Schedulability Analysis of Distributed Real-Time Sensor Network Applications using Actor-based Model Checking

Ehsan Khamespanah [1,2], Kirill Mechitov [3], Marjan Sirjani [2], and Gul Agha [3]

[1] University of Tehran, School of ECE
[2] Reykjavik University, School of Computer Science and CRESS
[3] University of Illinois at Urbana-Champaign, OSL

Abstract. Many *wireless sensor and actuator network* (WSAN) applications have real-time requirements. Programmers often use informal worst-case analysis and debugging to ensure schedules that satisfy these requirements. Not only can this process be tedious and error-prone, it is inherently conservative and thus likely to lead to an inefficient use of resources. We propose a model refinement based approach to schedulability analysis that results in a guarantee of efficient resource utilization. Specifically, we represent a WSAN as a collection of *actors* whose behavior is specified using a C-based actor language extended with operators for real-time scheduling and delay representation. We show how the abstraction and compositionality properties of the actor model may be used to incrementally build a model of the entire WSAN's behavior from constituent node-level and protocol models. We demonstrate the approach with a case study of a distributed real-time data acquisition system for high frequency sensing.

Keywords: Sensor Network, Schedulability Analysis, Actor, Timed Rebeca, Model Checking

1 Introduction

Wireless Sensor and Actuator Networks (WSANs) have become an attractive method for providing low-cost continuous monitoring solutions; however, because of the complexity of concurrent and distributed programming, networking, and real-time and embedded requirements, building WSAN applications can be particularly challenging. In WSAN applications, coordination among distributed sensors must be well configured to achieve the optimum point which satisfies the tight resource constraints of low power wireless sensor nodes. The difficulty of low power constraints is coupled with the real-time deadlines of physical processes tied to a WSAN impose constraints on scheduling and resource utilization. Hence, in WSAN applications, it can be hard to find the configuration which results in the optimum coordination among nodes.

One approach is to perform an informal analysis based on conservative worst-case assumptions and empirical measurements. This can lead to schedules that do not utilize resources efficiently. For example, a workload consisting of two periodic tasks would be guaranteed safe only if the sum of the two worst-case execution times (WCET) is less than the shorter period, whereas it is possible in practice to have many safe schedules violating this restriction. For example, in continuous online structural health monitoring of civil infrastructure [24], a large amount of sensor data is collected and must be processed; to achieve this, it is critical for resources such as memory, processor cycles, and energy to be used efficiently.

Another approach often used in WSAN applications is trial and error. For example, in [21], an empirical test-and-measure approach based on binary search is used to find configuration parameters (worst-case task runtimes, timeslot length of communication protocol, etc.). This is a laborious process which nevertheless fails to provide any safety guarantees for the resulting configuration.

One possibility is to try to extend scheduling techniques that have developed for real-time systems [22] so that they can be used in WSAN environments. Unfortunately, this turns out to be difficult in practice. Many WSAN platforms rely on highly efficient event-driven operating systems such as TinyOS [16]. Unlike real-time operating systems (RTOS) common in embedded control applications, these event-driven operating systems do not provide real-time scheduling guarantees, priority-based scheduling, or resource reservation functionality. Without such support, schedulability analysis techniques cannot be effectively used. For example, in the absence of job preemption and priority-based scheduling, we are forced to use unnecessarily conservative assumptions to guarantee correctness.

We propose a model-driven approach that allows WSAN application programmers to assess the performance and functional behavior of their code throughout the design and implementation phases. We believe this methodology can result in more robust applications and reduce the number of hard-to-debug scheduling and resource use violations in WSAN applications. Our work is based on the actor model.

A WSAN application is represented as a collection of autonomous, interacting actors which communicate via message passing. A simple model can be incrementally extended and refined during the application design process, adding new interactions and scheduling constraints. Specifically, we use Timed Rebeca [29] as the modeling language and its supporting model checking tool Afra [1,18] for analysis of WSAN applications. Timed Rebeca is a high level language capable of representing functionalities and timing behaviors at a high level of abstraction, based on the actor model of computation. Afra uses the concept of Floating Time Transition System (FTS) [18]. FTS significantly reduces state space that needs to be searched; the idea is to focus on event-based properties while relaxing the constraint of generating states where the actors are all synchronized. As the examples in [19] suggest, this approach may reduce the size of state spaces from 50% to 90%. Using FTS fits with the computation model of WSAN applications and the properties that we are interested in.

We present a case study involving real-time continuous data acquisition for structural health monitoring and control (SHMC) of civil infrastructure [21].

This system has been implemented on the Imote2 wireless sensor platform, and has been deployed for long-term monitoring of several highway and railroad bridges. SHMC application development has proven to be particularly challenging: 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. Ensuring safe execution requires modeling the interactions between the CPU, sensor and radio within each node, as well as interactions between nodes. Moreover, the application tasks are not isolated from other aspects of the system: they execute alongside tasks belonging to other applications, middleware services, and operating system components. In this application, all periodic tasks (sample acquisition, data processing, and radio packet transmission) are required to complete before the next iteration starts. Our results show that a guaranteed-safe application configuration can be found using the Afra model checking tool. Moreover, this configuration improves resource utilization compared to the previous informal schedulability analysis used in [21], supporting a higher sampling rate or a larger number of nodes.
Contributions. This paper makes the following contributions:

- We show how a WSAN application may be modeled naturally as a system of actors.
- We show how abstraction and modularity can make the approach scalable.
- We present a real-world case study that illustrates the effectiveness of our approach for a real WSAN application.
- We compare the effects of using different communication protocols on the performance of the case study.

