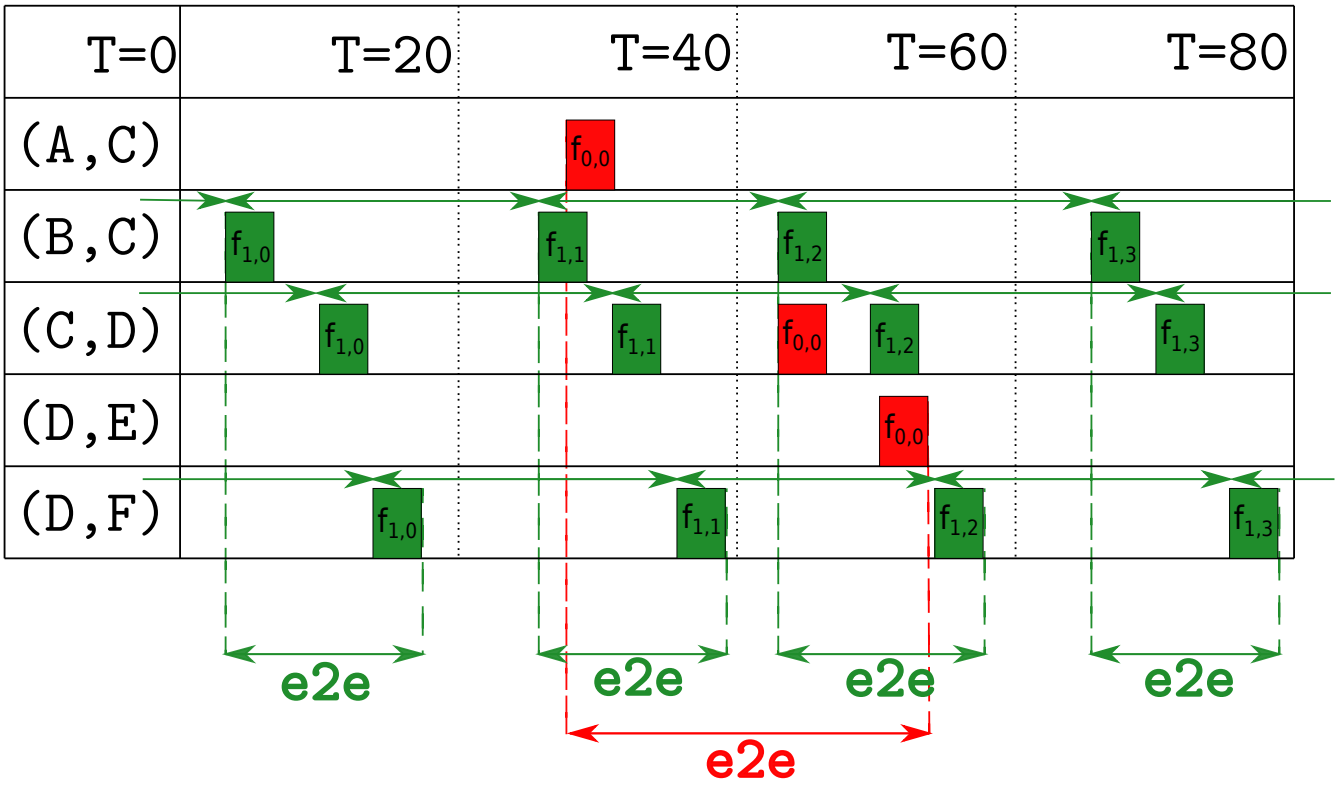


Fig. 2: Sample Gantt chart with two flows of class II, highlighted end to end latencies in each periodic occurence and highlighted start time ranges that we need to check with respect to the given maximal jitter and period. The presented frames are missing the upper link index for clearer presentation (it corresponds to the line of the Gantt chart). The Green flow has period 80, the red flow has period 20.

B. Problem Categorization

The goal of this work is to add flows of class II to the existing schedule with class I. More specifically to assign start time and queue to each frame such that all constraints are sattisfied.

II. Related works

Minaeva et al. [2] address the problem by creating a constructive heuristic that sequentially adds tasks to the schedule based on priority order. In the case of the infeasibility of the given task, a reason graph is used to help with the backtracking. After this step, the schedule is optimized by local neighborhood search using ILP.

