Schedule Generation Using Knowledge Base for Contention-Free Medium Access in IoT Networks

Shashi Prabh
School of Engineering and Applied Science
Ahmedabad University
Ahmedabad, India
shashi.prabh@ahduni.edu.in

Jay Patel
School of Engineering and Applied Science
Ahmedabad University
Ahmedabad, India
jay.p7@ahduni.edu.in

***Abstract*—Timely delivery of data in safety-critical as well as real-time Internet of Things (IoT) applications is important. Contention-based medium access control (MAC) protocol stacks make channel access delay non-deterministic. Since both contention-based and contention-free MAC protocols are currently in use in various standards, a solution that is agnostic to protocol stack is explored in this paper. Finding a minimum makespan conflict-free schedule can be reduced to graph coloring problem which is known to be NP-complete. This problem becomes challenging in IoT networks since the devices are resource limited. This paper presents a method for short makespan conflict-free schedule search that can be implemented on resource limited IoT nodes with no dependency on a solver package. Experimental results from simulation and a proof-of-concept testbed are presented.**

***Index Terms*—Deterministic Networking, Quality of Service, Industrial IoT, Knowledge Base, Constraint Satisfaction**

I. Introduction

Timely delivery of data in safety-critical Internet of Things (IoT) applications is necessary for proper functioning of the system [1]. Achieving bounded delay is a major challenge in these networks due to the nodes having limited capabilities, heterogeneity and large scale. Carrier Sense Multiple Access with collision avoidance (CSMA/CA) is a contention-based distributed medium access control (MAC) protocol. While CSMA has been shown to be throughput optimal [2], it is not fair [3], [4]. This can lead to unbounded medium access delays which is highly undesirable. A conflict-free Time Division Multiple Access (TDMA) can provide deterministic channel access and thus bounded delays. Both CSMA-based and TDMA-based protocol stacks are currently used in IoT and Industrial Internet of Things (IIoT) standards. Though several MAC protocols designed to provide deterministic channel access have been proposed in the literature [5]–[8],their practical applicability had been limited. The approach explored in this work is to provide a conflict-free medium access abstraction to the application which is agnostic to the underlying protocol stack.

We consider the general scenario in an IoT network where a node transmits (sends or forwards) a mix of time-sensitive and best-effort data. A conflict-free medium access schedule is needed for the time-sensitive traffic. This schedule also needs to accommodate the best effort traffic requirements by maximizing the opportunities for the transmission of these packets. In this paper, we solve the problem of creating a conflict-free schedule for time-sensitive transmissions such that the makespan of the schedule is small. Note that minimizing the schedule makespan is reducible to edge coloring problem which is known to be NP-complete. Further, solver packages such as IBM CPlex and Google OR-Tools [9]–[11] have too large a memory footprint and resource requirements to be of practical use in a typical IoT/IIoT network. Since the search space gets very large for even moderate size networks [12], the main challenge is to find methods that can scale to networks of hundreds of nodes and can be run on low-cost limited-capability devices.

A conflict graph is defined by a set of vertices and edges where an edge connects a pair of vertices whose simultaneous transmissions can cause interference. The objective is to find schedules for each node such that any two nodes connected by an edge in the conflict graph representation of the IoT network do not share a common slot for transmitting time-sensitive traffic. Starting with the conflict graph representation and data rate requirements, a constraint satisfaction problem (CSP) formulation and a propositional knowledge base (KB) are constructed which are solved to obtain conflict-free schedule of short makespan. A KB approach can provide better integration with AI systems.

As discussed in Sec. III, directly solving the KB formulation is slow. First solving the constraint satisfaction problem (CSP) formulation and then adding the solution to the KB as constraints reduces the search space considerably, speeding up the inference in KB significantly. The CSP solution guarantees conflict-free transmissions and produces a *minimal* makespan schedule. Then with this solution added to the KB as a constraint, the inference then produces augmented schedules where slack within the time-sensitive transmission schedule is identified and allocated to the nodes as extra slots.The KB inference ensures that these new allocations do not conflict with existing schedule. These extra slot allocations along-with the unscheduled part of the frame can then be used for transmitting the best-effort traffic.