2 Preliminaries

2.1 Real-Time Data Acquisition System

A WSAN application is a distributed system with multiple sensor nodes, each comprised of the independent concurrent entities: CPU, sensor, radio, and bridged together via a wireless communication device, controlled by a transmission control protocol. Interactions between these components, both within a node and between nodes, are concurrent and asynchronous, and may come with hard real-time deadlines. Due to the stringent performance requirements, and the latencies of these operations on sensor nodes, the sensing, data processing, and communication processes must be coordinated.

Specifically, once a sample is acquired from a sensor, its corresponding radio transmission activities must be performed. At the same time, data processing tasks are performed, such as temperature correction that adjusts the sensor data to compensate for effects of temperature changes. The timing of radio transmissions from different nodes must also be coordinated via the communication protocol. This real-time data acquisition system is thus highly sensitive to timing, with soft deadlines at each step of the process needed to assure correct and efficient operation.

2.2 The Actor Model of WSAN Applications

Actor model is a well-established paradigm for modeling distributed and asynchronous component-based systems. This model was originally introduced by Hewitt as an agent-based language where goal directed agents did logical reasoning [13, 15]. Subsequently, the actor model developed as model of concurrent computation where *actors* for distributed systems [3]. One way to think of actors is a service oriented framework: each actor provides services that may be requested by other actors which send messages to the service providers. Messages are buffered until the provider is ready to execute the message. As are result of processing a message, an actor may send messages to other actors (including to itself). The actor model is suitable for modeling of component-based systems [14]. There are some extension on the actor model for supporting modeling and analysis of real-time systems, e.g., RT-synchronizer [28], real-time Creol [8], and Timed Rebeca [29].

The characteristics of real-time variants of the actor model make them useful for modeling WSAN applications: many concurrent processes and interdependent real-time deadlines, and common tasks shared by different applications (sample acquisition, sample processing, radio transmission). To model WSAN application, using the actor model, we propose mapping of Figure 1 which closely mimics the structure of real WSAN applications. Observe that the fact that such common tasks are usually periodic and have well-known or easily measurable worst-case execution times makes the analysis feasible. However, because of event-triggered nature of the applications, the initial offsets between the tasks is variable.

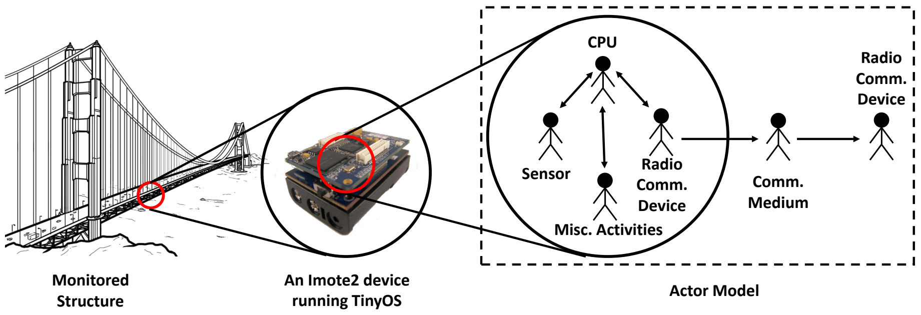


Fig. 1: How the behavior of a WSAN application in its real-world installation is modeled by the actor model

We represent components of each WSAN node capable of independent action as an actor. The figure shows that a sensor on a bridge can be modeled using four different actors for the data acquisition (i.e., *Sensor*), processing (i.e., *CPU*), radio transmission tasks (i.e., *Radio Comm. Device*), and miscellaneous tasks unrelated to sensing or communication (i.e., *Misc Activities*). This way, the sensor actor collect data and send it to CPU for further data processing. Meanwhile, CPU may respond to the miscellaneous tasks. The processed data is sent to radio communication device to send it to a data collector node. In this model, communication medium and receiver node are modeled by *Comm. Medium* and *Radio Comm. Device* actors respectively.

To keep the design of the model modular, we define *Comm. Medium* actor to model the behavior of “ether” (the communication medium for wireless communication) and push the implementation of MAC level details of communication protocols into *Radio Comm. Device* actor. This way, it is easy to replace component sub-models for modeling different communication protocols without significantly affecting the remainder of the model. This is a desirable feature, as during the application design phase different components, services, and protocols may be considered. For example, TDMA [10] as a MAC level communication protocol may be replaced by B-MAC [27] with minimal changes to the original model.

Although schedulability analysis of WSAN applications can be challenging in the absence of a real-time scheduler, we can reduce the problem of checking for deadline violations in our model to the problem of reachability from a relatively small set of possible initial configurations. Model checking is the natural approach to this class of problems, and it is the approach we explore in this paper.

2.3 Timed Rebeca and the Model Checking Toolset

We use Timed Rebeca [2, 29] modeling language and Afra [1, 18] toolset as the basis of the model checking of WSAN applications. Timed Rebeca (TR) is an extension to Rebeca [33, 34]. TR is an actor-based modeling language which can represent the timing behavior of distributed reactive systems in addition to their functional behavior.

A TR model consists of reactive classes and a main part that contains instantiation of actors (we also call them rebecs) from reactive classes. Actors in Timed Rebeca have encapsulated states, local times, and their own execution thread. Each actor contains a set of state variables, methods and a set of known actors (i.e., actors which it can communicate with). Communication is asynchronously established through message passing. Message passing is fair and implemented by method calls; calling a method of an actor results in sending a message to the actor that invokes corresponding service. Each actor has a message bag, for arriving messages.

