Schedulability Analysis of WSAN Applications: Outperformance of A Model Checking Approach

Ehsan Khamespanah*, Morteza Mohaqeqi†, Mohammad Ashjaei‡, Marjan Sirjani‡
*University of Tehran, Tehran, Iran, e.khamespanah@ut.ac.ir
†Uppsala University, Uppsala, Sweden, morteza.mohaqeqi@it.uu.se
‡Mälardalen University, Västerås, Sweden, {mohammad.ashjaei,marjan.sirjani}@mdu.se

***Abstract*—Wireless sensor and actuator networks (WSAN) are real-time systems which demand timing requirements. To ensure this level of requirements, different timing analysis approaches have been proposed for WSAN systems. Among different alternatives, analytical analysis and model checking approaches are two common ones which are widely used for the timing analysis of WSAN systems. Analytical approaches apply worst-case response time analysis techniques, whereas model checking generates explicit states of models to analyze them. In this paper, we develop schedulability analysis techniques based on two approaches, i.e., analytical and model checking approaches. We apply and compare the proposed analysis approaches on WSAN systems with an application in monitoring and control of civil infrastructures implemented on the Imote2 wireless sensor platform. We show that the highest possible data acquisition frequency for this application is computed while meeting the deadlines, and compare the results of the two approaches in terms of scalability, extensibility, and flexibility.**

I. Introduction

Wireless sensor and actuator networks (WSANs) provide low-cost continuous monitoring as the infrastructure for control systems. Thanks to the use of wireless communications and distributed architectures, WSANs encompass many advantages compared to traditional hardwired networked control systems. In the design of control systems that benefit from WSANs, interdependencies between a control algorithm and communication have to be considered. For example, while low sampling rate for sensor data acquisition usually degrades control performance, high sampling rate may increase resource contention in bandwidth-constrained WSANs, which in turn may lead to degrading the control performance. This makes the design of WSAN systems more complicated and it is necessary to develop techniques and methods that facilitate design and verification of WSAN systems.

Nevertheless, trial and error is a widely used approach for the WSAN applications design. For example, the work in [1] presents an empirical test-and-measure approach based on binary search to find configuration parameters, including worst-case task runtimes and the communication time-slot. Extending real-time scheduling theory [2], that has been developed for the timing analysis of real-time systems, is another approach for the timing verification of WSAN systems. However, due to the use of event-driven operating systems in WSANs with very limited scheduling supports, e.g., TinyOS [3], many schedulability analysis approaches cannot be effectively employed.

A. Related analysis approaches

A set of techniques have been proposed to provide schedulability analysis, including utilization-based tests [4], response-time analysis [5], and real-time calculus [6] as well as model checking approaches [7], [8]. Considering the analytical approaches, several works in the literature addressed the schedulability analysis of WSANs considering different settings and requirements. The work in [9] defined three different task scheduling techniques, then presented an analytical method to find the maximum schedulable load. The work in [10] and [11] presented analytical analysis; in [10] the target is the earliest deadline first scheduling algorithm, and in [11] the target is the high energy first scheduling algorithm.

Among the works that analyzed real-time systems using model checking approaches, Petri Nets are used to check the behavior of task executions in [12], [13]. In the context of uniprocessor real-time systems, the work in [14] developed a timed automata model for schedulability analysis of preemptable tasks that have interactions with each other through synchronization methods. A technique in [15] is proposed to tackle similar problem in analyzing tasks executing synchronously. The work in [16] extended the reachability analysis on timed automata to consider precedence and resource constraints in uniprocessor real-time systems. In the domain of multiprocessor systems, the work in [17] proposed a schedulability analysis based on model checking for multiprocessor real-time systems. The work in [18] performed exact schedulability analysis of multi-core real-time systems using model checking.

B. Motivation and contributions

To the best of our knowledge, although each of the above mentioned techniques widely used in the analysis of WSAN systems, it is not clear which of them has to be used for the analysis of a given WSAN system. In other words, there is no work on comparing analytical and model checking approaches to make their strengths and weaknesses clear. The main reason is that the techniques are normally optimized for specific applications, which makes it difficult to globally compare the techniques.

In this paper, we develop an analytical approach and a model checking approach and we compare them in the analysis of a WSAN framework. The WSAN framework in this paper is proposed for developing structural health monitoring and control applications [1]. This framework has been implemented on the Imote2 wireless sensor platform and used in several long-term developments of highway and railroad bridges. Note that applications of this framework can be extended to a wide range of monitoring purposes as it has a flexible and open architecture [19].

The results of our study show that the analytical analysis for the application described in this paper, which is based on conservative worst-case assumptions and empirical measurements, may lead to schedules that do not utilize resources in the most efficient way. By using model checking we can get better utilization of resources. However, model checking is more sensitive to the values which are set for timings of the model and different values may result in different memory and time consumption for model checking of the models. Note that having different values for timings do not affect the analysis cost of the analytical approach. We also compare the extensibility and flexibility of the two approaches by checking how each approach can be reused for different network communication protocols and different characteristics of tasks.

II. Application Model and Design Objectives

WSANs provide a new communication technology for a wide range of cyber-physical applications. A WSAN involves feedback control loops between sensors and actuators through a wireless network. Sensors of WSANs measure process variables and deliver the set of gathered data to a controller through the network. The controller sends control commands to the actuators, which then operate the control and safety components. Structural health monitoring and control (SHMC) of civil infrastructure [1] is an instance of WSANs, used in several long-term development of several highway and railroad bridges [19]. SHMC application development has proven to be particularly challenging as it has the complexity of a large-scale distributed system with real-time requirements, while having the resource limitations of low-power embedded WSAN platforms. Sensing and control in these systems need to meet stringent real-time performance requirements on communication latency in harsh environments. Violation of these requirements may result system failure or significant costs.