This paper is organized as follows. We present system model in Sec. II. Conflict-free scheduling approach is de-

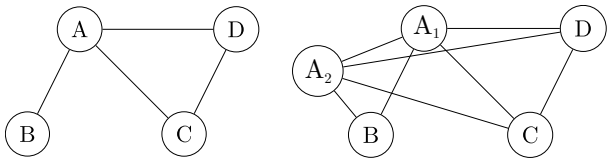
scribed in Sec. III. Details of a proof-of-concept implementation and experimental results is presented in Sec. IV. Related work is presented in Sec. V.

II. System Model

Consider a network where each node transmits (sends or forwards) a mix of time-sensitive and best-effort data. We divide the unit time interval into T timeslots where the slot duration is large enough to transmit the largest frame in a single slot plus the upper bound on operating system and channel access delays, a part of which is non-deterministic. Further, suppose that each node $v \in \boldsymbol{V}$ requires $n_v \leq T$ slots per time unit for transmitting the time-sensitive data where $\boldsymbol{V}$ denotes the set of nodes. Our objective is to construct a conflict-free schedule for all the nodes such that the makespan of the schedule for time-sensitive transmissions is minimized. This ensures that all the time-sensitive traffic requirements are met and the number of remaining slots (out of T) used for the best effort traffic is maximized. In addition to minimizing the makespan, any slack within the time-sensitive transmission schedule is also made available to the nodes for increasing the best-effort throughput, which is referred to as *schedule augmentation*. This schedule augmentation is done in a such manner that the schedule remains conflict-free.

In a conflict-graph $G(\boldsymbol{V}, \boldsymbol{E})$ representation of a network, an edge $(v_i, v_j) \in \boldsymbol{E}$ exits if simultaneous transmissions of v_i and v_j can interfere (Fig. 1a). Thus an edge in G represents the constraint that at-most one of the two nodes that the edge connects can be active at any given time. Nodes may have non-homogeneous time-sensitive data-rate requirement. For example, a temperate sensing device on a motor might be configured to send sensor readings at a higher rate than the one that monitors room temperature. Or, a node might be routing such traffic from several sources. Consider the rate requirements in the form of number of slots. The non-homogeneous rate demand is handled by transforming the initial conflict graph to a conflict graph with homogeneous rate demand as follows. Suppose a node v requires $n_v > 1$ slots per frame, i.e., per T timeslots. The transformation is done by replicating the node v n_v times, joining all the node replicas with the same set of neighbors, and then fully connecting all the replicas of n_v. In the example shown in Fig. 1a, suppose A requires 2 slots and all the other nodes require 1 slot. Then the conflict graph is transformed by replicating A twice as A_1, A_2 and connecting these by edges to the neighbors of A in the base conflict graph. Finally, A_1 and A_2 are connected by an edge (Fig. 1b). Thus in the transformed conflict graph all the nodes have homogeneous time-sensitive data rate requirement. In the rest of the paper, we will consider the transformed conflict graph.

An edge device or a network coordinator is assumed to compute and disseminate the schedules. It is assumed that nodes perform neighborhood discovery periodically. A number of energy-efficient neighborhood discovery methods have been reported in the literature [13]. The neighborhood list is assumed to be stored and maintained by the edge device. Nodes also communicate their data rate requirement to the edge device. The conflict graph is inferred on the basis of neighborhood and bandwidth requirements, as discussed previously. We also assume that the clocks of all the nodes are synchronized to a common time source. An application layer program initiates transmissions at the beginning of its assigned slots.



