Resource Allocation Is All You Need: The Routing and Scheduling Problem in 6TiSCH Networks

Victor Gerling*, Henning Cui*, Jörg Hähner*, Michael Seufert†
* *University of Augsburg, Chair of Organic Computing*, Augsburg, Germany
† *University of Augsburg, Chair of Networked Embedded Systems and Communication Systems*, Augsburg, Germany, {victor.walter.gerling|henning.cui|joerg.haehner|michael.seufert}@uni-a.de

***Abstract*—Deterministic Wireless Sensor Networks over IEEE 802.15.4 can provide latency-bounded transmission of flows, which is an important enabler for current and future Internet of Things (IoT) use cases. To realise such networks, a viable routing and scheduling solution must be found that can accept all given flows and maintain their latency requirements. The *joint routing and scheduling* (JRaS) problem promises optimal routing and scheduling decisions—however, at the expense of very high computation times. To overcome this issue, we propose efficient modifications to the separate routing and scheduling problems, such that we can obtain a success rate *similar to the optimal JRaS approach*. However, our solutions can be found in *much less time*. We conduct extensive performance evaluations for different problem complexities and found a speedup in the range of $3.03\times$ up to $6.09\times$ compared to JRaS while having *almost no* statistical difference in success rates.**

***Index Terms*—WSN, IEEE 802.15.4, 6TiSCH, Deterministic Networking, DetNet, Routing, Scheduling, JRaS, Optimisation.**

I. Introduction

The Internet of Things (IoT) and the Tactile Internet are major enablers for civic, business, and industry applications, and thus, can bring enormous benefits for society as a whole. However, they also pose major challenges to network and service operators. According to a recent report [1], the number of connected IoT devices was 15.9 billion in 2023 and is expected to double by 2030. Thus, sophisticated network management techniques are required to cope with the high amount of connected devices and their communication requirements.

IoT devices are resource-constrained devices, which communicate at low data rates, e.g., sensors, which periodically upload a sensor value, or actuators, which receive a control instruction. They are often battery-powered and need to resort to wireless communication when deployed in situations which require mobility and/or in which they cannot rely on supporting infrastructure like cables for power and network access. In such Wireless Sensor Networks (WSNs), IEEE 802.15.4 [2] is the de facto standard to allow devices to exchange data with each other and/or with a well-connected gateway node.

To bring real-time capabilities to traditional networking devices, the IEEE Time-Sensitive Networking (TSN) and the IETF Deterministic Networking (DetNet) working groups are developing a series of standards, also for wireless communication. Here, IEEE 802.15.4 is considered—among others—a suitable candidate for deterministic wireless networks [3], especially for wireless industrial applications [4]. In particular, the Architecture for IPv6 over the Time-Slotted Channel Hopping Mode of IEEE 802.15.4 (6TiSCH) [5] can provide a low-latency, low-jitter, and high-reliability packet delivery.

Two problems have to be solved to achieve latency guarantees for traffic flows over 6TiSCH in WSNs: **routing and scheduling**. Clearly, as network management (or network planning in static scenarios) tries to efficiently use the available network resources, this constitutes an optimisation problem: make optimal routing and scheduling decisions for all latency-critical flows, such that their latency requirements can be held. While classical approaches solve both problems separately, i.e., first finding optimal paths and then finding an optimal schedule, cross-layer approaches have been proposed, such as [6], which solve the joint routing and scheduling (JRaS) problem. The advantage is that JRaS considers the space (routing) and time (scheduling) dimensions at the same time, which can be beneficial for successfully finding a viable solution. Although JRaS solutions are generally superior to separate routing and scheduling (SRaS), optimisation algorithms need to search a much larger solution space, and thus, require substantially more computation time.

In this work, to the best of our knowledge, we are the first to compare the performance of JRaS against a classic resource allocation approach. We propose efficient modifications to the SRaS problem by *allocating resources (bandwidth) for flows over different hops*, thereby, avoiding or mitigating bottlenecks. This results in a success rate similar to JRaS, but at a much lower runtime. To evaluate our proposal, we conducted an empirical study and compare three different approaches: 1. *Shortest path routing* – the most basic SRaS approach, which represents a lower bound baseline, 2. *JRaS* – the optimal solution as an upper bound baseline, and 3. our proposed approach, which is based on *constrained shortest path routing*. For this task, we choose a realistic communication scenario where all nodes periodically communicate with a centralized or externally facing WSN gateway. We consider different problem complexities in which we vary flow characteristics or the network topology. We analyse the success rate and time consumption of the generated solutions, which allows us to accurately quantify the practical value of our proposed approach for network management.