Syed and Fohler [3] present a search-tree pruning heuristic based on job response-time. The heuristic searches for groups of symmetrical sub-schedules and uses only one sub-schedule from the group in the searching process (tree pruning technique). They emphasize the fact that pruning of infeasible search-tree paths is as important as looking for new feasible ones and use Parallel Iterative Deepening A* to create the schedule.

Finally, Bansal creates a divide-and-conquer heuristic in his master thesis [4]. Contrary to most of the algorithms in the literature, he proposes a decentralized approach aiming mostly at large-scale networks. Following the divide-and-conquer paradigm, the schedule for each link is created separately, and then the schedules are completed together. However, from the text, it is unclear how the algorithm treats conflicts among the pre-created schedules.

There are several different approaches to consider when designing a heuristic algorithm. From the literature described above, it is clear that any introduction of backtracking significantly increases the time for which the solver runs. Therefore, we decided to focus on one pass heuristics that don't use any backtracking and their power lies in creating the order in which the frames are added to the schedule. This concept was partially introduced in [2] but was not deeply explored. Further, we apply the knowledge

gained from the one pass heuristics and design Conflict-Directed Backjumping [5] which is using dynamic granularity to speed up searching.

III. Solution and Implementation

As a first step towards maybe a better solution in the future, we will apply the following workflow of the new algorithm:

- Schedule flows of class I by existing methods.
- Add flows of class II to the existing schedule by a modified algorithm for Conflict-Directed Backjumping.

The reasoning behind this is that class I has more restrictive constraints, and based on [6], typically much smaller periods. This means that it will be harder to fit class I than to fit class II into the schedule. If solving the problem as two (almost) independent subproblems does not work, it would be necessary to come up with an algorithm that solves them all together or allows some rescheduling of flows of class I.

Since we removed Frame Isolation Constraint from class II specification, I decided to also implement Earliest Deadline First method as it makes a bit more sense than for class I. To test the performance on instances with both classes I and II, I used our current method called CBJ (exact version) to schedule class I flows. In total, three methods were implemented.

- CBJ + EDF_ZJ_2
- CBJ + EDF_JC_2
- CBJ + CBJ_2

All of the implemented methods are available at our Git repository.

A. Earliest deadline first

The scheduling unit is flow instance $f_i^{(v_a,v_b)}$. The algorithm iterates over start times on all links and always selects the flow instance with lowest LST (latest start time) to be scheduled. Next, the individual start times for the frame instances of the current flow instance are assigned. These start times are chosen as the earliest possible start times that comply with Frame Constraint, Flow Transmission Constraint and Jitter Constraint.

The assigned frame instances are then checked against already scheduled traffic for link consistency. In case of a link conflict, we try to shift the problematic frame instance either left or right. This is done by calculating the necessary earliest (right shift) or latest (left shift) start times that ensure avoiding the conflicting frame instance. We prefer the lower shift and check whether it is possible by checking jitter constraint (shift can be too large), link constraints (there can be another conflicting frame) and latency constraint (the shift can cause that in some of the periodic occurrences the maximal latency is exceeded). In case the shift is consistent, we reassign the start time for the current problematic frame instance. Otherwise, we try to perform the same process in the other shift direction.

After the conflicts are resolved for all the frame instances of the current flow instance, we can end up in two situations – The current assignment is *not consistent*, then we undo the current assignment and try to assign another available flow instance for the combination of time and link that we currently have or we proceed to the next link and/or to the next start time if that's not possible. The current assignment is consistent, then we perform postprocessing and continue by assigning the next combination of link and start time a flow instance. In the postprocessing step, we reduce the domains of the not yet assigned flow instances by updating the earliest and latest start time. These values are affected by the preceding or succeeding flow instances of the same flow (flow transmission and latency constraints).

The algorithm ends when it reaches the last available start time (hyper period). In case that all of the flow instances were assigned, the algorithm yields FEASIBLE solution, otherwise it yields UNKNOWN solution (heuristical algorithm cannot prove infeasibility).

The difference between JC (jitter constraint) and ZJ (zero jitter) version of the EDF algorithm is that ZJ does not perform shifts (i.e. uses jitter equal to 0 for all flow instances).