(a) Conflict graph (b) Transformed conflict graph

Fig. 1: Transformation to homogeneous rate requirement conflict graph

III. Knowledge-Based Scheduling Approach

Schedule search using propositional KB is described in this section. Inference in a propositional KB is known to be co-NP-complete. In order to reduce the complexity of KB inference, we first solve a constraint satisfaction problem (CSP) formulation using a fast search algorithm and then add the CSP solution to the KB as a new constraint. Seeding the inference reduces the search space considerably which relaxes the need to use high-end compute platforms for solving large instances. In our experiments, without adding the CSP solution to the KB, Davis-Putnam-Logemann-Loveland (DPLL) algorithm could not terminate on modest 25 node instances even after running for six hours.

A. CSP Formulation

Let $\boldsymbol{A} \in \{0,1\}^{N \times N}$ be the adjacency matrix representation of the homogeneous conflict graph at some given time. In other words, $A_{ij} = 1$ if i and j are connected by an edge in the conflict graph and 0 otherwise. Based on $\boldsymbol{A}$, define a CSP where $\boldsymbol{X} = \{X_i\}, i = 1, 2, \ldots, N$ is the set of variables and $\boldsymbol{D} = 1, 2, \ldots, N$ is the set of domains. Each node in the conflict graph is represented by one X_i. The assignment $X_i = D_k$ means that the node represented by X_i can transmit in slot k. The set of constraints $\boldsymbol{C}$ is defined as $\boldsymbol{C} = \{X_i \neq X_j\}$ for all (i, j) where $A_{ij} = 1$. In other words, any node pair connected by edge in conflict graph is not scheduled simultaneously. The solution of this CSP forms a part of the KB as discussed later.

Backtracking search is commonly used to solve a CSP [14]. Several heuristics have been studied to improve the performance of backtracking search [12]. We experimented with two modifications, referred here as `Backtracking-Search` and `Backtracking-Search-AC3`. The `Backtracking-Search` algorithm selects the first

TABLE I: Heuristics used in CSP search

	`Backtracking Search`	`Backtracking Search AC-3`
Next unassigned variable selection	First unassigned variable found	Variable with smallest number of legal values
Next domain value selection	No particular order	Value with the smallest number of conflicts
Consistency check	Not used	`AC-3`

unassigned variable for assignment and assigns it the domain value in no particular order. On the other hand, `Backtracking-Search-AC3` uses the variable with smallest number of legal values and assigns it the value with the smallest number of conflicts. A major difference between the two is that `Backtracking-Search-AC3` uses `AC-3` algorithm for consistency check. `AC-3` has worst case time complexity of $O(e|\boldsymbol{D}|^3)$ where e is the number of edges in the conflict graph which makes `Backtracking-Search-AC3` slower. A comparison of the two is summarized in Table I. `AC-4`, an optimized version of `AC-3`, has better worst case time complexity of $O(e|\boldsymbol{D}|^2)$, but `AC-3` has been found to be superior [15]. Since the domain values are first assigned in order, both produce minimal makespan schedules.

An evaluation of the two backtracking search algorithms was done on instances of varying sizes of conflict graphs where node locations were distributed uniformly randomly in a rectangular area. Interference range was defined and nodes separated by distance within the range were connected by edge. For the larger instances, spatial reuse was also taken into account and the average node degree was kept near 20. The implementation was done in `Python` language and tests were run on a PC with 3.4+ GHz 12^{th} generation Intel Core i7-1260 processor and 16 GB RAM.

The makespan of the time-sensitive transmission schedule returned by both the search methods are comparable, as shown in Fig. 2. Each data point represents the average over a sample of 100 instances of randomly generated networks. As shown in the figure, the makespans of the CSP schedules were close to the average node degree. Due to spatial reuse, the makespan grows sublinearly and tends to flatten out. The search time of `Backtracking-Search` was smaller by at-least one order of magnitude than that of `Backtracking-Search-AC3`. For the case of 250 nodes, `Backtracking-Search` took 29 ms but `Backtracking-Search-AC3` took 350 ms. Further, the search time of `Backtracking-Search-AC3` grows superlinearly with node count due to its dependence on the number of edges.