This paper is structured as follows. Section II provides background on the 6TiSCH stack and the joint routing and scheduling problem. Section III presents the problem formulation and our proposed modifications for obtaining efficient solutions. We describe the evaluation scenarios and results in Section IV. Finally, Section V concludes.

II. Background and Related Work

A. 6TiSCH

RFC 9030 [5] describes an architecture for IPv6 over the Time-Slotted Channel Hopping Mode of IEEE 802.15.4 (6TiSCH). The 6TiSCH architecture provides low-latency, low-jitter, and high-reliability packet delivery. In Time-Slotted Channel Hopping (TSCH), time is split into timeslots. The timeslots are organised into slotframes, and the slotframes repeat over time. Moreover, a transmission can choose from 16 frequencies. In Fig. 1, we see a possible schedule for node B. A green timeslot indicates that the node is listening, while orange indicates that it is transmitting. The grey timeslot is reserved for control messages. The schedule instructs each node on what to do in a timeslot: send to a neighbour, receive from a neighbour, or sleep. In IEEE 802.15.4 TSCH, the duration of a timeslot is usually set to 10 ms.

The 6TiSCH architecture supports both best-effort and deterministic traffic. The Routing Protocol for Low Power and Lossy Networks (RPL) is used for best-effort routing. Tracks, on the other hand, are used for deterministic traffic. A Track requires allocating physical resources like timeslots and buffers for nodes along the path, which is the challenge that we tackle in this work. The path computation and resource allocation for Tracks are performed centrally.

Because of these two routing options, the 6TiSCH architecture differentiates between soft and hard cells in TSCH schedules. Tracks reserve and use hard cells, while RPL uses soft cells for best-effort traffic. The 6TiSCH architecture supports multiple forwarding strategies, including GMPLS for Tracks. In GMPLS, a frame received at a particular timeslot can be switched into another timeslot at Layer 2 without regard to the upper layer protocol; this introduces less processing overhead than a Layer 3 forwarding scheme. DetNet is expected to enable end-to-end deterministic paths across heterogeneous networks, such as a 6TiSCH low power and lossy network and an Ethernet backbone.

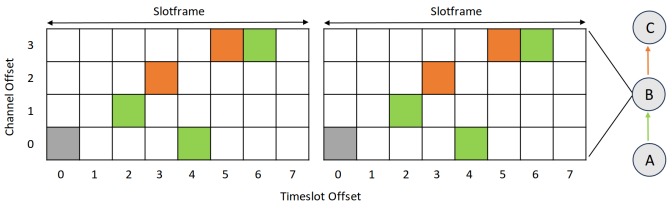


Fig. 1: A possible TSCH schedule for node B; time is cut into timeslots, which are organised into slotframes (see [7])

B. Related Work

1) Wireless Mesh Networks (WMN): Cross-layer optimisation is a topic with early roots in wireless ad hoc network literature [8].Capone et al. [9] studied a resource assignment problem that includes routing, scheduling, and channel assignment in WMNs. A column generation approach is adopted to solve the problem. In [10], the authors solved a similar problem with Integer Programming and Constraint Programming. These two papers solved extensive cross-layer problems but are not concerned with latency-constrained flows.

Cappanera et al. investigated the problem of joint routing and link scheduling in TDMA-based Wireless Mesh Networks for latency-constrained traffic [11]. The authors formulated the problem as a mixed integer-nonlinear problem. They also provided a fast heuristic based on Lagrangian decomposition.

2) Time-Sensitive Networking (TSN): Many recent publications on Time-Sensitive Networking have studied the JRaS problem. Although the concrete problem formulation differs from that of IEEE 802.15.4 networks, the TSN literature provides valuable insights for this work. Schweissguth et al. [6] proposed an Integer Linear Programming (ILP) model for the JRaS problem in TSN. The authors combined routing with a Time-Aware Shaper and compared it with a separate routing and scheduling formulation. In the SRaS problem, the authors evaluate two cost functions that promote either shortest path or load balancing. The two cost functions have drawbacks in schedulability or latency, while the JRaS approach offers a sweet spot between those two objectives.

Falk et al. [12] presented an ILP formulation that models the JRaS problem for flows of periodic real-time transmissions in TSN networks. Hellmanns et al. combined a Time-Aware Shaper with routing in [13]. The paper systematically compared the influence of different ILP formulations on the runtime of the JRaS problem. Vlk et al. [14] investigated the JRaS problem for TSN with time-triggered and periodic schedules. The problem is modelled with Constraint Programming and Benders Decomposition.