A WSAN application is a distributed system with multiple sensor nodes, each comprised of the independent concurrent entities. It contains a CPU and several sensors which bridged together via a wireless communication device that uses a transmission control protocol. Interactions between entities, both within a node and among nodes, are concurrent and asynchronous. Moreover, WSAN applications are sensitive to the timing of events, with soft deadlines at each step of the process that are required to ensure correct and efficient operation. Due to the performance requirements, coordination among sensing, data processing, and communication activities is required in WSANs. In particular, once a sample is acquired from a sensor, its corresponding radio transmission activities must be performed. At the same time, data processing tasks must be executed. For example, because of the environmental changes in the temperature, a kind of data compensation must be applied on sensor data to adjust the acquired values. Moreover, the timing of radio transmissions from different nodes must be coordinated using a wireless communication protocol.

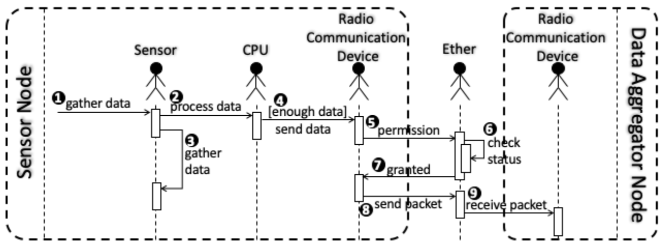

Figure 1. The sequence diagram of the main scenario of a WSAN for Structural Health Monitoring and Control

A. The System Model of a WSAN for Structural Health Monitoring and Control

As shown in Figure 1, we use a UML sequence diagram to present how WSAN applications work. To this end, we need to have a look into the interaction of the components and events which are triggered and serviced by them. Based on the specification of the WSAN applications, there are many nodes which have the role of data acquisition and data transmission. Note that all of the message calls in this model are asynchronous, thus they are shown by open arrows in Figure 1 and they do not have any return value.

For data acquisition, a node has a set of sensors which periodically acquire data from the environment (messages 1 and 3 in Figure 1) and send the data to the processing unit of the node (message 2 in Figure 1). The processing unit is responsible for validating the data and storing it into an internal buffer. Upon receiving enough data (the guard on message 4 in Figure 1), the processor unit puts the data in one packet and sends it to the radio communication unit (message 4 in Figure 1). The radio communication unit tries to send data via a wireless medium (message 5 in Figure 1), considering a predefined communication protocol (message 6 to 9 in Figure 1). In this work, we assumed that the communication protocol of nodes is Time Division Multiple Access (TDMA). Note that TDMA [20] is a MAC-level communication protocol which is widely used in WSAN applications. But it can be replaced by other MAC-level protocols (e.g. by B-MAC [21]). In addition to the mentioned components, one additional component for carrying out miscellaneous tasks unrelated to sensing or communication is needed in the model of WSAN application. This additional component is necessary for making the model close to its real configuration.

Regarding the implementation, each node employs a single-core processing unit. The mentioned functionalities are assumed to be implemented by two real-time tasks, referred to as sensor task and miscellaneous task. Further, the corresponding jobs are scheduled and executed by a non-preemptive FIFO scheduling policy.

B. Design Objective

In the WSAN application the goal is that each node reaches its maximum sampling rate. This way the resource is fully utilized and the cost is reduced. Meanwhile, system buffer size as well as its processing capacity are the main limitations. In summary, the problem is to compute the maximum achievable sampling rate of sensors for a given system specification without missing any data.

The constraints which have to be satisfied in finding the maximum sampling rate can be divided into two requirements. The first requirement focuses on the timely execution of the tasks inside a node, which is expressed in Requirement 1.

Requirement 1. *All instances of sensors tasks as well as those of any other miscellaneous tasks should be served prior to the arrival of a new instance of that task.*

The second requirement, Requirement 2, is related to the communication protocol parameters. Once a packet is ready in a node, this packet should be sent before another packet becomes ready for sending in that node.

Requirement 2. *The transmission of any packet should be finished before the next packet becomes ready.*

III. Analytical Analysis Approach

In this section, we present an analytical approach to obtain the maximum sampling rate of data collection. Our solution provides a rate by which both requirements are met (but maybe it is not the maximum possible rate); hence, the solution is a *safe* (but possibly pessimistic) approximation.

Based on the description of the previous section, a node contains two tasks, namely τ_M and τ_S. Both tasks are assumed to be periodic with period of T_M and T_S, although their activation sources are an event for τ_M and a timer for τ_S. The tasks have a known worst-case execution time, which is denoted by C_M for τ_M and C_S for τ_S. Whenever any of these tasks is activated, it is inserted to a FIFO queue to be scheduled non-preemptively. Each task has a relative deadline that is equal to its period. In other words, we consider the model to be *implicit* deadline. Note that the tasks do not have activation jitter and they have no dependency, such as sharing resources or communication between each other. In this model, the Sensor task τ_S collects data from the environment. When N data samples are collected and processed by this task, a packet is created for sending the data through the network.

As mentioned before, the network access protocol is assumed to be TDMA and each sender node has a dedicated time-slot for transmission of its data. In this model, we define a super-frame with the length of T_{tdma} that all time-slots are allocated within that. If the packet is not ready at the beginning of the dedicated time-slot, it will be unused. Figure 2 demonstrates the execution of N jobs of task τ_S, preparing a packet of the collected data, and sending the packet in the dedicated time slot.