B. Conflict-Directed Backjumping

The scheduling unit is Frame Instance $f_{i,j}^{(v_a,v_b)}$. The main difference from EDF is that it uses backtracking and is complete in the sense that it always finds a solution if the solution exists. The general workflow is shown in Figure 3. After initialization, we gradually try to assign start times to each frame until we find a feasible schedule. To do so, we apply a search tree algorithm which backjumps each time a conflict is found. Backjumping is a special form of backtracking which skips assignments that would surely lead to infeasible solutions. This can be done by keeping a conflict set for each variable (frame).

The run of the algorithm can be significantly improved by choosing a suitable method for variable selection. There are many features one can take into account while doing so. Common strategies are for example: Earliest deadline first (frame with the

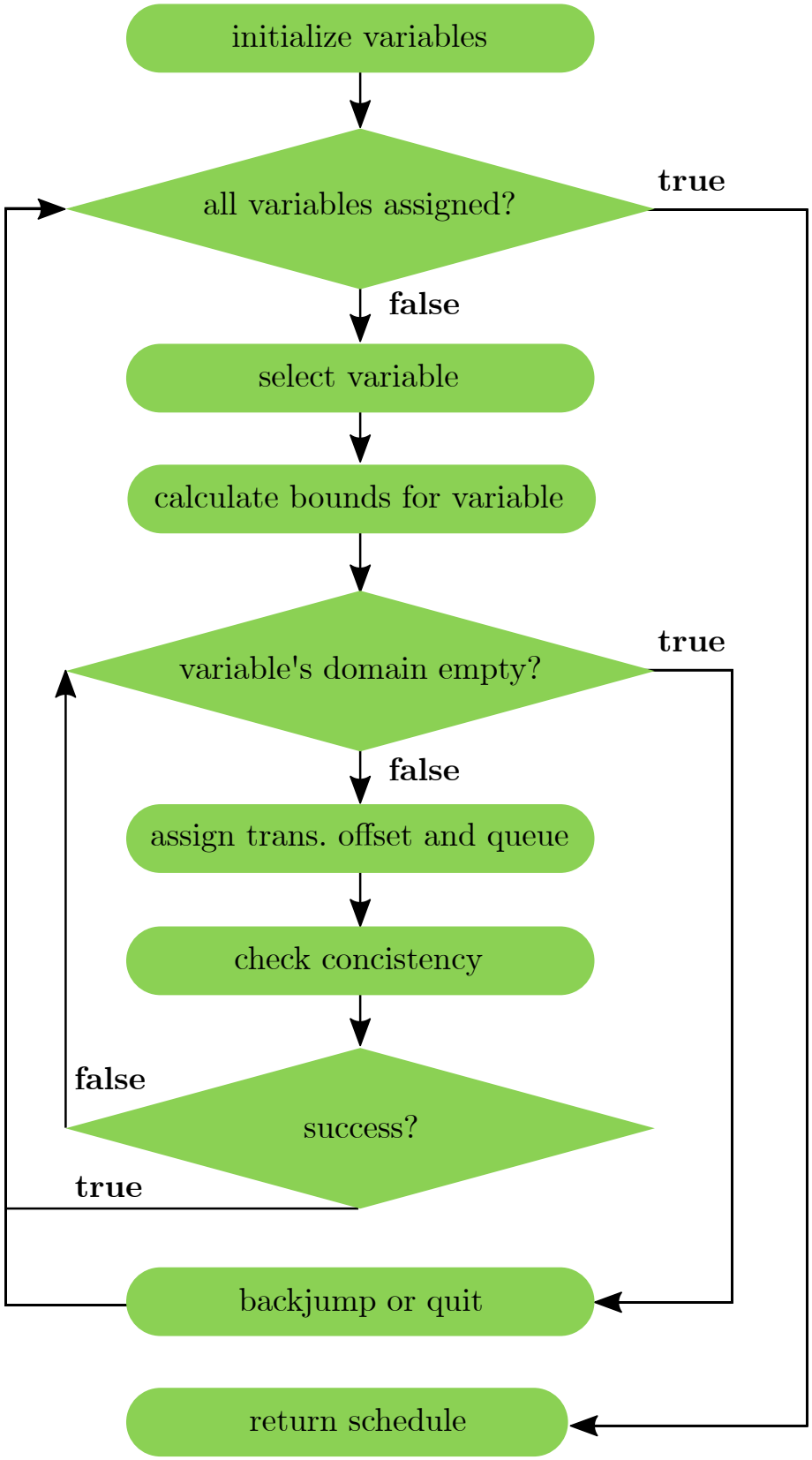


Fig. 3: Overview of the scheduling algorithm

earliest deadline is scheduled first) or Most required time (frame with the largest sum of durations over all periods is scheduled first). After exploring different options, we came up with a selection strategy that uses the following criterion:

$$crit = \frac{period \cdot mslk + id}{conf} + hpID \tag{4}$$

The criterion is calculated for each frame instance, and it is ensured that the precedences are kept by adding *id* which represents the order of flow instances in the flow path and *hpID* which represents the periodic repetition of the given flow instance. *Conf* represents the number of conflicts that the given flow encountered during the whole scheduling process and *mslk* corresponds to the size of the interval to which the flow instance can be scheduled based on the predecessors and successors. As there is no deadline for class II flows, we consider $s_i.\tilde{d} = s_i.Tdc$.

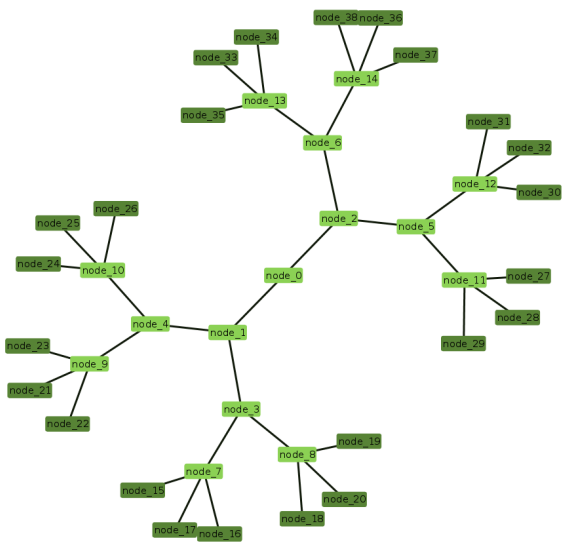
When the algorithm deploys, it prefers frames with small domain sizes and low periods, and as it continues, it starts do prefer frames with the most conflicts (the counter is incremented each time we reassign given frame).

The main differences in the implementation for class I vs class II CBJ are as follows:

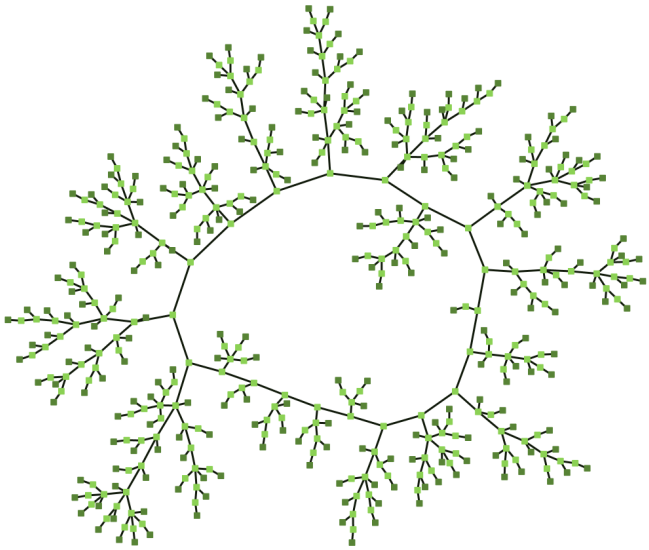
- *Granularity* – The scheduling unit is frame instance $f_{i,j}^{(v_a,v_b)}$, not flow instance $f_i^{(v_a,v_b)}$ as for class I.
- *Jitter* – After a frame instance of a given flow instance is scheduled, the bounds for the following frame instances are predetermined based on the jitter constraint. In case such bound tightened the interval to which the current variable can be scheduled, the preceding frame instance is added to the conflict set of the current variable.
- *Latency* – In case latency is exceeded for $f_{i,j}^{(v_a,v_b)}$, frame instance $f_{i,j}^{(v_{s_i.src},v_x)}$ is added to the conflict set.