Krolikowski et al. [15] investigated JRaS in a large-scale IP radio access network. The authors used Cycle-Specified Queuing and Forwarding (CSQF) as defined by the IETF DetNet working group. The authors formulated the problem as a variant of the multi-commodity network flow problem and proposed a practical solution based on column generation and dynamic programming. Sharma et al. [16] investigated JRaS for *1+1* protected flows with CSQF and TSN. The problem was modelled as ILP, and compared against two heuristic approaches: greedy and tabu-search. These heuristics offer a trade-off between speed and number of accepted flows.

We are the first to introduce the JRaS problem to IEEE 802.15.4 networks in a DetNet setting. We experimentally show that our constrained shortest-path model is a suitable alternative to JRaS for many practical scenarios. Because our model is based on separate routing and scheduling, it is also very fast. We are not concerned with radio aspects like interference, which is left open for future work.

III. Problem Formulation

In this section, we give an overview of three different problem formulations: *Shortest Path Routing* (SP), *Constrained Shortest Path Routing* (CSP), and *Joint Routing and Scheduling* (JRaS). In SP and CSP, the routing and scheduling problems are solved separately. In JRaS, both problems are combined into a single, more complex problem which is then solved end-to-end. The different problems are modelled as a combination of Integer Linear Programming (ILP) and Constraint Programming (CP).

A. Routing

We model the routing problem as a variant of a multicommodity network flow problem as it is described, for example, in [17]. A network topology is described with the triple $(\mathcal{N}, \mathcal{A}, \mathcal{F})$. Nodes are denoted by $\mathcal{N}$, edges (or arcs) by $\mathcal{A}$, and $\mathcal{F}$ denotes the flows in the network. A flow $f \in \mathcal{F}$ contains n packets $p_{f,n} \in \mathcal{P}_f$ and is destined to a sink node. Furthermore, each flow f is restricted by $latency_f$, and all nodes share a common slotframe size sf. The variables for the routing problem are given by

$$\forall f \in \mathcal{F}, \forall (i,j) \in \mathcal{A}: \quad x_{f,(i,j)}\,.$$

The integer variables are in $\{0,1\}$. Next, we can formulate the *flow conservation constraint*. The constraint is responsible for routing a flow f between a source node s_f and a shared sink node z and is commonly used in network flow problems. In the constraint, we differentiate between three cases. First, we formulate the flow conservation at transit nodes

$$\forall f \in \mathcal{F}, \forall i \in \mathcal{N}: \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{f,(i,j)} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{f,(j,i)} = 0\,,$$

second, the flow conservation at the source nodes s_f

$$\forall f \in \mathcal{F}: \sum_{\{j|(s_f,j)\in\mathcal{A}\}} x_{f,(s_f,j)} - \sum_{\{j|(j,s_f)\in\mathcal{A}\}} x_{f,(j,s_f)} = 1\,,$$

and last, the flow conservation at the sink node z

$$\forall f \in \mathcal{F}: \sum_{\{j|(j,z)\in\mathcal{A}\}} x_{f,(j,z)} - \sum_{\{j|(z,j)\in\mathcal{A}\}} x_{f,(z,j)} = 1\,.$$

Each flow can contain multiple packets; however, we enforce that all packets in a flow follow the same path in the network. The flow conservation constraint works on the flow level and not on the packet level, and hence, only one path is selected. In the cost function, we minimise the combined path length of all flows. As the cost function minimises the path length, it also minimises overall transmissions, which increases the network's energy efficiency:

$$z(x) = \sum_{f\in\mathcal{F}} \sum_{(i,j)\in\mathcal{A}} x_{f,(i,j)}\,.$$

1) Constrained Shortest Path: The CSP model is an extension of the previous SP formulation. Besides the flow conservation constraint, we introduce two additional constraints. First, the *route-latency constraint* can be formulated with

$$\forall f \in \mathcal{F}: \quad \left(\sum_{(i,j)\in\mathcal{A}} x_{f,(i,j)}\right) + h_f \cdot (|P_f| - 1) \leq latency_f\,.$$