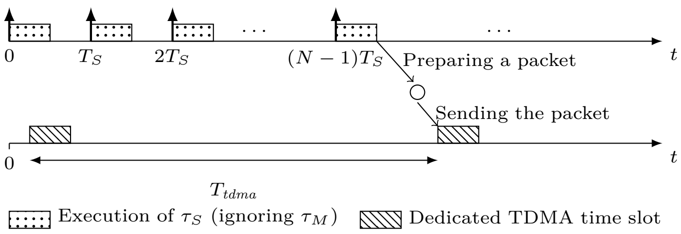

Figure 2. Creating a packet out of N data samples. The packet is sent in the first dedicated TDMA time slot.

Regarding the design objective, we are interested in finding the maximum allowable rate of activation of task τ_S. A sampling rate is valid to be used if it respects system constraints, specified in Requirements 1 and 2. To treat the mentioned requirements, we define the notion of *response time* of a job. Consider the i-th job of the Sensor task, that is the instance of τ_S which is responsible for processing the i-th data. The difference between the release time of this job and the point at which the data is completely processed is defined as the response time of the job, denoted by R_i.

A. Addressing Requirement 1

To decide schedulability of the tasks, i.e., to fulfill Requirement 1, we need to ensure that all instances of both tasks meet their deadlines. In the following, we derive a relation which guarantees tasks schedulability.

Each sender node has two periodic non-preemptive tasks which are scheduled with a FIFO queue. As there is no other priority levels in the system, the only interference for the tasks is from the same priority tasks waiting in the same queue. Therefore, in a schedulable system with implicit deadlines, when a task is queued in the FIFO queue at most one instance of other tasks can be ahead of the task under analysis [22]. This means that if there are two instances of a task ahead of the task under analysis, the system is not schedulable. According to the assumptions, a task suffers from the worst-case execution time of the other task, only. Therefore, in our system model with two periodic non-preemptive tasks, the task τ_S is schedulable if $C_M + C_S \leq T_S$. Similarly, the task τ_M is schedulable if $C_M + C_S \leq T_M$. Putting these together, the system is schedulable if,

$$C_M + C_S \leq min(T_M, T_S) \qquad (1)$$

Condition (1) serves as a sufficient schedulability test, i.e., a sufficient condition for fulfilling Requirement 1.

B. Addressing Requirement 2

In order to address the second requirement, we first compute the time when the j-th packet becomes ready. Let m_j denotes this time instant. Figure 3 depicts the relation between m_j and the execution of τ_S's jobs.

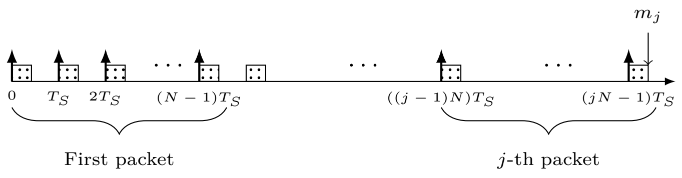

Figure 3. The execution of any N jobs of τ_S results a packet.

According to the figure, m_j can be computed as:

$$m_j = (jN-1)T_S + R_{jN}, \tag{2}$$

where R_{jN} denotes the response time of the jN-th job of τ_S. Note that the jN-th job is released at $(jN-1)T_S$.In order for a packet to be sent before the next one becomes ready, there must be at least one dedicated time slot between the points in which the packets are ready. Formally, between the instants m_j and m_{j+1}, for all $j > 0$, there must be a time slot in which the j-th packet can be sent.

To fulfill this requirement, it is sufficient to have $T_{tdma} < m_{j+1} - m_j$, which, according to Eq. (2), means:

$$\begin{aligned} T_{tdma} &< m_{j+1} - m_j \qquad (3)\\ &= ((j+1)N-1)T_S + R_{(j+1)N} - ((jN-1)T_S \\ &\quad +R_{jN}) \\ &= ((j+1)N)T_S + R_{(j+1)N} - (jN)T_S - R_{jN} \\ &= NT_S + R_{(j+1)N} - R_{jN} \end{aligned}$$

This inequality forces a lower bound on the sampling period T_S as follow:

$$T_S > \frac{T_{tdma} + R_{jN} - R_{(j+1)N}}{N} \tag{4}$$

Hence, for an exact timing analysis (for the second problem), we need to precisely calculate the response times of the Sensor jobs.

C. Response-Time Analysis

As shown above, to compute the value of m_j, we need to accomplish a response-time analysis of a job that is scheduled by a non-preemptive FIFO scheduling scheme. As exact response-time of the jobs are not known, we use upper bound and lower bound of the response time to derive a (sufficient) condition which satisfies Ineq. (4).

Proposition 1. *Let W and B denote, respectively, an upper bound and lower bound for the response time of a job of τ_S. Then,*

$$T_{tdma} < NT_S + B - W \tag{5}$$

implies Eq. (3).

Proof. Based on the definition of W and B, it holds that

$$\begin{aligned} B &\leq R_{(j+1)N}, \qquad (6)\\ R_{jN} &\leq W, \end{aligned}$$

or equivalently,

$$\begin{aligned} B &\leq R_{(j+1)N} \qquad (7)\\ -W &\leq -R_{jN}. \end{aligned}$$

This leads to

$$B - W \leq R_{(j+1)N} - R_{jN}. \tag{8}$$

With adding NT_S to both left-hand side and right-hand side of (8), it follows that $NT_S + B - W \leq NT_S + R_{(j+1)N} - R_{jN}$. Writing this together with (5), we have

$$\begin{aligned} T_{tdma} < \quad & NT_S + B - W \\ & NT_S + B - W \quad \leq NT_S + R_{(j+1)N} - R_{jN} \end{aligned}$$

This means that $T_{tdma} < NT_S + B - W$ implies $T_{tdma} < NT_s + R_{(j+1)N} - R_{jN}$, by which the proof completes. $\square$

In other words, Eq. (5) provides a sufficient condition to ensure that the messages are sent in a timely manner. Next, we consider calculating the parameters B and W. The (minimum) execution time of a single job of the Sensor task can be considered as (a lower bound on) B. Besides, an upper bound for the response time of a job is $W = C_S + C_M$. Substituting these values of B and W in Eq. (5) reveals:

$$T_{tdma} < NT_S - C_M. \tag{9}$$

D. Maximum Sampling Rate

Eqs. (1) and (9) provide two lower bounds on the period of the Sensor task, i.e., T_S. Based on this, to ensure timing constraints are met, T_S needs to satisfy:

$$T_S \geq \max(C_M + C_S, \frac{T_{tdma} + C_M + C_S}{N}) \tag{10}$$

This relation provides a limit on T_S, or equivalently, on the sampling rate, which fulfills the requirements.