B. KB Formulation

In the CSP schedule, each node in the transformed conflict graph is scheduled once and only once. Solution of the toy example given in Fig. 1a returned by the CSP search is shown in Fig. 3a (assume that all the four nodes require 1 slot per frame for time-sensitive data). This schedule can be further augmented by scheduling B in the last slot along

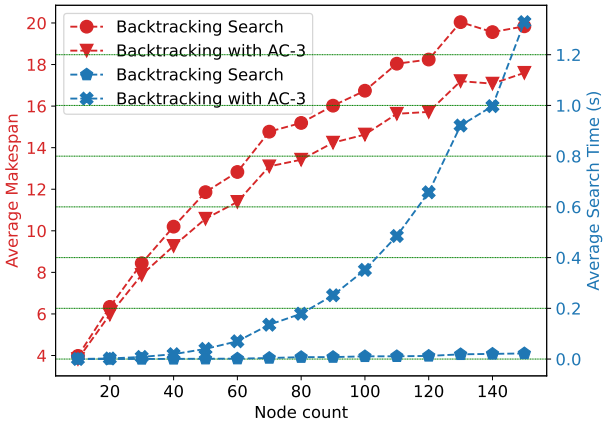
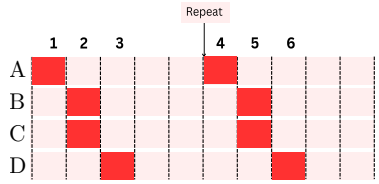
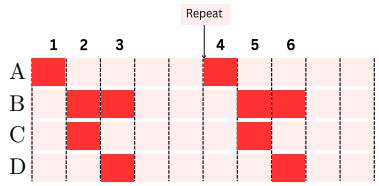


Fig. 2: Schedule makespan and search time

with D (Fig. 3b). The model search over the propositional KB described below performs the schedule augmentation while keeping it conflict-free.



(a) CSP solution: conflict-free schedule



(b) KB solution: Augmented CSP schedule

Fig. 3: Schedule augmentation

Suppose the makespan of the schedule returned by CSP search is $M \leq N$. We construct a propositional KB as follows. Symbols

$$S_{m,n}, m = 1, \ldots, M; n = 1, \ldots, N \tag{1}$$

indicate whether node n is scheduled during slot m. Thus, $S_{m,n}$ is true if node n is scheduled during slot m. Rules of the from

$$\neg(S_{m,n} \wedge S_{m,n'}), m = 1, \ldots, M \tag{2}$$

are added to the KB for all node pairs (n, n') connected by an edge in the conflict graph. These rules correspond to the conflict-free schedule requirement.

Non-homogeneous data-rate requirement was handled by conflict graph node replication earlier. Accordingly, rules of the form

$$S_{1,n} \vee S_{2,n} \vee \ldots \vee S_{m,n}, n = 1, \ldots, N \tag{3}$$

are added. These rules correspond to the requirement that each node must be scheduled at-least once.

Finally, we add the CSP output as a rule to seed the search:

$$S_{m_1,1} \wedge S_{m_2,2} \wedge \ldots \wedge S_{m_N,N}, m_i \in \{1, \ldots, M\}, \quad (4)$$

where m_v is the slot in which node v is scheduled. Adding this rule helps in reducing the search space and also makes the rule (3) redundant. Moreover, it *decouples* the model search for one time slot from another, thus further reducing the search space considerably. Giving slot-wise instances to the DPLL algorithm one by one produces all the valid models for the particular slot. Out of all the models it returns, the model that schedules the largest number of nodes in a particular slot is selected. Instead of maximizing the utilization per timeslot, fairness or some other metric could also be used.