The constraint ensures that the flow's path length is shorter or equal to the flow latency. A flow's latency depends on when its last packet arrives at the sink node. Hence, we include the number of packets $|P_f|$ in the calculation. The constant $h_f \in \{1,2\}$ is chosen as follows: If the path between s_f and z requires only one hop, it is set to 1; otherwise, it is set to 2 because a node cannot simultaneously send and receive. The constraint does not give any latency guarantees since it is the scheduler's task to decide on the concrete transmission timeslot. However, a longer path is invalid in any case. Next, we introduce the *capacity constraint* with

$$\forall i \in \mathcal{N}:$$
$$\sum_{f\in\mathcal{F}} \left(\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{f,(i,j)} + \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{f,(j,i)} \right) \cdot |P_f| \leq sf\,.$$

The constraint counts the transmissions and receptions per node and ensures that they do not exceed the chosen slotframe size. This is equivalent to allocating bandwidth on the receiving and sending links of the node.

B. Scheduling

After route computation, a valid TSCH schedule must be found. We denote the routing solution for a particular flow with $\mathcal{S}_f$. A path segment $(i,j) \in \mathcal{S}_f$ denotes an active edge for flow f. A time variable is added to a node $i \in \mathcal{N}$ if it is part of a transmission or reception in the routing solution. This variable is integer-valued in $\{1, ..., sf + latency_f\}$ and will be denoted as:

$$\forall f \in \mathcal{F}, \forall p_{f,n} \in \mathcal{P}_f, \forall (i,j) \in \mathcal{S}_f: \quad t_{p_{f,n},(i,j)}\,.$$

The variable denotes the starting time of the packet transmission at node i and the reception time at node j. Hence, the same variable can be shared between nodes i and j. Furthermore, we assume that each transmission will take precisely one timeslot. We use a Constraint Programming formulation to formulate the first constraint; the resulting constraint is called *non-overlapping constraint*:

$$\forall i \in \mathcal{N}: \quad alldifferent(t_{p_{*,*,(i,*)}} \cup t_{p_{*,*,(*,i)}})\,.$$

The asterisk operators can be interpreted as follows: a set that contains all incoming/outgoing messages for a particular node for all packets in each flow. Because of the *alldifferent* global constraint, each transmission must start in a different timeslot; hence, we prevent overlapping transmissions. Next, we must ensure that the sequential transmissions of a particular packet are processed in the correct order. We call this *precedence constraint* and formulate it like

$$\forall f \in \mathcal{F}, \forall p_{f,n} \in \mathcal{P}_f, \forall (i,j) \in \mathcal{S}_f, (j,k) \in \mathcal{S}_f:$$
$$t_{p_{f,n},(j,k)} \geq t_{p_{f,n},(i,j)} + 1\,.$$

The incoming packet transmission (i,j) must occur before packet transmission (j,k); hence, node j must receive the packet from node i before transmitting it to node k. Furthermore, we want to ensure that the packets in a flow are sent in the correct order. We achieve this with the *order constraint*:

$$\forall f \in \mathcal{F}, \forall p_{f,n} \in \mathcal{P}_f, \forall (i,j) \in \mathcal{S}_f: \; t_{p_{f,n},(i,j)} < t_{p_{f,n+1},(i,j)}\,.$$

The *time-latency constraint* checks for latency violations:

$$\forall f \in \mathcal{F}: \quad t_{p_{f,l},(j,z)} - t_{p_{f,0},(s_f,i)} < latency_f\,.$$

Here, $p_{f,0,(s_f,i)}$ denotes the first transmission of the first packet, and $p_{f,l,(j,z)}$ denotes the sink arrival time of the last packet. The constraint ensures that the difference does not

exceed the flow latency. Note that the start time of each flow can be chosen freely, which seems to be a sensible choice for time-triggered traffic in a WSN environment.

Finally, we define the *phase shifting constraint*, which serves two purposes: First, it checks whether the number of transmissions and receptions do not exceed the maximum slotframe size. Second, it allows for phase shifting. The slotframes of all nodes have the same size; however, we do not require them to be in phase. So, on a global timescale, the execution time of the first timeslot can differ between nodes. Note that the timeslot boundaries between nodes are still aligned correctly. This simple extension allows for a higher flow admission rate and can be formulated as follows:

$$\forall i \in \mathcal{N}: \quad t_{max,i} - t_{min,i} < sf.$$

Here, the integer variable $t_{max,i}$ is equal to the maximum time variable for node i. Analogously, the integer variable $t_{min,i}$ equals the minimum time variable for node i. Additional constraints must be placed to set $t_{max,i}$ and $t_{min,i}$ to their correct values. The scheduling problem has no cost function; instead, we only check for feasible solutions.