IV. Model Checking Approach

Real-time Maude [23] and Timed Automata [24] are two widely used modeling languages which are equipped with model checking facilities. Real-time Maude is a high level declarative programming language supporting specification of real-time and hybrid systems in timed rewriting logic. A network of timed automata models the behavior of timed systems using a set of automata that is equipped with the set of clock variables, which their values are increased in the same rate to model progress in time. Although either of real-time Maude and timed automata can be used as the modeling formalism of the WSAN framework, their faithfulness [25] to the WSAN framework is poor. This results in increasing the cost of modeling and model checking. To avoid this inefficiency, we used Timed Rebeca [26] as the modeling language and benefit from Afra, its model checking toolset [27]. Timed Rebeca is an extension on Rebeca [28] modeling language with time features for modeling and verification of time-critical systems. Rebeca is an actor-based language for modeling concurrent and reactive systems with asynchronous message passing. This characteristics conforms the characteristics of the WSAN framework which consists of a set of concurrently executing components which are communicating by asynchronous message passing. Note that in the development of the Timed Rebeca model of the WSAN framework we do our best to consider the same level of details in comparison with the analytical model. This is necessary to be able to fairly compare the results of the two approaches.

We illustrate the Timed Rebeca language with the simplified model of the WSAN framework, shown in Listing 1. A Timed Rebeca model consists of a set of reactive classes and the main block. In the main block, actors which are instances of the reactive classes are declared (five actors in this model which are `medium`, `cpu`, `misc`, `snsd`, and `rcv`). The body of the reactive class includes the declaration of its known rebecs (line 3), state variables (line 4), and message servers (lines 9 to 12). A message server declaration consists of specifying its signature and a body which contains its corresponding Rebeca statements. The statements in the body can be assignments, conditional statements, enumerated loops, non-deterministic assignment, and method calls. Method calls are sending asynchronous messages to other actors (or to itself in line 11). A modeler can express progress in time using the `delay` function and communication delay by associating `after` to method calls.

Listing 1. The Simplified Timed Rebeca implementation of the WSAN

```
 1 env int samplingRate = 25; // Hz
 2 reactiveclass Sensor(2) {
 3   knownrebecs { CPU cpu; }
 4   statevars { int period; }
 5   Sensor() {
 6     self.sensorLoop();
 7     period = 1000 / samplingRate;
 8   }
 9   msgsrv sensorLoop() {
10     cpu.sensorEvent() deadline(period);
11     self.sensorLoop() after(period);
12   }
13 }
14 reactiveclass CPU(10) { ... }
15 reactiveclass Misc(2) { ... }
16 reactiveclass WirelessMedium(5) { ... }
17 reactiveclass CommunicationDevice (10) { ... }
18 main {
19   WirelessMedium medium():();
20   CPU cpu (snsd, rcv, sensor):();
21   Sensor sensor(cpu):();
22   Misc misc(cpu):();
23   CommunicationDevice snsd(medium):(1);
24   CommunicationDevice rcv(medium):(0);
25 }
```

A. *Addressing Requirement 1*

The first requirement of this model is implemented in the model of `Sensor` in Listing 1. The behavior of `Sensor` is to acquire data and send it to `CPU` periodically, which is implemented using the message server `sensorLoop` (lines 10-11). Note that the timing values in this model are exact values as they are specified as period of events. To address Requirement 1 we have to make sure that the sent data is served before the start time of the next period of data acquisition, which is specified by the value of `period` as the parameter of `deadline` in line 10. A similar approach is used in the implementation of `Miscellaneous`.

B. *Addressing Requirement 2*

To address the second requirement of the WSAN framework, we have to add the remaining part of the model and put a constraint in a message server which handles sending data to the radio communication device.

The behavior of `CPU` as the target of messages of `Sensor` and `Misc` is more complicated (Listing 2). Upon receiving `miscEvent`, `CPU` has to represent computation cycles consumed by miscellaneous tasks which is implemented by waiting nondeterministically for one to ten units of time (this value is represented by C_M in the analytical approach in the previous sections). Similarly, after receiving the `sensorEvent` message from `Sensor`, `CPU` waits nondeterministically for one or two units of time, which is the required computation timed for the intra-node data processing. The processed data has to be packed in a packet which has the capacity of storing `bufferSize` number of data (this value is represented by N in the analytical approach in the previous sections). When the threshold of number of data in a packet is reached (line 17), `CPU` asks `senderDevice`, to send the collected data as one packet (line 18). The type of `senderDevice` is `RCD` (the short name for the radio communication device actor). In the model of Listing 2 the required computation time of `sensorEvent` and `miscEvent` are implemented by nondeterministic values as the worst-case execution time (WCET) is 2 and 10. So, one of the values from 1 to those worst-case values may needed at each round of execution, which are modeled by nondeterministic expressions. Note that putting the WCET as the delay values of Listing 2 may result in incorrect analysis results as the model may have timing anomaly. As shown in [29], timing analysis methods which assume that considering the WCETs corresponds to the worst-case timing behavior of the system do not work for all types of applications and shorter computation times may result in the worst-case timing behavior.