In the experiments, schedule augmentation improved the average number of slots per node by over 50% from the baseline CSP solution of exactly 1 slot per node. Using the setup described earlier for evaluating the two CSP solvers, a KB was constructed for each network instance, the two CSP solutions were added to the KB one at a time and solved separately for comparison. The results are shown in Fig. 4. The improvement to the `Backtracking-Search` schedule is a bit higher because the makespans were larger.

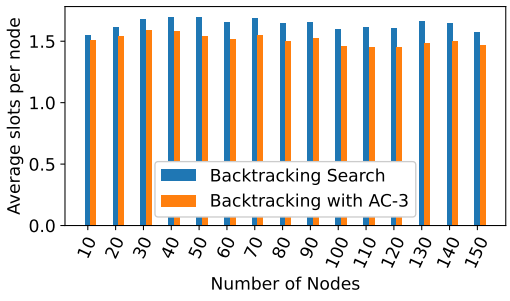


Fig. 4: Scheduled throughput after solving the KB

C. Evaluation on Raspberry Pi

We also evaluated the implementation on a Raspberry Pi 5 board. The board had 2.4 GHz quad-core 64-bit Arm Cortex-A76 CPU and 4GB RAM [16]. A comparison of the average makespan of schedule and KB inference time with those obtained on a PC whose specification was described earlier shows that this method can be used on low-end devices (Fig. 5). This figure shows the results of KB inference where `Backtracking-Search-AC3` was used for solving the CSP formulation. Each data point is an average over 100 randomly generated network instances. The average time taken by the KB inference procedure (including the CSP search phase) to produce a set of schedules on a network of 100 nodes was approximately 95 s. We gave a 25 node network instance to the KB inference without the CSP solution added as constraint and ran the implementation on the PC. The search did not terminate even after running for 6 hours. On the other hand, schedules for instances of

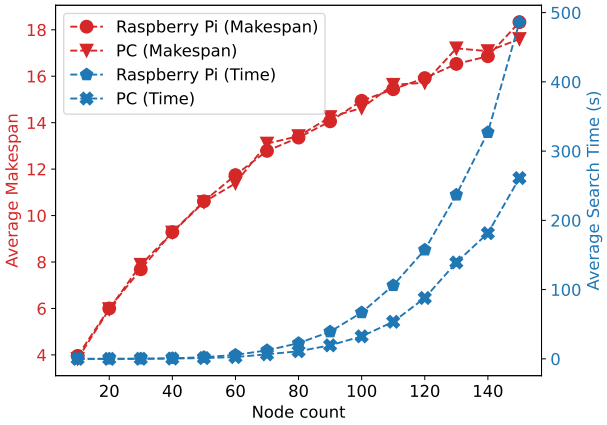


Fig. 5: Comparison of makespan and KB inference time (including the CSP solution time) on Raspberry Pi and PC

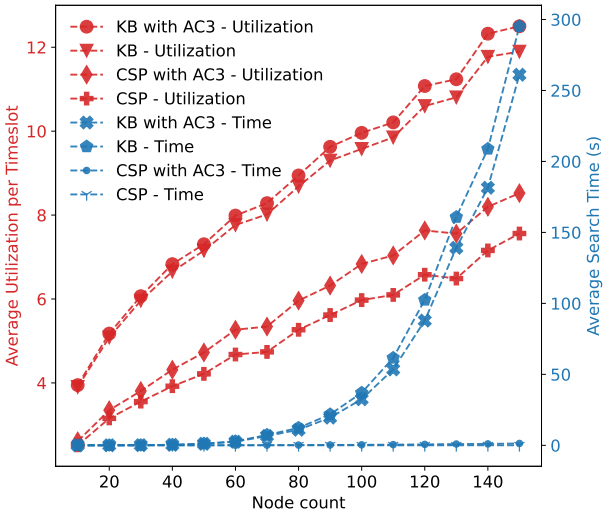


Fig. 6: Average per timeslot utilization of CSP and KB solutions

over 100 node networks could be produced on Raspberry Pi board in a short time using the method described above. We note that this inference time can further be reduced by applying suitable early stopping criteria.