C. Joint Routing and Scheduling

The JRaS formulation combines the routing and scheduling problem into a more complex problem formulation. Equivalent to the routing problem, we need variables that indicate if a flow takes a particular edge in the network:

$$\forall f \in \mathcal{F}, \forall (i,j) \in \mathcal{A}: \quad x_{f,(i,j)}\,.$$

The difference between the SP/CSP and JRaS formulation is that the variables are Boolean. Internally, the variables are still represented with integer values in $\{0, 1\}$. Hence, we can reuse the flow conservation constraint from Section III-A. The routing constraints from Section III-A1 are not required in the JRaS formulation.

In the independent problem formulation, there is a fixed routing solution, and exactly one scheduling variable is created for each transmission. This is not the case for the JRaS model. Because we solve for all decision variables at the same time, we create a variable for each possible transmission like

$$\forall f \in \mathcal{F}, \forall p_{f,n} \in \mathcal{P}_f, \forall (i,j) \in \mathcal{A}: \quad t_{p_{f,n},(i,j)}\,.$$

Hence, the number of time variables is increased immensely compared to the SRaS approaches. All scheduling constraints from Section III-B are used in the JRaS model. However, a constraint is only enforced if the associated routing variable is set to true. Hence, we add an enforcement literal to each scheduling constraint. The enforcement literal consists of one or multiple routing variables, and the conjunction of all these literals determines whether the constraint is active. First, we extend the non-overlapping constraint:

$$\forall f \in \mathcal{F}, \forall p_{f,n} \in \mathcal{P}_f, \forall (i,j) \in \mathcal{A}: \; x_{f,(i,j)} \implies t_{p_{f,n},(i,j)} > 0\,,$$

$$\forall f \in \mathcal{F}, \forall p_{f,n} \in \mathcal{P}_f, \forall (i,j) \in \mathcal{A}: \; \neg x_{f,(i,j)} \implies t_{p_{f,n},(i,j)} = 0\,,$$

$$\forall i \in \mathcal{N}: \; alldifferent\_except\_0(t_{p_{*,*},(i,*)} \cup t_{p_{*,*},(*,i)})\,.$$

The implication ensures that only active transmissions are scheduled. Transmissions not activated by a routing variable, i.e., $x_{f,(i,j)} = 0$, will be ignored in the *alldifferent_except_0* constraint. Note that this constraint can be implemented more straightforwardly with optional interval variables which are supported in CP-SAT. The precedence constraint is extended like

$$\begin{aligned}&\forall f \in \mathcal{F}, \forall p_{f,n} \in \mathcal{P}_f,\\ &\quad \forall j \in \mathcal{N}, \forall i \in \{i|(i,j) \in \mathcal{A}\}, \forall k \in \{k|(j,k) \in \mathcal{A}\}\\ &\qquad x_{f,(i,j)} \wedge x_{f,(j,k)} \implies t_{p_{f,n},(j,k)} \geq t_{p_{f,n},(i,j)} + 1\,.\end{aligned}$$

The order constraint, time-latency constraint, and phase shifting constraint from Section III-B are all modified analogously. The functionality of the constraints stays the same, but they are only activated if the associated routing variables are true. We omit the concrete implementation of the remaining constraints for readability.

The cost function is set to the combined shortest path length, which is the same as for the SP/CSP formulation. We also want to emphasise on the topic of *optimality*. Because CSP is less constrained in its routing selection compared to JRaS, CSP will always return a solution that is at least as good as the JRaS solution, ensuring the optimality of the CSP approach with respect to the combined shortest path length.

IV. Evaluation

A. Evaluation Setup

For the experimental evaluation, we use a dedicated PC equipped with an Intel(R) Core(TM) i7-13700K @ 3.40GHz (24 threads), 64GB of RAM, and Ubuntu 22.04.4 LTS as the Operating System. We solve the SP and CSP routing problems with Gurobi (version 11.0.1). The scheduling and JRaS problems are solved with the CP-SAT solver, which is included in Google's OR-Tools (version 9.9.3963). Furthermore, we use Python 3.10.12 for our implementation. In the experiments, the time limit of all solvers is set to 10 minutes.