Timing behaviors of systems are addressed in Timed Rebeca models using three timing primitives: `delay`, `after`, and `deadline`. A `delay` statement models the passing of time for an actor. The primitives `after` and `deadline` can be used in conjunction with a sending message statement. The term `after` n indicates it takes n units of time for the message to be delivered to its receiver. The term `deadline` n shows that if the message is not taken in n units of time, it will be purged from the receiver's bag automatically. (We will review the syntax of Timed Rebeca using an example in the next section.)

Afra is the modeling and model checking toolset of Rebeca family models. Afra 1.0 supports model checking of Rebeca models against LTL and CTL properties. To support model checking of Timed Rebeca models, Afra 2.0 is released as the extension of Afra 1.0. Using Afra 2.0, deadlock-freedom and schedulability analysis of Timed Rebeca models became possible.

TR and Afra toolset have previously been used to model and analyze actor based models such as routing algorithms and scheduling policies in NoC (Network on Chip) designs [30, 31].

3 Schedulability Analysis of A Stand-Alone Node

We now illustrate our approach with a node-level TR model of a WSAN application. Using the node-level model, we check for possible deadline misses. Moreover, by changing the timing parameters of our model, we find the maximum safe sampling rate in the presence of other (miscellaneous) tasks in the node. We will then show how the specification of a node-level model can be naturally extended to network-wide specifications.

Based on the mapping of Figure 1, the model of a node is decomposed into four different actors. So, in the corresponding Timed Rebeca model, four different *reactive classes* are defined as the types of the actors, which are `CPU`, `Sensor`, `Radio`, and `Misc` (miscellaneous tasks unrelated to sensing or communication). Reactive classes are defined using `reactiveclass` keyword. The definition of `Sensor` reactive class is shown in Figure 2.

As shown in the figure, the only known actor of `Sensor` is an actor of type `CPU` (line 4) and it does not have any state variable (line 5). The behavior of this sensor is to periodically acquire data and send it to the `CPU`. The sensor behavior is implemented in message server `sensorLoop` (lines 13-17). The message server sends the acquired data to `CPU` (line 15) and the sent data must be serviced before the start time of the next period, specified by the value of `period` as the parameter of `deadline`.

Recall the following property of WSAN applications: there is a non-deterministic initial offset after which the data acquisition becomes a periodic task. To support this property, message server `sensorFirst` is defined in `Sensor` which sends `sendLoop` message to itself that delivers after non-deterministic amount of time elapse in range of 10, 20, and 30 (Timed Rebeca expression $?(e_1, e_2, \cdots, e_n)$ non-deterministically chooses one of e_1 to e_n). To start the sensors periodic behavior after passing a random offset, constructor of `Sensor` sends `sensorFirst` message to itself (line 8). The same approach is used in the implementation of `Misc` reactive class. Note that, in the first line of Figure 2, the sampling rate of the sensor is defined as the constant value in the model, using `env` keyword.

The behavior of `CPU` as the target of `Sensor` and `Misc` events is more complicated than the other two actors (Figure 3). Upon receiving a `miscEvent`, cpu waits for `miscTaskDelay` units of time to model the computation power which consumes by miscellaneous tasks. In the same way, after receiving `sensorEvent` message, cpu waits for `sensorTaskDelay` units of time to model the time which

```
env int samplingRate = 25; //25Hz

reactiveclass Sensor(10) {
  knownrebecs { CPU cpu; }
  statevars { }

  Sensor() {
    self.sensorFirst();
  }
  msgsrv sensorFirst() {
    self.sensorLoop() after(?(10, 20, 30)); // 10, 20, or 30 ms
  }
  msgsrv sensorLoop() {
    int period = 1000 / samplingRate;
    cpu.sensorEvent() deadline(period);
    self.sensorLoop() after(period);
  }
}
```

Fig. 2: Reactive class of the `Sensor`

```
env int sensorTaskDelay = 2; // 2ms
env int miscTaskDelay = 10; // 10ms
env int bufferSize = 3; // 3 acquired data

reactiveclass CPU(10) {
  knownrebecs { CommunicationDevice senderDevice, receiverDevice; }
  statevars { int collectedSamplesCounter; }

  CPU() { collectedSamplesCounter = 0; }

  msgsrv miscEvent() {
    delay(miscTaskDelay);
  }
  msgsrv sensorEvent() {
    delay(sensorTaskDelay);
    collectedSamplesCounter += 1;
    if (collectedSamplesCounter == bufferSize) {
      senderDevice.send(receiverDevice, 1);
      collectedSamplesCounter = 0;
    }
  }
}
```

Fig. 3: Reactive class of the `CPU`

is required for intra-node data processing. As data must be packed in a packet (number of data in packet is specified by `bufferSize`), the number of collected data increased by one unit (line 16) and if the threshold is reached (line 17), cpu asks its communication device, `senderDevice`, to send the collected data (line 18).

As in the node-level model of WSAN application, the communication of nodes is omitted, the behavior of `CommunicationDeveice` is limited to waiting for some amount of time to model the sending time of a packet.

Note that, computation times in the model (which are modeled by `delay`s) generally depend on the low-level aspects of the system and is largely application-independent, and hence can be obtained before the application design. For schedulability analysis, we set `deadline` for messages in a way that any scheduling violations are caught by the model checker.

4 Schedulability Analysis of Multi-Node Model with a Distributed Communication Protocol

Transitioning from a stand-alone node model to an entire distributed application, multi-node model, and analyzing the communication protocol requires additional modeling efforts. Essentially, the wireless communication medium must be defined and the behavior of `CommunicationDevice` must be enriched according to represent the communication protocol it supports. For these models, we are interested in analyzing the behavior of the entire distributed sensing application; so, both the node-level and multi-node models must be considered at once. As shown in the actor model of Figure 1, nodes in the multi-node model periodically send their data to an aggregator node. The sending process is coordinated by a wireless network communication protocol. Based on the figure, one new actor must be added to the model to mimic the behavior of the wireless communication medium.