D. Discussion

A comparison of average per timeslot utilization and search time of CSP and KB solutions is shown in Fig. 6, where the utilization in a given timeslot is defined as the number of nodes scheduled in that timeslot. The choice of the most suitable method depends on the timescale of the system and capabilities of the edge device or the coordinator. The utilization of the KB schedules was about 1.5 times that of the CSP schedules. The DPLL inference time, however, dominates the CSP search time for the two variants of backtracking search by about 2-3 orders of magnitude. However, KB inference when seeded with the schedule given by `Backtracking-Search-AC3` produces shorter makespan schedules with higher per

TABLE II: Schedule search time on PC and Raspberry Pi

	CSP	CSP & AC-3	KB	KB & AC-3
PC, 50 nodes	1.7 ms	52 ms	1.31 s	1.23 s
RPi, 50 nodes	3.32 ms	123.22 ms	2.56 s	2.37 s
PC, 100 nodes	10.7 ms	0.35 s	37.01 s	32.49 s
RPi, 100 nodes	20.29 ms	1.16 s	93 s	66.90 s

slot utilization in lesser amount of time (Table II). The smaller makespan schedule results in faster search for augmented schedule. On the other hand, the makespan of the time-sensitive transmission schedule returned by both the CSP solution methods `Backtracking-Search` and `Backtracking-Search-AC3` are comparable (Fig. 2) but `Backtracking-Search` is faster by one to two orders of magnitude. Thus in short timescale situations, a good strategy would be to set a search time limit, obtain CSP schedule using `Backtracking-Search`, perform KB inference terminating the search at the set time limit and use the partially augmented schedule. When the timescale is large enough, using `Backtracking-Search-AC3` for the CSP schedule followed by KB inference performs better.

IV. Proof-of-Concept Implementation

To evaluate MAC agnostic scheduling, we implemented a proof-of-concept system on a testbed consisting of Raspberry Pi Pico W boards [17] as shown in Fig. 7. These boards have 133 MHz dual-core Arm Cortex-M0+ processor, 264kB on-chip SRAM, 2MB RAM and 2.4GHz 802.11n for wireless connectivity. The Pico nodes received schedule from a PC and were synchronized using NTP. The application was implemented in `MicroPython`.

Fig. 7: Experimental setup

The setup consisted of 9 Pico W nodes. All the nodes were in the same broadcast domain, connected to the same 802.11 router (not shown in the figure). Each node, upon synchronization using NTP, broadcasted 225 Byte messages during its assigned slots. The payload included sequence number, timestamp, node identifier (IPv4 address) and a random string. Broadcasts from all the nodes were logged at a PC for analysis. In this test, the slot duration was 0.1 s and the frame duration was 0.9 s. Starting with one of the nine slots, each node was assigned every 9^{th} slot. Thus, each node attempted one transmission every 0.9 s.

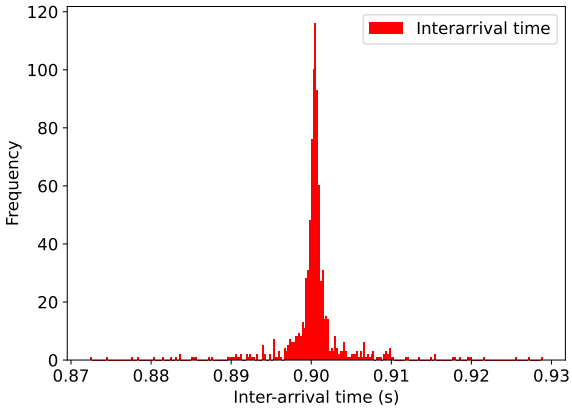


Fig. 8: The distribution of inter-arrival time

Since the timestamps were generated at the application layer, we consider inter-arrival delays. Fig. 8 shows the histogram of 900 packet inter-arrival times at the PC from the Pico nodes. The mean inter-arrival time from a given source node was 0.9 s and standard deviation was 0.0043 s. The minimum and maximum inter-arrival times across all source nodes were 0.8723 and 0.9289 s, respectively. This corresponds to a maximum jitter equal to 29 ms. The jitter is primarily due to non-deterministic channel access and operating system delays. Zero packet loss was observed. Thus, protocol stack agnostic conflict-free scheduling at the application layer can achieve deterministic delays except for the jitter due to the underlying system and protocol stack delays, which must be taken into consideration for determining the timeslot duration.