B. Experimental Design

The experiments are conducted on *random grid topologies* of size 7x7; we keep the topology size the same for all experiments. For each topology, the connectivity level is randomly picked from $\{60\%, 70\%, \ldots, 100\%\}$. The value indicates how many links from all possible links are active, and the links are picked randomly. For example, a connectivity of 60% is a sparse network with almost no redundant paths. The network has one sink node which is picked randomly from the set of all nodes, and several random source nodes. Source nodes send a random number of packets to the sink node, and the number of packets per flow is chosen randomly from $\{1, \ldots, 8\}$. The latencies of the different flows are randomised as well. The minimum time required to reach the sink node depends on: The distance $d(s_f, z)$ between the source node s_f and the sink z, the number of packets $p_f \in \{1, \ldots, 8\}$ in that particular flow f, the hop constant $h_f = \min(d(s_f, z), 2)$, and the timeslot length, which is 10 ms:

$$latency\_min_f = 10\text{ms} \cdot (d(s_f, z) + h_f \cdot (p_f - 1)).$$

The latency per flow is then chosen randomly from $\{latency\_min_f, \ldots, \lfloor 1.5 \cdot latency\_min_f \rfloor\}$. Hence, some flows have strict latency requirements, while others allow for

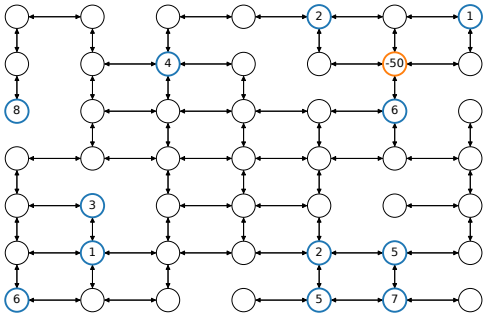


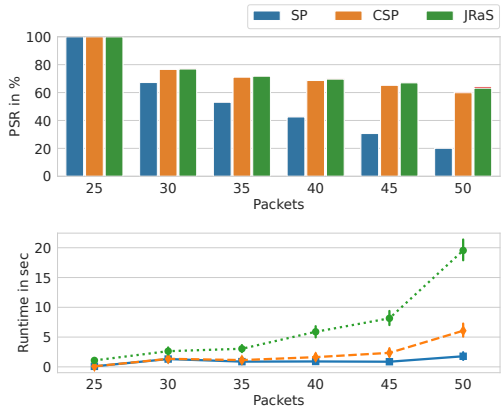
Fig. 2: An exemplary 7x7 network with a connectivity of 80% and 50 packets. The sink node is orange, and the source nodes are blue. The labels show the number of packets per flow.

short buffer times. In our experiments, all flows have the same periodicity, and hence, the slotframe size is equal to the flow period. Overall, many parameters vary on each run: Connectivity, link placement, packets per flow, latency per flow, as well as the position of the sink and source nodes. This randomisation allows us to explore a wide range of diverse scenarios, ensuring the thoroughness of our experiment. An illustrative problem instance is shown in Fig. 2.

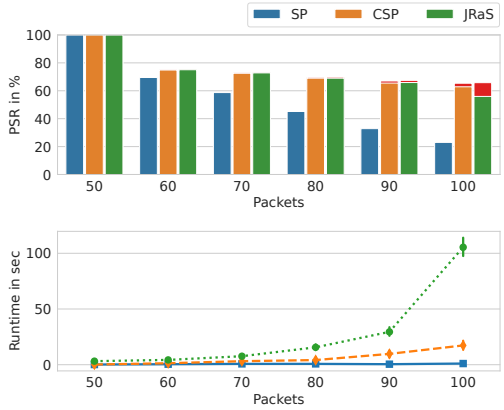
For all experiments, the slotframe size and the number of packets p send in the network are not randomised but carefully chosen. Note that the number of flows can vary because the number of packets per flow is randomised. We tested two different slotframe sizes with different p. The *first* configuration $\mathcal{C}_{50,p}$ has a flow period of 500 ms, which corresponds to 50 timeslots per slotframe, i.e., at most 50 packet transmissions per slotframe. Keeping the slotframe size constant, varying degrees of p are tested. As the number of packets changes the overall network utilisation, p is increased in small steps to gauge its impact on the network. Hence, $p = \{25, 30, \ldots, 50\}$. To better illustrate this setting: Let $p = 25$. This leads to a configuration $\mathcal{C}_{50,25}$ which allows 25 packets per period. As a result, the overall sink utilisation is 50% for this period. Compared to the first one, the *second* configuration $\mathcal{C}_{100,p}$ doubles the timeslots per slotframe to 100. This allows to evaluate whether increasing the number of packets for the same topology size influences the model performance or not. As the timeslots per slotframe are doubled, the number of packets p is increased: $p = \{50, 60, \ldots, 100\}$. Please note that this does *not* change the overall percentage of sink utilisation compared to the first configuration. Both configurations start with a sink utilisation of 50 %, which ramps up to 100 %.