The reactive class of `CommunicationMedium` models the behavior of the air in the wireless communication. As shown in Figure 4, there are three message servers in `CommunicationMedium` which are responsible for sending the status of the medium, broadcasting data, and resetting the condition of the medium after a successful transmission.

In this model, broadcasting data takes place by sending data to an actor of type `CommunicationDevice` which addressed by `receiverDevice` variable. So, we can easily examine the status of the `CommunicationMedium` using the value of `receiverDevice` (i.e., medium is free if `receiverDevice` is not `null`, line 13). This way, after sending data, the value of `receiverDevice` and `senderDevice` must be set to `null` to show that the transmission is completed (lines 28 and 29). Data broadcasting is the main behavior of the medium which is modeled in lines 15 to 26. As depicted in line 16, before the start of broadcasting, medium status is checked and data-collision error is raised in case of asking for two simultaneous broadcastings (line 24). In a successful data broadcasting, the medium sends an acknowledgment to itself (line 19) and the sender (line 20), and informs the receiver that a number of packets are sent to it (line 21). In addition to the functional requirements of communication medium, the non-functional requirements can be added to the medium. For example, the radio of WSAN nodes offers a theoretical maximum transfer speed of 250 Kbps. When considering only the maximum possible data payload, the maximum data throughput further reduces to about 125 Kbps. This limitation can be considered in the `CommunicationMedium`.

```
 1 env int OnePacketTT = 7; //One Packet Transmission Time
 2
 3 reactiveclass WirelessMedium(5) {
 4   statevars {
 5     CommunicationDevice senderDevice, receiverDevice;
 6   }
 7
 8   WirelessMedium() {
 9     senderDevice = null;
10     receiverDevice = null;
11   }
12   msgsrv getStatus() {
13     ((CommunicationDevice)sender).receiveStatus(receiverDevice != null);
14   }
15   msgsrv broadcast(CommunicationDevice receiver, int packetsNumber) {
16     if(senderDevice == null) {
17       senderDevice = (CommunicationDevice)sender;
18       receiverDevice = receiver;
19       self.broadcastingIsCompleted() after(packetsNumber * OnePacketTT);
20       ((CommunicationDevice)sender).receiveResult(true) after(packetsNumber
             * OnePacketTT);
21       receiverDevice.receiveData(receiver, packetsNumber);
22     } else {
23       ((CommunicationDevice)sender).receiveResult(false);
24     }
25   }
26   msgsrv broadcastingIsCompleted() {
27     senderDevice = null;
28     receiverDevice = null;
29   }
30 }
```

Fig. 4: Reactive class of the `CommunicationMedium`

Now, we have to extend the model of `CommunicationDevice` to support communication protocols. Figure 5 shows the model of TDMA protocol implementation of `CommunicationDevice`. TDMA protocol defines a cycle, over which each node in the network has one or more chances to transmit a packet or a series of packets. If the node has data available to transmit during its time slot, it can be sent immediately. On the other hand, if the data becomes available during some other node's time slot, the packet sending is delayed until its next transmission slot arrives. The periodic behavior of TDMA slot is handled by `handleTDMASlot` message server. This message server periodically sets and unsets `inActivePriod` to show that the node is in its associated time slot or not. Upon entering into the associated slot, the device checks for pending data to send them (line 31) and schedules `handleTDMASlot` message to leave the slot (line 30). On the other hand, when `CPU` sends a packet to the communication device (i.e., sending `send` message), this message added to the other pending packets which wait for the next TDMA slot. For the sake of simplicity, the implementation details of the communication device are omitted in Figure 5. The complete source code of this model and the model which communicates by B-MAC protocol is available on the Rebeca web page [1].

Once a complete model of the distributed application has been created, the Afra model checking tool can verify whether the schedulability properties hold in all reachable states of the system. If there are any deadline violations, a counterexample will be produced, indicating the path—sequence of states from an initial configuration—that results in the violation. This information can be helpful with changing the system parameters, such as increasing TDMA time slot length, to prevent such situations.

5 Experimental Results and a Real-World Case Study

We examined the applicability of our approach using a WSAN model intended for use in structural health monitoring and control (SHMC) applications. Wireless sensors deployed on civil structures for SHMC collect high-fidelity data such as acceleration and strain. Structural Health Monitoring (SHM) involves identifying and detecting any damaged elements of the structure by measuring changes in strain and vibration response. SHM can also be employed for *structural control*, where it is fed into algorithms that control centralized or distributed control elements such as active and semi-active dampers. The control algorithms attempt to minimize vibration and maintain stability in response to excitations from rare events such as earthquakes, or more mundane sources such as wind and traffic. The system we examine has been implemented on the Imote2 wireless sensor platform [21], which features a powerful embedded processor, sufficient memory size, and a high-fidelity sensor suite required to collect data of sufficient quality for SHMC purposes. These nodes run the TinyOS operating system, supported by middleware services of the Illinois SHM Services Toolsuite [17].