V. Related Work

Minimum makespan scheduling problem has been shown to be intractable [18], [19]. Several MAC protocol designs for providing bounded delay in wireless sensor networks have been reported in the literature. TRAMA [20] is a TDMA-based MAC protocol that is designed to provide collision-free transmissions. ZMAC [6] is another MAC protocol for sensor networks. It is a hybrid protocol that operates in CSMA mode under light load conditions and in TDMA under heavy load conditions. A logical regular topology backbone based design for bounded end-to-end delay scheduling in sensor networks is presented in [21]. The logical regular topology enables low-complexity scheduling policies. A survey of previous work on reducing latency in the Internet is presented in [22]. However, reducing delay is not the same as providing worst-case delay guarantee.

Time Sensitive Networking (TSN) is a set of IEEE standards designed to provide deterministic services over Ethernet [23]. The mechanism for bounding end-to-end latency in TSN involves synchronized nodes, centralized network coordinator, scheduled flows, separating time-sensitive traffic

from best-effort traffic, frame preemption and policing time-sensitive traffics. TSNsched [24] is a tool which generates schedules for TSN. It takes as input a network topology, including the network flows, and network performance requirements, including values for the maximum latency and jitter per flow. Extending TSN to wireless networks is of current research interest [25], [26]. WiFi TSN is a system designed to provide deterministic delay in wireless networks [27]. The design of WiFi TSN offers integration of IEEE 802.11 with the wired TSN standard. Widespread use of TSN in IoT/IIoT networks in near future is unlikely primarily because of its complexity.

VI. Conclusion

Conflict-free scheduling enables bounding delay which is needed in safety-critical and real-time IoT/IIoT applications. This paper presented a protocol stack agnostic approach for conflict-free scheduling of the time-sensitive data. Since the problem is intractable, the resource limitations in IoT/IIoT devices pose a major challenge. We discussed how combining the result of backtracking search on a CSP formulation to a KB formulation and then using DPLL to solve it can produce conflict-free schedules on resource-limited devices. It turns out that the fast backtracking search alone can produce good (short makespan) schedules and is especially useful for dealing with very large instances. Results of experimental evaluations presented here support the effectiveness of the proposed method. A trade-off for using the protocol agnostic approach is that the slot duration needs to accommodate the overhead of the operating system and protocol stack delays, in addition to the data transmission time. A margin of about $30\,\mathrm{ms}$ was found to be sufficient for Pi Pico nodes that use IEEE 802.11. This is not necessarily a limitation in a typical IoT/IIoT scenario where the data rate is low. Further reducing the KB inference complexity and making the method distributed are interesting future directions.

Acknowledgment

The comments of anonymous reviewers helped in improving the quality of the paper. We are grateful for their time. This work was supported by Ahmedabad University grant number URBSEASE20A6/SUG/20-21/03-SP-08.23.

References

[1] H. Kopetz and W. Steiner, "Internet of Things," in *Real-time systems: design principles for distributed embedded applications*. Springer, 2022, pp. 325–341.

[2] L. Jiang and J. Walrand, "A distributed CSMA algorithm for throughput and utility maximization in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 18, no. 3, pp. 960–972, 2009.