C. Experimental Results

For $\mathcal{C}_{50,p}$, we carry out $8,000$ independent repetitions for each packet number p. Each problem instance is solved by our three model formulations (SP, CSP, JRaS). We track whether the problem instance could be solved successfully for each model. Here, a successful solution means all flows can be placed in the network without violating any latency requirements. We are interested in two metrics: problem



(a) Configuration $\mathcal{C}_{50,p}$: 50 timeslots per slotframe.



(b) Configuration $\mathcal{C}_{100,p}$: 100 timeslots per slotframe.

Fig. 3: The PSR and runtime results for the two configurations. The PSRs also show the number of timeouts in red.

solution ratio (PSR) and runtime. The PSR is the percentage of successfully solved problem instances for a packet level. The PSRs and runtime for $\mathcal{C}_{50,p}$ are shown in Fig. 3a.

After visually inspecting the PSRs in Fig. 3a, we can conclude that the PSRs of the *SP model* decrease as the network load increases. This model performs very well for a low network utilisation of 25 packets as it solves all problem instances. However, after increasing the number of packets to 30, the PSR drops to 67.17%. For the maximum utilisation of 50 packets, the PSR decreased significantly to a value of 20.06%. It almost appears that the PSRs decrease linearly as the network utilisation increases.

We observe that the PSRs of the *CSP and JRaS models* are significantly higher compared to the SP model from $\mathcal{C}_{50,30}$ and upwards. For 30 packets, the PSR is 76.57% for CSP and 76.85% for JRaS. 50 packets decrease the PSR to 59.86% (CSP) and 63.11% (JRaS). We carried out a statistical significance test (Wilson score with continuity correction [18]) to compare the difference in PSRs of CSP and JRaS. We are 95% confident that for $\mathcal{C}_{50,45}$, JRaS solves 0.30% to 3.25% more problem instances than CSP, and for $\mathcal{C}_{50,50}$, JRaS solves 1.73% to 4.76% more problem instances than CSP. There is no significant difference in the 95% confidence interval (CI)

for the other packet values. Hence, we conclude that CSP and JRaS perform equally well in many scenarios. However, when sink utilisation reaches 90% or higher, JRaS has a slight performance advantage of up to 4.76%.

Next, we compare the runtime of CSP and JRaS. The plot in Fig. 3a shows the mean runtime for the different packet values, and the error bar indicates the 95% CI. We can visually observe that the runtime of CSP is lower in all cases compared to JRaS. For 50 packets, both the runtime of CSP and JRaS increase significantly. Here, the mean runtime are 6.06 sec. for CSP but 19.54 sec. for JRaS, which is a speedup of factor $3.22\times$. Furthermore, JRaS suffers from timeouts in 1.17% of all runs in $\mathcal{C}_{50,50}$. For a decreased network utilisation of 45 packets, we can report a speedup of $3.46\times$. When we compare CSP to SP in $\mathcal{C}_{50,50}$, the runtime of CSP is significantly higher. After analysing the separate runtime of the routing and scheduling problem for CSP, we can report that the scheduling problem requires about 99.4% of the overall time. A likely reason for the SP speedup is that the scheduler can quickly sort out infeasible routing suggestions, which appear much more frequently for the SP model than for the CSP model.

We show the results of configuration $\mathcal{C}_{100,p}$, in which we increased the number of packets and the slotframe size in Fig. 3b. For the second configuration, we carry out $2,000$ independent repetitions for each p. If we compare the PSRs from $\mathcal{C}_{50,p}$ against $\mathcal{C}_{100,p}$, the performance of all models for all packet values appears to be similar. Because both configurations show a similar trend, the SP model tends to be sufficient when the network is underutilised (50%) but should be replaced with a more refined model like CSP or JRaS when sink utilisation reaches 60% or higher.

In the second configuration, there is no significant difference in the 95 % CI for any packet value when comparing CSP and JRaS. Hence, CSP and JRaS perform equally well. There might be two reasons for this behaviour. First, because we increased the number of packets to 100, there are more flows in the network, and hence, a wider variety of problems—problems that JRaS can solve but CSP cannot might appear less frequently. Second, because we increased the slotframe size to 100, the scheduler has more freedom in the time dimension, which might benefit CSP.