This flexible data acquisition system can be configured to support real-time collection of high-frequency, multi-channel sensor data from up to 30 wireless smart sensors at frequencies up to 250 Hz. As it is designed for high-throughput sensing tasks that necessitate larger networks sizes with relatively high sampling

```
env int OnePacketTT = 7; //One Packet Transmission Time

reactiveclass CommunicationDevice (3) {

  knownrebecs { WirelessMedium medium; }

  statevars {
    byte id;
    int slotSize;
    boolean inActivePeriod;

    int sendingPacketsNumber;
    CommunicationDevice receiverDevice;
    boolean busyWithSending;
  }

  CommunicationDevice(byte myId) {
    id = myId;
    inActivePeriod = false;
    busyWithSending = false;
    sendingPacketsNumber = 0;
    receiverDevice = null;

    if (id != 0) { handleTDMASlot(); }
  }
  msgsrv handleTDMASlot() {
    inActivePeriod = !inActivePeriod;
    if(inActivePeriod) {
        assertion(tmdaSlotSize - currentMessageWaitingTime > 0);
        self.handleTDMASlot() after(tmdaSlotSize -
            currentMessageWaitingTime);
        self.checkPendingData();
    } else {
        self.handleTDMASlot() after((tmdaSlotSize * (numberOfNodes - 1))-
            currentMessageWaitingTime);
    }
  }

  msgsrv send(CommunicationDevice receiver, int packetsNumber) {
    assertion(receiverDevice == null);
    sendingPacketsNumber = packetsNumber;
    receiverDevice = receiver;
    self.checkPendingData();
  }
  msgsrv checkPendingData() { ... }

  msgsrv receiveResult(boolean result) { ... }
}
```

Fig. 5: Reactive class of the Sensor

rates, it falls into the class of *data-intensive sensor network applications*, where efficient resource utilization is critical, since it directly determines the achievable scalability (number of nodes) and fidelity (sampling frequency) of the data acquisition process. Configured on the basis of network size, associated sampling rate, and desired data delivery reliability, it allows for near-real-time acquisition of 108 data channels up to 30 nodes—each node may provide multiple sensor channels, such as 3-axis acceleration, temperature, or strain—with minimal data loss. In practice, these limits are determined primarily by the available bandwidth of the IEEE 802.15.4 wireless network and sample acquisition latency of the Imote2 sensors. The accuracy of estimating the safe limits of sampling and data transmission delays thus directly impacts the system's efficiency.

To illustrate the applicability of this work, we considered applications where the maximum possible sampling rate (which does not result in deadline missed) is required. It is a very common requirement in WSAN applications in SHMC domain as increasing the sampling rate increases the chance of early detection of faults. To this aim, we set the value of `OnePacketTT` to 7ms (i.e., the average transmission time of this type of applications) and fixed the value of `sensorTaskDelay`, `miscPeriod`, and `miscTaskDelay` to some predefined values. This way, in addition to the sampling rate, the number of nodes in the network and the packet size remained variable. So, by assuming different values for the number of nodes and the packet size, different maximum sampling rates are achieved, shown in Figure 6. As shown in the figure, higher sampling rates are possible when the buffer size is set to a big number to have more space for data in each packet. Similarly, increasing the number of nodes decreases the sampling rate: in competition among three different parameters of Figure 6, the cases with the maximum buffer size (i.e., 9 data) and minimum number of nodes (i.e., 1 node) results in the highest possible maximum sampling rates. Decreasing the buffer size or increasing the number of nodes, non-linearly reduces the maximum possible sampling rate.

For this experiment, a server with Intel Xeon E5645 @ 2.40GHz CPUs and 50GB of RAM storage, running Red Hat 4.4.6-4 as the operating system is used as the model checking host. In these experiments, the size of the state spaces varies from less than 500 states to more than 140K states, results in different model checking times in range of 0 to 6 seconds. Analyzing the specifications of the state spaces, some relations between the size of the state spaces and the configurations of the models are distinguishable. For example, top biggest state spaces correspond to configurations where `sensorTaskDelay`, `bufferSize`, and `numberOfNodes` are set to high values. The Timed Rebeca code of this case study, some complimentary shell scripts, the model checking toolset, and the details of the specifications of the state spaces in different configurations are accessible from Rebeca homepage [1].

We also wanted to compare the effect of the communication protocol and the value of `sensorTaskDelay` in the supported maximum sampling rate, considering 648 different configurations. The found maximum sampling rates of each configuration is depicted in Figure 7. As shown in the figures, increasing the

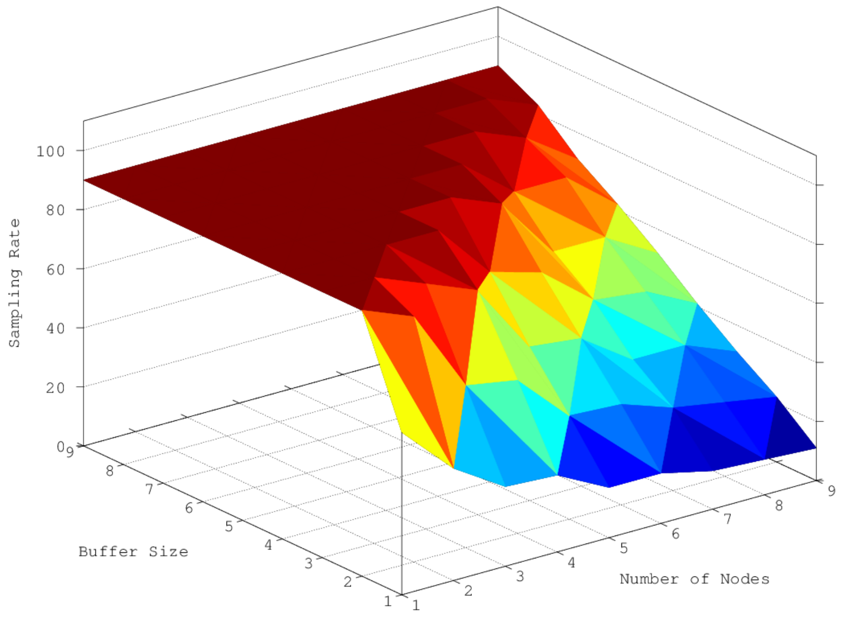


Fig. 6: The maximum sampling rate in case of using TDMA protocol and setting the value of `sensorTaskDelay` to 2ms