IV. Experiments

We evaluated the proposed algorithms on instances suitable for highly critical Ethernet communication. We generated schedulable instances of various sizes and TREE or HYBRID TREE topology with the aim to have instances with sequentially growing average link utilization allowing us to test both easily and hardly schedulable instances. Example of large-sized tree topology and huge_500 hybrid tree topology is depicted by our GUI in Figures 4a, 4b. In total, we have six different topologies to test on. The topologies are in detail described in Table I.



(a) Large TREE topology with 39 nodes



(b) HYBRID TREE topology with 500 nodes

Fig. 4: Sample topologies

Parameters of nodes and links in the network are chosen to suit the Ethernet communication problem. Link speed is set to 1 Gbit/sec for all links. The propagation delay which is equal to $wire\ length/speed\ of\ light$ and is set to 0.17 μs, the node time lag is set to 10 μs for all nodes, and the synchronization delay is set to 0.1 μs.

The communication is generated using the as-soon-as-possible first-fit approach, talker and listener for each flow is chosen randomly. The periods are distributed uniformly randomly from set {500, 800, 1000, 2000, 4000} for class I flows and from set

$\{2000, 2500, 5000, 10000, 20000\}$ for class II flows. All units are in μs. 90 % of all flows are of class II, 20 % of all flows are of class I. Note, that this can actually mean that there is more traffic of class I since the periodic occurrence is more often. The frame size is selected randomly from range 72 to 1542 bytes to fit the Ethernet frame.

We set upper bound on number of flows for each topology size as shown in Table I and iterate from lower to upper bound with a step size equal to the interval size divided by 10 resulting in 10 different communication complexities for each topology. Upper bound ub on number of flows was set experimentally for each topology as the maximum number of flows for which the generator yielded a schedulable instance in a reasonable time (in order of hundreds of seconds). Lower bound lb was set as $lb = \frac{ub}{10}$ for each setting.

topMode	topSize	numSwitches	numEndSystems	lb on flows	ub on flows
TREE	SMALL	1	4	25	250
TREE	MEDIUM	3	16	50	500
TREE	LARGE	15	24	65	650
HYBRID_TREE	HUGE_100	28	72	150	1500
HYBRID_TREE	HUGE_500	151	349	500	5000
HYBRID_TREE	HUGE_1000	300	700	1000	10000

TABLE I: Parameters of different topologies

Release time $s_i.r$ and deadline $s_i.\tilde{d}$ are randomly set so that the interval covers 20–50 % of the period $s_i.Tdc$ and the interval is large enough to cover the sum of transmission durations over all reoccurred stream instances on the path of the stream. Maximum end to end latency is set to 50 % of the period. Allowed jitter is set to 10 % of the period for flow instances heading to/from end systems and to 10 % of the period otherwise .

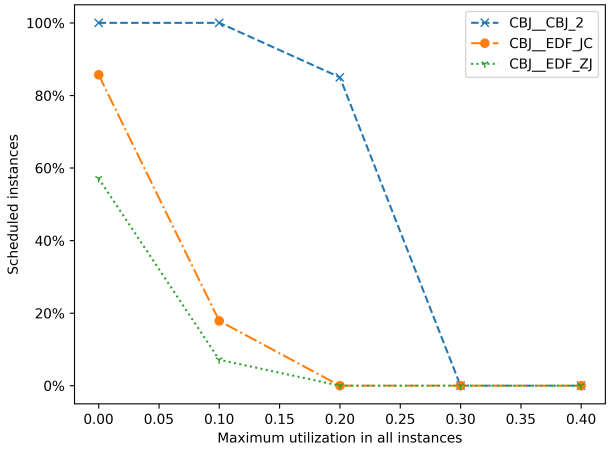
Summarized, we have 60 folders in total – each folder differs in topology size and the amount of transmitted communication. Each folder consists of 5 instances with the same parameters – the instances differ in the generated communication which is partially random.

The experiments were run on a system with 4x Intel® Xeon® CPU E5-2690 v4 @ 2.60GHz with 14 cores (56 cores in total) and in total 251GB of RAM. We set the time limit to 1800 seconds per one solver (half an hour). Note that the problem instance initialization (data loading, etc.) is not included in this time limit and is done in order of seconds.