Regarding the runtime of CSP and JRaS, there are some differences. For $\mathcal{C}_{100,90}$ the speedup is $3.03\times$, and for $\mathcal{C}_{100,100}$ it increases to $6.09\times$. For $\mathcal{C}_{100,100}$, timeouts reach 2.55% for CSP and 9.85% for JRaS. In this complex scenario, CSP seems particularly useful as it has an equivalent PSR at a much lower runtime.

V. Conclusion

This paper compared the *problem solution ratio* (PSR) and the *runtime* of *SP*, *CSP*, and *JRaS*. The simple SP approach shows deficits in its PSR as network utilisation increases. The JRaS approach provides an upper bound on the PSR. The runtime is mostly acceptable but increases rapidly with the number of packets. CSP shows similar PSR values to JRaS but at a much lower runtime, i.e., we saw a speedup in the range of $3.03\times$ up to $6.09\times$. Hence, reserving bandwidth for latency-restricted flows appears to be a reliable method for the routing and scheduling problem.

Various aspects still need to be examined in future work. First, we want to increase the PSR of CSP even further. CSP is not complete, and hence, there are cases in which JRaS performs better regarding PSR. Analysing and comparing the routing behaviours of CSP and JRaS might give valuable insights in this direction. Second, we want to extend our research on dynamic problems including dynamic topologies, i.e., nodes and links can be added or fail, and varying flow demands. Here, fast models like CSP are important because they can keep network downtimes low in the case of unexpected network failures.

References

[1] Transforma Insights, "Current IoT Forecast Highlights," 2024, accessed: July 9, 2024. [Online]. Available: https://transformainsights.com/research/forecast/highlights

[2] "Ieee standard for low-rate wireless networks," *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pp. 1–800, 2020.

[3] P. Thubert, D. Cavalcanti, X. Vilajosana, C. Schmitt, and J. Farkas, "Reliable and Available Wireless Technologies," Internet-Draft draft-ietf-raw-technologies-08, Jul. 2023.

[4] F. Chen, T. Talanis, R. German, and F. Dressler, "Real-time enabled ieee 802.15. 4 sensor networks in industrial automation," in *International Symposium on Industrial Embedded Systems*. IEEE, 2009.

[5] P. Thubert, "An Architecture for IPv6 over the Time-Slotted Channel Hopping Mode of IEEE 802.15.4 (6TiSCH)," RFC 9030, May 2021.

[6] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," ser. RTNS '17, Oct. 2017.

[7] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "Ietf 6tisch: A tutorial," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, 2020.

[8] A. Goldsmith and S. Wicker, "Design challenges for energy-constrained ad hoc wireless networks," *IEEE Wireless Communications*, vol. 9, no. 4, Aug. 2002.

[9] A. Capone, G. Carello, I. Filippini, S. Gualandi, and F. Malucelli, "Routing, scheduling and channel assignment in Wireless Mesh Networks: Optimization models and algorithms," *Ad Hoc Networks*, vol. 8, no. 6, Aug. 2010.

[10] ——, "Solving a resource allocation problem in wireless mesh networks: A comparison between a CP-based and a classical column generation," *Networks*, vol. 55, no. 3, 2010.

[11] P. Cappanera, L. Lenzini, A. Lori, G. Stea, and G. Vaglini, "Optimal joint routing and link scheduling for real-time traffic in TDMA Wireless Mesh Networks," *Computer Networks*, vol. 57, no. 11, Aug. 2013.

[12] J. Falk, F. Dürr, and K. Rothermel, "Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP," ser. RTCSA '18', Aug. 2018.

[13] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen, "How to Optimize Joint Routing and Scheduling Models for TSN Using Integer Linear Programming," ser. RTNS '21, Jul. 2021.

[14] M. Vlk, Z. Hanzálek, and S. Tang, "Constraint programming approaches to joint routing and scheduling in time-sensitive networks," *Computers & Industrial Engineering*, vol. 157, Jul. 2021.

[15] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng, "Joint routing and scheduling for large-scale deterministic IP networks," *Computer Communications*, vol. 165, Jan. 2021.

[16] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet, "Routing and scheduling for *1+1* protected DetNet flows," *Computer Networks*, vol. 211, Jul. 2022.

[17] D. P. Bertsekas, "Network optimization : continuous and discrete models," 1998.

[18] R. G. Newcombe, "Interval estimation for the difference between independent proportions: comparison of eleven methods," *Statistics in Medicine*, vol. 17, no. 8, pp. 873–890, 1998.