Listing 2. The Timed Rebeca implementation of `CPU` reactive class

```
1 env int bufferSize = 3; // samples
2
3 reactiveclass CPU(10) {
4   knownrebecs {RCD senderDevice, receiverDevice;}
5   statevars { int collectedSamplesCounter; }
6
```

```
 7   CPU() { collectedSamplesCounter = 0; }
 8
 9   msgsrv miscEvent() {
10     //Worst-case execution time is 10
11     delay(?(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
12   }
13   msgsrv sensorEvent() {
14     //Worst-case execution time is 2
15     delay(?(1, 2));
16     collectedSamplesCounter += 1;
17     if (collectedSamplesCounter == bufferSize) {
18       senderDevice.send(receiverDevice, 1);
19       collectedSamplesCounter = 0;
20     }
21   }
22 }
```

To fulfill Requirement 2, we have to make sure that in the case of sending a `senderDevice` message in line 18, there is no ongoing sending data in `RCD`. As we show in Listing 4 of Appendix A, this requirement is implemented in the body of the `senderDevice` message server of `RCD`. Developing the radio communication device actor requires that the wireless communication medium Ether be specified and a communication protocol is implemented. The detailed implementation of Ether and TDMA protocol in `RCD` are presented in Appendix A.

V. Experimental Results and Discussion

For the aim of better understanding of trade-offs of the approaches, comparison between the analytical analysis and model checking in calculating the maximum sampling rate of the sensor is presented in this section. We defined a set of configurations and prepare the analysis result of each approach on it and provide a discussion on different types of criteria.

A. Results of Applying the Analysis Approaches

For the case of analytical approach, we consider the following values for the parameters of the model: $T_{tdma} = 10$ ms, $C_M = 10$ ms, $T_M = 120$ ms. Then, we evaluate the model for different values for the WCET of τ_S and the internal buffer size (N), as $C_S \in \{2, 10, 20, 30\}$ms and $N \in \{1, \ldots, 10\}$. As a result, according to equation (10) we obtain the minimum feasible sampling periods, i.e., T_S, as shown in Table I and the maximum sampling rates as shown in Table II. These tables also contain the minimum sampling periods and maximum sampling rates which are computed using the model checking approach. For this case, we model check the Timed Rebeca model using the same set of configurations and valuation. Comparing the values of these tables shows that in all cases, the maximum possible sampling rates which are calculated by the model checking approach are higher than that of the analytical analysis approach. So, it seems that the analytical analysis approach is a more conservative approach and computes sampling rates based on more pessimistic assumptions.

Table I
The minimum feasible sampling periods (milliseconds), computed by both approaches in different configurations

	WCET of τ_S (C_S)							
	Analytical				Model Checking			
Internal Buffer Size (N)	2	10	20	30	2	10	20	30
1	20	20	30	40	11	11	22	33
2	12	20	30	40	11	11	22	33
3	12	20	30	40	11	11	22	33
4	12	20	30	40	11	11	22	33
5	12	20	30	40	11	11	22	33
6	12	20	30	40	11	11	22	33
7	12	20	30	40	11	11	22	33
8	12	20	30	40	11	11	22	33
9	12	20	30	40	11	11	22	33
10	12	20	30	40	11	11	22	33

Table II
The maximum sampling rates (samples per second), computed by both approaches in different configurations

	WCET of τ_S (C_S)							
	Analytical				Model Checking			
Internal Buffer Size (N)	2	10	20	30	2	10	20	30
1	50	50	33	25	90	90	45	30
2	83	50	33	25	90	90	45	30
3	83	50	33	25	90	90	45	30
4	83	50	33	25	90	90	45	30
5	83	50	33	25	90	90	45	30
6	83	50	33	25	90	90	45	30
7	83	50	33	25	90	90	45	30
8	83	50	33	25	90	90	45	30
9	83	50	33	25	90	90	45	30
10	83	50	33	25	90	90	45	30

B. Discussion

Different modeling approaches affect the complexity and flexibility of the modeling process and the models. It can also affect the analyzability and precision of the results [25]. In the following, we discuss three criteria for comparing the two approaches: (i) flexibility and extensibility, i.e., the effort needed for modifying and extending models, (ii) accuracy of results, and (iii) the resource consumption for the analysis process, which concerns how much computation power are needed for approaches.

1) Flexibility and Extensibility.: For comparing the analytical and model checking approaches, from the flexibility and extensibility point of view, we study how changing the communication protocol of the model from TDMA to B-MAC is realized in two approaches. Changing the communication protocol does not affect the task model and resource model which are introduced in Section III. But, the algorithm of using the resource (the communication medium as a critical section) has to be modified. Based on the model of this paper, the behavior of communication protocols is abstracted to a value which shows the worst-case delay between requesting for a data transmission to its corresponding successful data transmission. Thus, changing the communication protocol only changes this value. In order to show this, we have modified the analysis to accommodate B-MAC communication protocol by using the schedulability analysis of B-MAC given in [30].