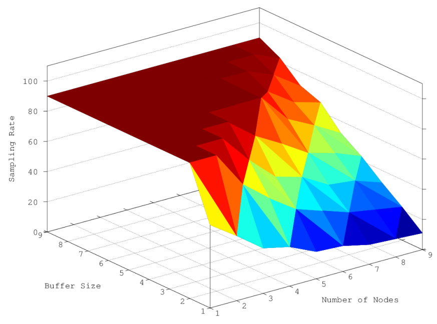
value of `sensorTaskDelay` as the representor of intra-node activities, decreases the sampling rate dramatically. It is also inferred that using B-MAC results in achieving higher sampling rates in comparison to TDMA.

We used the approach for the analysis of configurations using parameters that were determined through a real-world installation of a WSAN application. Specifically, we set the parameters of the model to values measured from a running SHMC application and found that the current installation can be tunes to a more optimized one. By changing the configuration, the performance of that system is safely improved by 7% percent.

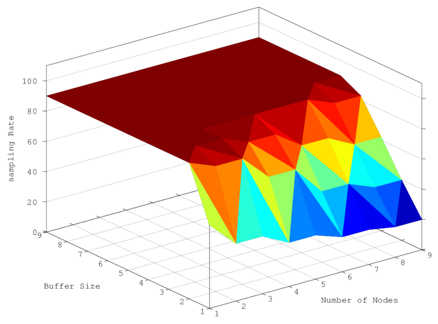
6 Related Work

WSAN applications, as real-time distributed systems, need to be supported by complementary methods that aim at correctness of them with a very high level of confidence. To achieve this goal, three different approaches have been suggested which are system simulation, analytical approach, and formal verification.

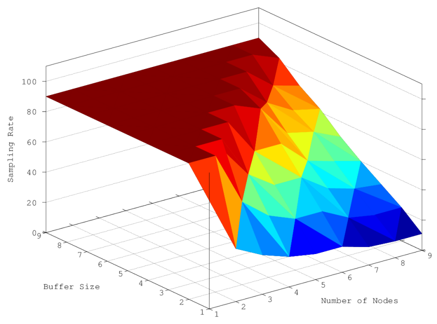
System Simulation. Simulation of WSAN applications is extremely useful for their early design exploration. It provides adequate estimates in performance evaluation of systems and sometimes detects conditions which lead to deadline misses. But even extensive simulation does not guarantee that deadline misses will never occur in the future [7]. For WSAN applications with hard real-time requirements this is not satisfactory.



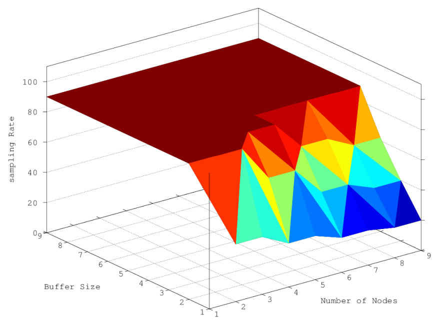
(a) TDMA, Sensor task delay is 5ms



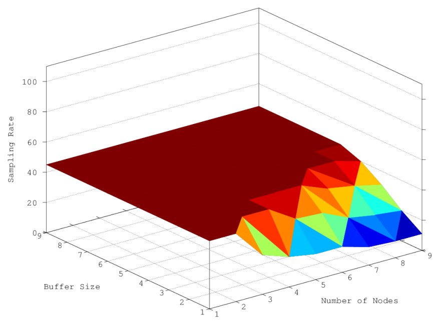
(b) B-MAC, Sensor task delay is 5ms



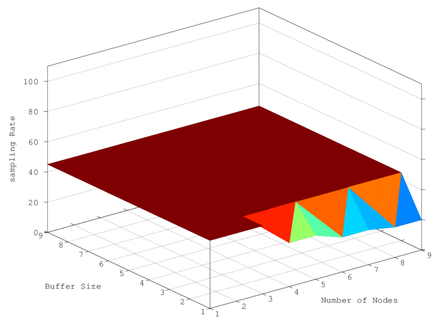
(c) TDMA, Sensor task delay is 10ms



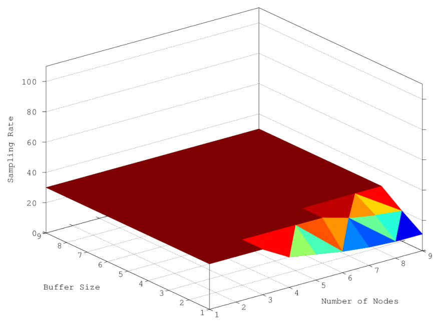
(d) B-MAC, Sensor task delay is 10ms



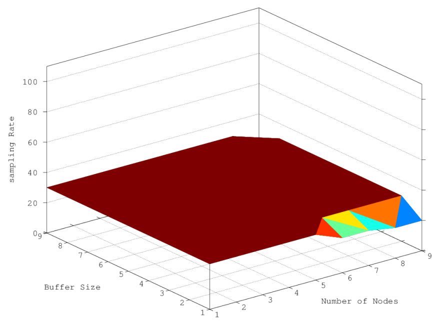
(e) TDMA, Sensor task delay is 20ms



(f) B-MAC, Sensor task delay is 20ms



(g) TDMA, Sensor task delay is 30ms



(h) B-MAC, Sensor task delay is 30ms

Fig. 7: Maximum possible sampling rate in case of different communication protocols, number of nodes, sensor internal task delays, and radio packet size

In addition, there exists only two simulation toolsets for WSAN applications, which support analysis of networking [20] and power-consumption [32] of application. So, none of them can be used for analysis of software-wise aspects of WSAN applications, which addressed in this work.