[3] Y. Jian and S. Chen, "Can CSMA/CA networks be made fair?" in *Proc. of the 14th ACM Intl. Conf. on Mobile Computing and Networking*, ser. MobiCom '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 235–246. [Online]. Available: https://doi.org/10.1145/1409944.1409972

[4] T. Tanaka, S. Prabh, and Y. Liu, "Throughput equalization in mean-field hard-core models for CSMA-based wireless networks," in *14th Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2016, pp. 1–8.

[5] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks," in *SenSys '03: Proc. of the 1st Intl. Conf. on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2003, pp. 181–192.

[6] I. Rhee, A. Warrier, M. Aia, and J. Min, "Z-MAC: a hybrid MAC for wireless sensor networks," in *SenSys '05: Proc. of the 3rd Intl. Conf. on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2005, pp. 90–101.

[7] G.-S. Ahn, S. G. Hong, E. Miluzzo, A. T. Campbell, and F. Cuomo, "Funneling-MAC: a localized, sink-oriented MAC for boosting fidelity in sensor networks," in *SenSys '06: Proc. of the 4th Intl. Conf. on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2006, pp. 293–306.

[8] S. Garg, V. S. S. Kuchipudi, E. S. Bentley, and S. Kumar, "A real-time, distributed, directional TDMA MAC protocol for QoS-aware communication in multi-hop wireless networks," *IEEE Access*, vol. 9, pp. 26 343–26 361, 2021.

[9] "IBM Decision Optimization Package CPLEX." [Online]. Available: http://ibmdecisionoptimization.github.io/docplex-doc/index.html

[10] "AI Space." [Online]. Available: http://aispace.org

[11] "Google OR-Tools." [Online]. Available: https://developers.google.com/optimization

[12] J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt, M. Sterna, and J. Weglarz, *Handbook on scheduling: From theory to practice*. Springer, 2019.

[13] W. Sun, Z. Yang, X. Zhang, and Y. Liu, "Energy-efficient neighbor discovery in mobile ad hoc and wireless sensor networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1448–1459, 2014.

[14] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. USA: Pearson, 2020.

[15] R. J. Wallace, "Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs," in *Proc. of the 13th Intl. Joint Conf. on Artifical Intelligence - Volume 1*, ser. IJCAI'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, p. 239–245.

[16] "Raspberry Pi 5 Specifications." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/

[17] "Raspberry Pi Pico W Specifications." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-pico

[18] L. J. Stockmeyer and V. V. Vazirani, "NP-Completeness of some generalizations of the maximum matching problem," *Inf. Process. Lett.*, vol. 15, no. 1, pp. 14–19, 1982.

[19] G. Sharma, R. R. Mazumdar, and N. B. Shroff, "On the complexity of scheduling in wireless networks," in *Proc. of the 12th Intl. Conf. on Mobile computing and networking*. IEEE, 2006, pp. 227–238.

[20] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks," in *SenSys '03: Proc. of the 1st Intl. Conf. on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2003, pp. 181–192.

[21] K. S. Prabh and T. Abdelzaher, "On scheduling and real-time capacity of hexagonal wireless sensor networks," pp. 136–145, 2007.

[22] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing Internet latency: A survey of techniques and their merits," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2149–2196, third quarter 2016.

[23] "IEEE 802.1 Time-Sensitive Networking (TSN) Task Group." [Online]. Available: https://1.ieee802.org/tsn/

[24] A. C. T. dos Santos, B. Schneider, and V. Nigam, "TSNSCHED: automated schedule generation for time sensitive networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2019, pp. 69–77.

[25] A. Mildner, "Time sensitive networking for wireless networks-a state of the art analysis," *Network*, vol. 33, pp. 907–922, 2019.

[26] L. Lo Bello and W. Steiner, "A perspective on IEEE Time-Sensitive Networking for industrial communication and automation systems," *Proc. of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[27] D. Cavalcanti, C. Cordeiro, M. Smith, and A. Regev, "WiFi TSN: Enabling deterministic wireless connectivity over 802.11," *IEEE Communications Standards Magazine*, vol. 6, no. 4, pp. 22–29, 2022.