According to the analysis, the communication delay of a sender node is composed by several parameters, including (i) the initial backoff time t_{b1}; (ii) the frozen time between every sequential retransmission t_{f1}; (iii) the congestion backoff time t_{b2}; and (iv) the frozen time during the congestion backoff t_{f2}. The congestion can occur multiple times in a network, thus the times for them should be accounted as well. The packet transmission time (t_{pkt}) should be also added to the communication delay. Eq (11) calculates the communication delay of a sender node in B-MAC, denoted by t_{sd} [30], where k is the maximum number of retransmission in the case of congestion. This equation is representative of B-MAC transmission delay, thus T_{tdma} in Eq (10) must be replaced with t_{sd}.

$$t_{sd} = t_{b1} + t_{f1} + k(t_{b2} + t_{f2}) + T_{pkt} \tag{11}$$

Considering the same experimental setup as in Section V-A, we compute the minimum sampling periods and maximum sampling rate of sensors. Note that in B-MAC communication we assumed the maximum number of retransmission for a successful transmission is 4 (the maximum number of sender nodes in the sample network).

Although changing the communication protocol looks minor and easy to apply in Eq (11), the worst-case delay is computed based on an assumption on the maximum number of retransmissions which is not easy to compute accurately. The accurate timing behavior of B-MAC protocol is illustrated in collaboration between network nodes in runtime. So, a very complicated formula is needed to express the accurate value of the worst-case delay in B-MAC, or it has to be approximated to a value which corresponds to the very pessimistic configuration of the application (for some cases it is infinity), which is unreal for the majority of the cases.

In contrast, using Timed Rebeca, no expertise in finding delays is needed and the protocol has to be modeled using the language statements. Using B-MAC protocol, a radio communication device tries to detect when the channel is free and sends data upon receiving a request from CPU. The detailed implementation of this protocol and how collisions are handled are depicted in Appendix A,

Changing the behavioral type of sensor or miscellaneous tasks is another modification which can be used to compare flexibility in two approaches. Assume that we want to modify the models in a way that miscellaneous task is no more periodic but aperiodic. In this case, all the formulas which are used in Section III to specify the task model have to be modified. As we will discuss in the following subsection, the current formulation of the model does not support non-periodic tasks and new modeling formalism has to be used to address aperiodic tasks. In contrast, the only modification needed in the Timed Rebeca model is in the statements of the `miscLoop` message server.

2) Results Accuracy.: As shown in the experimental result, the model checking approach provided better (more accurate) results. This is because our analytical approach exhibits an approximate solution. One of the main sources of needs for such approximation is in the low expressiveness of the chosen task model (our modeling was not exact).

To make the result of the analytical approach more accurate, we have to change the task model to a model with a higher level of expressiveness. In this paper, we employed a periodic task model for the analytical approach. As a result, we could not model the possible initial offset of the activation of tasks; we assumed a pessimistic case where the tasks are released synchronously while there are task models with a higher level of expressiveness. In particular, the DRT task model, proposed by Stigge et al. [31], allows non-deterministic initial offset of a task release. This means that we can take into account more details of the system behavior with such task models, possibly resulting in a more accurate solution. This is a potential future work to model and analyze the problem with such a task model. In addition, the inherent complexity of the problem enforces some relaxation in the model, results in pessimistic approximations. Fersman et al. [32] have shown that the feasibility problem is undecidable if the following conditions hold: (1) the scheduling method is preemptive; (2) execution time of jobs is not fixed, this value is picked from a range; (3) the completion time of a job can influence the release of a next job. On the other hand, if even one of these conditions does not hold, the problem is decidable. Such complexities leads to using approximation in our analysis. Specifically, to verify Relation (3), we need a specific response time analysis which can compute (a lower bound on) the difference of the response time of a job and its N_{th} successor. To the best of our knowledge, there is not such analysis for the known real-time task models in the literature. However, to avoid a complex analysis, we replace exact values with lower and upper bounds (Eq. (5)). While this relaxation does not compromise soundness of the analysis, it may lead to pessimistic results. We have made an attempt to make use of more expressive task models for which pseudo-polynomial time feasibility tests exists. This appeared that using such models still cannot increase the analysis accuracy.

3) Effort needed for doing the Analysis.: As the last criterion, we compare the computational power needed for the two approaches to do the analysis. For the case of analytical approach, the computation power is negligible as the result is computed by setting parameters of the schedulability formula. This argument is valid for any possible values of parameters. But, in the model checking approach the state space of the model has to be generated and explored. For the case of Timed Rebeca model of this paper, the needed time is 2 seconds and 2039 states are generated on a desktop computer with 1 CPU (2 cores) and 8 GB of RAM storage, running High Sierra OS X 10.13.5, which are not too much resources. However, as mentioned in Section IV-B, we used non-deterministic expressions to address worst-case execution times in CPU and increasing the value of worst-case execution time increases the options of the non-deterministic expression. Increasing the options of non-deterministic expressions may exponentially increase the size of the state space. Therefore, in this sense, model checking approach is less scalable comparing to the analytical approach.

VI. Conclusions and Future work

In this paper, we presented a comparison study between using an analytical analysis approach and a model checking approach on a WSAN application, to obtain the highest possible data acquisition frequency of nodes. Configuring the highest possible frequency in nodes leads to achieve more efficient use of resources and in turn reducing costs. As for the analytical analysis approach we used a response time analysis technique and for the model checking approach we used Timed Rebeca, which is a modeling language for time-critical systems. The comparison study shows that the model checking approach delivers more accurate results compared to the analytical approach for the presented WSAN application. In order to improve the analytical approach, the task model has to be improved to give a better expressiveness and more behavioral details, however at the same time it makes the model more complex to analyze. Future work aims at study the effects of other task models, e.g., DRT task model, on the analytical approach compared to model checking approaches.

References

[1] L. Linderman, K. Mechitov, and B. F. Spencer, "TinyOS-Based Real-Time Wireless Data Acquisition Framework for Structural Health Monitoring and Control," *Structural Control and Health Monitoring*, 2012.

[2] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, 2004.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGPLAN Notices*, 2000.

[4] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, 1973.

[5] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, 1986.

[6] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century*, 2000.