Analytical Approach. In the classical schedulability theory a large number of techniques have been suggested for schedulability analysis of real-time systems with periodic tasks and sporadic tasks with constraints, e.g. [23]. Although the classical techniques and algorithms work for schedulability analysis of real-time systems with periodic tasks and sporadic tasks efficiently, lack of ability of modeling random tasks make them inappropriate for schedulability analysis of WSAN applications. In addition, algorithms of analytical approaches do not work efficiently for multi-processor applications. The time complexity of these algorithms increases from polynomial time for single processor systems, to NP-Complete for multi-processor systems [5]. Compared to the analytical approaches, here we build a more realistic model of the system where we can consider details and also the concurrency of processes, and we explore the entire reachable state space of the system. So, we can provide an answer that is both correct and not unnecessarily conservative.

Formal Verification. Real-time model checking, as one of formal verification techniques, has become an attractive and maturing approach for schedulability analysis with absolute guarantees on the analysis results [7]. Model checking tools systematically check whether a model satisfies a given property [6]. The strength of model checking is not only in providing a rigorous correctness proof, but also in the ability to generate counter-examples, as diagnostic feedback in case a property is not satisfied. This information can be helpful to find flaws in the system. Norström et al. in [25] suggested an extension of timed automata to support schedulability analysis of real-time systems with random tasks. Following the work of [25], Feresman et al. studied an extension of timed automata which its main idea is to associate each location of timed automata with tasks, called task automata [12]. TIMES [4] is a toolset which is implemented based on the approach of Feresman et al. [11] for analysis of task automata using UPPAAL as back-end model checker. TIMES assumes that tasks are executed on a single processor. This assumption is the main obstacle against using TIMES for schedulability analysis of WSAN applications, which are real-time distributed applications. De Boer et al. in [9] presented a framework for schedulability analysis of real-time concurrent objects. This approach supports both multi-processor systems and random task definition, which are required for schedulability analysis of WSAN applications. But asynchronous communication among concurrent elements of WSAN application results in generation of complex behavioral interfaces which leads to state space explosion even for small size examples.

Another approach that is proposed for model checking of wireless sensor network algorithms is using Real-Time Maude. Real-Time Maude is used in [26] for performance estimation and model checking of WSAN algorithms. The approach is thorough and can support modeling of many details like communication range and amount of energy, but the definition of the behavior of a node requires a good knowledge of rewrite logic while our approach can be easily used by an engineer who is designing the system.

Compared to other formal methods and model checking tools, we use an actor-based language and tool. The language extends straight-forward C-like syntax to concurrency and enables sensing, radio communication, and data processing to be expressed. Our approach requires virtually no formal methods experience from the WSAN application programmer, as the language and structure of the model closely mirror those of the real application. Moreover, unlike theorem proving methods which require significant expertise, model checking is attractive due to its push-button nature.

7 Conclusion

We have shown how model checking method may be useful in analyzing schedulability and resource utilization in WSAN applications. WSAN applications are very sensitive to their configurations: the effects of even minor modifications to configurations must be analyzed. With little additional effort required on behalf of the application programmer, our approach provides a much more accurate view of an WSAN application's behavior and its interaction with the operating system and distributed middleware services than can be obtained by the sort of informal analysis or trial-and-error methods commonly in use today.

Our realistic–but admittedly limited–experimental results support the idea that the use of formal tools may result in more robust WSAN applications. Moreover, this may greatly reduce development time as many potential problems with scheduling and resource utilization may be identified before a single line of application code has been written.

An important direction for future research is the addition of probabilistic behavior analysis support to the tool. In many non-critical applications, infrequent scheduling violations may be considered a reasonable trade-off for increased efficiency in the more common case. Development of a probabilistic extension is currently underway.

References

1. Rebeca Formal Modeling Language. `http://www.rebeca-lang.org/`.
2. Luca Aceto, Matteo Cimini, Anna Ingólfsdóttir, Arni Hermann Reynisson, Steinar Hugi Sigurdarson, and Marjan Sirjani. Modelling and simulation of asynchronous real-time systems using timed rebeca. In Mohammad Reza Mousavi and António Ravara, editors, *Proceedings 10th International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2011, Aachen, Germany, 10th September, 2011.*, volume 58 of *EPTCS*, pages 1–19, 2011.
3. Gul A. Agha. *ACTORS - a model of concurrent computation in distributed systems.* MIT Press series in artificial intelligence. MIT Press, 1990.
4. Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times: A tool for schedulability analysis and code generation of real-time systems. In Kim Guldstrand Larsen and Peter Niebert, editors, *FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2003.