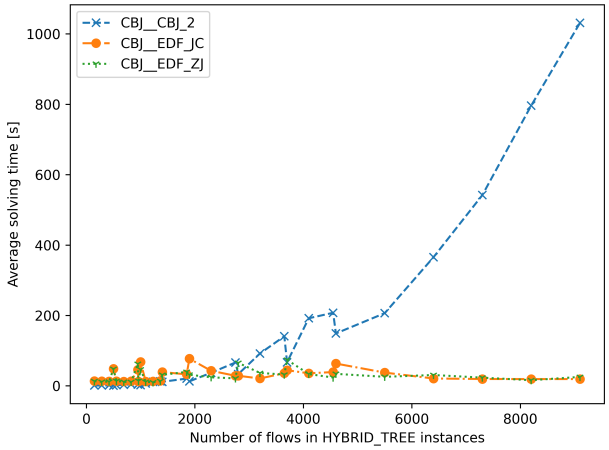
A. *Results*

In Table II we can see the individual success rate of each method on each topology type. We can see that CBJ significantly outperforms the EDF based methods. This is not much of a surprise since CBJ uses a form of backtracking but EDF is a one pass heuristic which means that it ends as soon as it encounters the first consistency conflict. We can also see that allowing jitter helps to increase schedulability, since EDF_JC performs better than EDF_ZJ.

In Figure 5a we can see that the increasing utility becomes a problem for the EDF methods since it is more probable that they will encounter a conflict due to a higher traffic rate. A problematic fact in a CBJ performance is the growth of solving time, see Figure 5b. It is possible that if we experimented on more complex instances (bigger topology, higher utilization), the time efficiency would be problematic.



(a) Success rate of the methods based on the maximal link utilization



(b) Time performance of the methods based on the number of flows on hybrid tree topologies

topMode	topSize	CBJ [%]	EDJ_JC [%]	EDF_ZJ [%]
TREE	SMALL	100	50	30
TREE	MEDIUM	70	20	10
TREE	LARGE	50	20	20
HYBRID_TREE	HUGE_100	100	20	0
HYBRID_TREE	HUGE_500	100	0	0
HYBRID_TREE	HUGE_1000	100	0	0
ALL	ALL	86.66	18.33	10

TABLE II: The success rate (percentage of scheduled instances) of the methods on different topologies

B. Discussion

Overall, it would be beneficial to perform experiments on a bit more complex instance, especially for CBJ. However, it is a positive surprise that the EDF methods are capable to schedule at least some of the instances as it was not possible when we run a similar EDF based algorithm for class I flows.

It may also be beneficial to compare the results with an ILP model. As the results of previous experiments on class I suggest, it would most certainly not perform better but we might get helpful insights on the CBJ correctness. We may also further improve CBJ performance by making it a heuristic – by reducing conflict sets or skipping some values from domain variables.

Note, that for the correctness of the algorithms it may be necessary to add constraint ensuring that there is no interference between class I and class II traffic. This interference can happen if some of the class II flows is lost or delayed. However, we haven't figured out the exact form of the constraint yet.

Another field of work that could be explored is a combined method for scheduling both class I and class II flows together. If we stick with the two-stage approach, it may be good to explore the options on how to optimize the class I schedule so that it allows more flows of class II to be scheduled.

V. Conclusion

This semestral work explores the possibilities of combining class I and class II scheduling in TSN networks. We show three solving approaches – two heuristics and one exact method. The methods do not use any proprietary software and are implemented in Java 8. We show that the problem is too complex to use only one pass heuristics and perform an experiment that compares the suggested methods.

References

[1] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 183–192. ACM, 2016.

[2] Anna Minaeva, Debayan Roy, Benny Akesson, Zdeněk Hanzalek, and Samarjit Chakraborty. Control Performance Optimization in Time-Triggered Periodic Scheduling. IEEE Transactions on Computers, submitted, 2019.

[3] Ali Syed and Gerhard Fohler. Efficient offline scheduling of task-sets with complex constraints on large distributed time-triggered systems. *Real-Time Systems*, 2018.

[4] Bharat Bansal. Divide-and-Conquer Scheduling for Time-sensitive Networks. Master's thesis, University of Stuttgart, 2018.

[5] Xinguang Chen and Peter Van Beek. Conflict-directed backjumping revisited. *Journal of Artificial Intelligence Research*, 2001.

[6] Time sensitive networks for flexible manufacturing testbed: Characterization and mapping of converged traffic types. Technical report, Industrial Internet Consortium, Object Management Group, Inc., 2019.