[7] L. Waszniowski and Z. Hanzálek, "Formal verification of multitasking applications based on timed automata model," *Real-Time Systems*, 2008.

[8] P. Krčál and W. Yi, "Decidable and undecidable problems in schedulability analysis using timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2004.

[9] X. Xu, X. Y. Li, and M. Song, "Distributed scheduling for real-time data collection in wireless sensor networks," in *IEEE Global Communications Conference*, 2013.

[10] C. Xia, X. Jin, L. Kong, and P. Zeng, "Bounding the demand of mixed-criticality industrial wireless sensor networks," *IEEE Access*, 2017.

[11] B. C. Cheng, H. H. Yeh, and P. H. Hsu, "Schedulability analysis for hard network lifetime wireless sensor networks with high energy first clustering," *IEEE Transactions on Reliability*, 2011.

[12] A. Furfaro and L. Nigro, "Modelling and schedulability analysis of real-time sequence patterns using Time Petri Nets and Uppaal," in *International Multiconference on Computer Science and Information Technology*, 2007.

[13] D. Xu, X. He, and Y. Deng, "Compositional schedulability analysis of real-time systems using Time Petri Nets," *IEEE Transactions on Software Engineering*, 2002.

[14] E. Fersman, P. Pettersson, and W. Yi, "Timed automata with asynchronous processes: Schedulability and decidability," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2002.

[15] G. Li, X. Cai, and S. Yuen, "Modeling and analysis of real -time systems with mutex components," in *International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, 2010.

[16] E. Fersman and W. Yi, "A generic approach to schedulability analysis of real-time tasks," *Nordic Journal of Computing*, 2004.

[17] W. Sheng, Y. Gao, L. Xi, and X. Zhou, "Schedulability analysis for multicore global scheduling with model checking," in *International Workshop on Microprocessor Test and Verification*, 2010.

[18] N. Guan, Z. Gu, M. Lv, Q. Deng, and G. Yu, "Schedulability analysis of global fixed-priority or EDF multiprocessor scheduling with symbolic model-checking," in *International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, 2008.

[19] B. F. Spencer, H. Jo, K. A. Mechitov, J. Li, S.-H. Sim, R. E. Kim, S. Cho, L. E. Linderman, P. Moinzadeh, R. K. Giles *et al.*, "Recent advances in wireless smart sensors for multi-scale monitoring and control of civil infrastructure," *Journal of Civil Structural Health Monitoring*, vol. 6, no. 1, pp. 17–41, 2016.

[20] A. El-Hoiydi, "Spatial TDMA and CSMA with preamble sampling for low power ad hoc wireless sensor networks," in *Symposium on Computers and Communications*, 2002.

[21] J. Polastre, J. L. Hill, and D. E. Culler, "Versatile low power media access for wireless sensor networks," in *International Conference on Embedded Networked Sensor Systems*, 2004.

[22] R. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller Area Network (CAN) schedulability analysis with FIFO queues," in *Euromicro Conference on Real-Time Systems*, 2011.

[23] P. C. Ölveczky and J. Meseguer, "Real-Time Maude 2.1," *Electr. Notes Theor. Comput. Sci.*, vol. 117, pp. 285–314, 2005.

[24] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[25] M. Sirjani, "Power is overrated, go for friendliness! expressiveness, faithfulness, and usability in modeling: The actor experience," in *Principles of Modeling - Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, ser. Lecture Notes in Computer Science, M. Lohstroh, P. Derler, and M. Sirjani, Eds., vol. 10760. Springer, 2018, pp. 423–448.

[26] A. H. Reynisson, M. Sirjani, L. Aceto, M. Cimini, A. Jafari, A. Ingólfsdóttir, and S. H. Sigurdarson, "Modelling and simulation of asynchronous real-time systems using timed Rebeca," *Science of Computer Programming*, 2014.

[27] E. Khamespanah, M. Sirjani, M. Viswanathan, and R. Khosravi, "Floating time transition system: More efficient analysis of timed actors," in *International Conference in Formal Aspects of Component Software*, 2015.

[28] M. Sirjani, A. Movaghar, A. Shali, and F. S. de Boer, "Modeling and Verification of Reactive Systems using Rebeca," *Fundamental Information*, 2004.

[29] T. Lundqvist and P. Stenström, "Timing anomalies in dynamically scheduled microprocessors," in *Real-Time Systems Symposium*, 1999.

[30] S. Jung, N. Choi, and T. Kwon, "An iterative analysis of single-hop b-mac networks under poisson traffic," *Journal of Communications and Networks*, February 2012.

[31] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 71–80.

[32] E. Fersman, P. Krcal, P. Pettersson, and W. Yi, "Task automata: Schedulability, decidability and undecidability," *Information and Computation*, 2007.

[33] "Rebeca Modeling Language," http://www.rebeca-lang.org/.

Appendix

The reactive class of Ether (Listing 3) has three message servers: these are responsible for sending the status of the medium, broadcasting data, and resetting the status of the medium after a successful transmission. Broadcasting data takes place by sending data to an RCD which results in setting the values of senderDevice and receiverDevice to their corresponding actors. So, the status of Ether can be easily examined by the value of receiverDevice (i.e., using null as the value of receiverDevice is interpreted as medium is free, line 7). This way, as shown in lines 22 and 23, upon successfully data transmission, the value of receiverDev and senderDev have be set to null to show that the transmission is completed. The main behavior of Ether is data broadcasting which is implemented in lines 9 to 20. Before the start of broadcasting, the Ether status is checked (line 11) and data-collision error is raised in the case of trying for more that one simultaneous data broadcasts (line 18). With a successful data broadcast, Ether sends an acknowledgment to itself (line 14) and the sender (line 15), and informs the receiver of the number of packets sent to it (line 16).