5. Jianer Chen and Chung-Yee Lee. General multiprocessor task scheduling. *Naval Research Logistics*, 46:57–74, 1999.
6. Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.
7. Alexandre David, Jacob Illum, Kim G. Larsen, and Arne Skou. *Model-Based Design for Embedded Systems*, chapter Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1, pages 93–119. CRC Press, 2010.
8. Frank S. de Boer, Tom Chothia, and Mohammad Mahdi Jaghoori. Modular Schedulability Analysis of Concurrent Objects in Creol. In Farhad Arbab and Marjan Sirjani, editors, *Fundamentals of Software Engineering, Third IPM International Conference, FSEN 2009, Kish Island, Iran, April 15-17, 2009, Revised Selected Papers*, volume 5961 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 2009.
9. Frank S. de Boer, Mohammad Mahdi Jaghoori, and Einar Broch Johnsen. Dating concurrent objects: Real-time modeling and schedulability analysis. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.
10. Amre El-Hoiydi. Spatial TDMA and CSMA with preamble sampling for low power ad hoc wireless sensor networks. In *Proceedings of the Seventh IEEE Symposium on Computers and Communications (ISCC 2002), 1-4 July 2002, Taormina, Italy*, pages 685–692. IEEE Computer Society, 2002.
11. Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability Analysis of Fixed-Priority Systems Using Timed Automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.
12. Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In Joost-Pieter Katoen and Perdita Stevens, editors, *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.
13. Carl Hewitt. Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot. MIT Artificial Intelligence Technical Report 258, Department of Computer Science, MIT, April 1972.
14. Carl Hewitt. What Is Commitment? Physical, Organizational, and Social (Revised). pages 293–307, 2007.
15. Carl Hewitt, Peter Bishop, and Richard Steiger. A Universal Modular ACTOR Formalism for Artificial Intelligence. In Nils J. Nilsson, editor, *IJCAI*, pages 235–245. William Kaufmann, 1973.
16. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGPLAN Notices*, 35:93–104, November 2000.
17. Illinois SHM Services Toolsuite. `http://shm.cs.illinois.edu/software.html`.
18. Ehsan Khamespanah, Marjan Sirjani, Zeynab Sabahi-Kaviani, Ramtin Khosravi, and Mohammad-Javad Izadi. Timed rebeca schedulability and deadlock freedom analysis using bounded floating time transition system. *Sci. Comput. Program.*, 98:184–204, 2015.
19. Ehsan Khamespanah, Marjan Sirjani, Mahesh Viswanathan, and Ramtin Khosravi. Floating Time Transition System: More Efficient Analysis of Timed Actors. In Christiano Braga and Peter Csaba Ölveczky, editors, *Formal Aspects of Component Software - 12th International Symposium, FACS 2015, Rio de Janeiro, Brazil, October 14-16, 2015*, Lecture Notes in Computer Science. Springer, 2016.

20. Philip Levis, Nelson Lee, Matt Welsh, and David E. Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. In Ian F. Akyildiz, Deborah Estrin, David E. Culler, and Mani B. Srivastava, editors, *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, Los Angeles, California, USA, November 5-7, 2003*, pages 126–137. ACM, 2003.
21. Lauren Linderman, Kirill Mechitov, and Billie F. Spencer. TinyOS-Based Real-Time Wireless Data Acquisition Framework for Structural Health Monitoring and Control. *Structural Control and Health Monitoring*, 2012.
22. Giuseppe Lipari and Giorgio Buttazzo. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *Journal of Systems Architecture*, 46(4):327–338, 2000.
23. Jane W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.
24. Tomonori Nagayama, Billie F. Spencer, Kirill Mechitov, and Gul Agha. Middleware services for structural health monitoring using smart sensors. *Smart Structures and Systems*, 5(2), 2008.
25. Christer Norström, Anders Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *RTCSA*, pages 182–189. IEEE Computer Society, 1999.
26. Peter Csaba Ölveczky and Stian Thorvaldsen. Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in real-time maude. *Theor. Comput. Sci.*, 410(2-3):254–280, February 2009.
27. Joseph Polastre, Jason L. Hill, and David E. Culler. Versatile low power media access for wireless sensor networks. In Stankovic et al. [35], pages 95–107.
28. Shangping Ren and Gul Agha. RTsynchronizer: Language Support for Real-Time Specifications in Distributed Systems. In Richard Gerber and Thomas J. Marlowe, editors, *Workshop on Languages, Compilers, & Tools for Real-Time Systems*, pages 50–59. ACM, 1995.
29. Arni Hermann Reynisson, Marjan Sirjani, Luca Aceto, Matteo Cimini, Ali Jafari, Anna Ingólfsdóttir, and Steinar Hugi Sigurdarson. Modelling and Simulation of Asynchronous Real-Time Systems using Timed Rebeca. *Sci. Comput. Program.*, 89:41–68, 2014.
30. Zeinab Sharifi, Siamak Mohammadi, and Marjan. Sirjani. Comparison of NoC Routing Algorithms Using Formal Methods. To be published in proceedings of PDPTA'13, 2013.
31. Zeinab Sharifi, Mahdi Mosaffa, Siamak Mohammadi, and Marjan Sirjani. Functional and performance analysis of network-on-chips using actor-based modeling and formal verification. *ECEASST*, 66, 2013.
32. Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoffrey Werner-Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In Stankovic et al. [35], pages 188–200.
33. Marjan Sirjani and Mohammad Mahdi Jaghoori. Ten years of analyzing actors: Rebeca experience. In Gul Agha, Olivier Danvy, and José Meseguer, editors, *Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday*, volume 7000 of *Lecture Notes in Computer Science*, pages 20–56. Springer, 2011.
34. Marjan Sirjani, Ali Movaghar, Amin Shali, and Frank S. de Boer. Modeling and verification of reactive systems using rebeca. *Fundam. Inform.*, 63(4):385–410, 2004.
35. John A. Stankovic, Anish Arora, and Ramesh Govindan, editors. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys 2004, Baltimore, MD, USA, November 3-5, 2004*. ACM, 2004.