Listing 3. The Timed Rebeca implementation of Ether reactive class

```
 1 reactiveclass Ether(5) {
 2   statevars { RCD senderDev, receiverDev; }
 3
 4   Ether() {
 5     senderDev = null;
 6     receiverDev = null;
 7   }
 8   msgsrv getStatus() {
         ((RCD)sender).receiveStatus(receiverDev !=
         null); }
 9   msgsrv broadcast(RCD receiver, int packets) {
10     byte OnePacketTT = ?(5, 6, 7); // ms(transmission
            time)
11     if(senderDev == null) {
12       senderDev = (RCD)sender;
13       receiverDev = receiver;
14       self.broadcastingIsCompleted() after(packets *
              OnePacketTT);
15       ((RCD)sender).receiveResult(true) after(packets *
              OnePacketTT);
16       receiver.receiveData(receiver, packets);
17     } else {
18       ((RCD)sender).receiveResult(false);
19     }
20   }
21   msgsrv broadcastingIsCompleted() {
22     senderDev = null;
23     receiverDev = null;
24   }
25 }
```

The Timed Rebeca implementation of the TDMA protocol in RCD is depicted in Listing 4. It shows that sending a packer, the send message server of RCD has to be called. As shown in line 18 and 19, upon receiving a request for sending data, receiverDevice and sendingData are set to the input parameters and data transmission is started. These state variables are set to null values upon finishing the transmission. So, to fulfill Requirement 2, we have to make sure that receiverDevice is set to null, i.e. there is no ongoing sending data, as implemented in line 17.

The TDMA protocol defines a cycle, over which each node in the network has one chances (a time slot) to transmit a packet or a series of packets. If a node has data available to transmit during its alloted time slot, it may be sent immediately. Otherwise, packet sending is delayed until reaching the next time slot. In the Timed Rebeca model of Listing 4 the periodic behavior of TDMA slot is implemented in the body of handleTDMASlot message server. As depicted in line 23, the value of inActivePeriod is toggled to show that whether the node is in its associated time slot. Upon starting an associated time slot of node, the existing pending data are sent (line 27) a handleTDMASlot is scheduled for terminating the time slot (line 28). Using this implementation, when CPU sends a packet to RCD, the packet is appended to the list of pending packets which are waiting for the next time slot of the node. For the sake of simplicity, some details of RCD are omitted in Listing 4. The complete source code of this model is available on the Rebeca homepage [33].

Listing 4. The Timed Rebeca implementation of the TDMA protocol in RCD

```
 1 reactiveclass RCD (10) {
 2   knownrebecs { Ether medium; }
 3   statevars {
 4     int id, slotSize, sendingData;
 5     boolean busyWithSending, inActivePeriod;
 6     RCD receiverDevice;
 7   }
 8
 9   RCD(byte myId) {
10     id = myId;
11     inActivePeriod = false;
12     sendingData = 0;
13     receiverDevice = null;
14     ...
15   }
16   msgsrv send(RCD receiver, int data) {
17     assertion(receiverDevice == null);
18     receiverDevice = receiver;
19     sendingData = data;
20     self.checkPendingData();
21   }
22   msgsrv handleTDMASlot() {
23     inActivePeriod = !inActivePeriod;
24     if(inActivePeriod) {
25       int remainedTime = tmdaSlotSize -
              currentMessageWaitingTime;
26       assertion(remainedTime > 0);
27       self.checkPendingData();
28       self.handleTDMASlot() after(remainedTime);
29     } else {
30       self.handleTDMASlot() after((slotSize *
              (numberOfNodes - 1))-
              currentMessageWaitingTime);
31     }
32   }
33   ...
}
```

As shown in Listing 5, using B-MAC protocol, a radio communication device tries to detect when the channel is free and sends data upon receiving a request from CPU (line 19 of Listing 5). In the case of busy channel, retrying for transmission takes place in line 20. If the channel is free, the requested packet is sent immediately (line 22) regardless of the status of the other nodes of the network. This way, collisions may occur. To resolve collision, the `receiveResult` message server has been changed to be aware of the result of sent data. In the case of collision, the state of the transmitter is reset and sending process is started from the beginning (line 32). In comparison with the TDMA protocol, B-MAC protocol does not need complicated and expensive synchronization methods. It also avoids data fragmentation. So, it would be more inefficient to coordinate long messages and B-MAC expects short messages, which is common for the passing packets of WSAN applications.

Listing 5. The Timed Rebeca implementation of B-MAC protocol in `RCD`

```
 1 reactiveclass RCD (10) {
 2   knownrebecs { WirelessMedium medium; }
 3   statevars {
 4     int id, sendingData;
 5     RCD receiverDevice;
 6   }
 7   RCD(int myId) {
 8     id = myId;
 9     sendingData = 0;
10     receiverDevice = null;
11   }
12   msgsrv send(RCD receiver, int data, int
        packetsNumber) {
13     assertion(receiverDevice == null);
14     receiverDevice = receiver;
15     sendingData = data;
16     medium.getStatus();
17   }
18   msgsrv receiveStatus(boolean result) {
19     if (!result) {
20       medium.getStatus() after(OnePacketTT);
21     } else {
22       medium.broadcast(receiverDevice, sendingData,
             packetsNumber);
23       delay(OnePacketTT * packetsNumber);
24     }
25   }
26   msgsrv receiveResult(boolean result) {
27     if (result) {
28       sendingPacketsNumber = 0;
29       receiverDevice = null;
30       sendingData = 0;
31     } else {
32       medium.getStatus() after (OnePacketTT);
33     }
34   }
35   msgsrv receiveData(RCD receiver, int data, int
        recPacketsNumber) { ... }
36 }
```