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, Tehran, Iran
[2] Reykjavik University, School of Computer Science and CRESS, Reykjavik, Iceland
[3] University of Illinois at Urbana-Champaign, OSL, Champaign, United States

Abstract. Programmers often use informal worst-case analysis and debugging to ensure schedules that satisfy real-time 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 to use model checking to find a schedule which optimizes the use of resources while satisfying real-time requirements. Specifically, we represent a *Wireless sensor and actuator network* (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 a WSAN's behavior from node-level and network models. We demonstrate the approach with a case study of a distributed real-time data acquisition system for high frequency sensing using Timed Rebeca modeling language and the Afra model checking tool.

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

1 Introduction

Wireless sensor and actuator networks (WSANs) can provide low-cost continuous monitoring. However, building WSAN applications is particularly challenging. Because of the complexity of concurrent and distributed programming, networking, real-time requirements, and power constraints, it can be hard to find a configuration that satisfies these constraints while optimizing resource use. A common approach to address this problem 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 to be safe only if the sum of the two worst-case execution times (WCET) were less than the shorter period, whereas it is possible in practice to have many safe schedules violating this restriction.

A second approach is trial and error. For example, in [18], an empirical test-and-measure approach based on binary search is used to find configuration parameters: worst-case task runtimes, timeslot length of the communication protocols, etc. Trial and error is a laborious process, which nevertheless fails to provide any safety guarantees for the resulting configuration.

A third possibility is to extend scheduling techniques that have been developed for real-time systems [19] 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 [12]. Unlike a real-time operating system (RTOS), event-driven operating systems generally do not provide real-time scheduling guarantees, priority-based scheduling, or resource reservation functionality. Without such support, many schedulability analysis techniques cannot be effectively employed. For example, in the absence of task preemption and priority-based scheduling, unnecessarily conservative assumptions must be used to guarantee correctness in the general case.

We propose an actor-based modeling approach that allows WSAN application programmers to assess the performance and functional behavior of their code throughout the design and implementation phases. The developed models are analyzed using model checking to determine the parameter values resulting in the highest system efficiency. Note that our use of model checking is similar to the work of Jorgerden et al. who use it to maximize the life-time of batteries in embedded systems [14].

We represent a WSAN application as a collection of *actors* [2]. The model can be incrementally extended and refined during the application design process, adding new interactions and scheduling constraints. We use Timed Rebeca [25] as the modeling language and its model checking tool Afra [1, 15] for analysis of WSAN applications. Timed Rebeca is a high-level actor-based language capable of representing functionality and timing behavior at an abstract level. Afra supports modeling and analysis of both of Rebeca and Timed Rebeca models; we use the timed model checking engine. Afra uses the concept of Floating Time Transition System (FTTS) [15] for the analysis of Timed Rebeca models. FTTS significantly reduces the state space that needs to be searched. The idea is to focus on event-based properties while relaxing the constraint requiring the generation of states where all the actors are synchronized. As the examples in [16] suggest, this approach can reduce the size of the state space by 50 to 90%. Using FTTS 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 [18]. This system has been implemented on the Imote2 wireless sensor platform, and used in several long-term development of several highway and railroad bridges [29]. 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 be-

tween the CPU, sensor and radio within each node, as well as interactions among the 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 the application we consider, 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 [18], supporting a higher sampling rate or a larger number of nodes without violating schedulability constraints.
Contributions. This paper makes the following contributions:

- We show how a WSAN application may be modeled naturally as a system of actors. The abstraction and modularity of the actor model makes the approach scalable.
- We present a real-world case study that illustrates the effectiveness of our approach for a real WSAN application.
- We show how model checking toolsets can be used for an efficient schedulability analysis of WSAN application. Our case study shows we can compare the effects of different communication protocols on system performance.

2 Preliminaries

A WSAN application is a distributed system with multiple sensor nodes, each comprised of the independent concurrent entities: CPU, sensor, radio system, and bridged together via a wireless communication device which uses a transmission control protocol. Interactions between these components, both within a node and across nodes, are concurrent and asynchronous. Moreover, WSAN applications are sensitive to timing, with soft deadlines at each step of the process needed to ensure correct and efficient operation.

Due to performance requirements, and latencies of operations on sensor nodes, sensing, data processing, and communication processes must be coordinated. In particular, once a sample is acquired from a sensor, its corresponding radio transmission activities must be performed. Concurrently, data processing tasks–such as compensating sensor data for the effects of temperature changes–must be executed. Moreover, the timing of radio transmissions from different nodes must be coordinated using a communication protocol.

2.1 The Actor Model of WSAN Applications

The 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 [11]. Subsequently, the actor model developed as a model of concurrent computation for open distributed systems where *actors* are the concurrently executing entities [2]. One way to think of actors is as a service oriented framework: each actor provides services that may be requested via messages from other actors. A message is buffered until the provider is ready to execute the message. As a result of processing a message, an actor may send messages to other actors, and to itself. Extensions of the actor model have been used for real-time systems, in particular: RT-synchronizer [24], real-time Creol [6], and Timed Rebeca [25].

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. Observe that common tasks such as sample acquisition, sample processing, and radio transmission are periodic and have well-known or easily measurable periods. This makes analysis of worst-case execution times feasible. However, because of the event-triggered nature of applications, initial offsets between the tasks are variable.

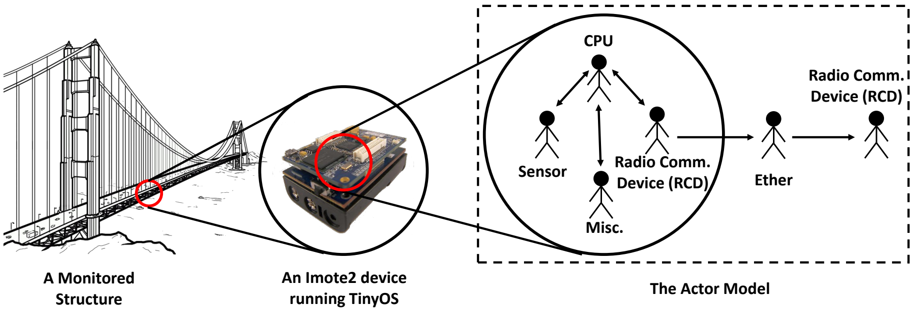


Fig. 1. Modeling the behavior of a WSAN application in its real-world installation in the actor model

We represent components of each WSAN node capable of independent action as an actor. Specifically, as shown in Figure 1, a sensor node is modeled using four actors: `Sensor` (for the data acquisition) `CPU` (processor), `RCD` (a radio communication device) and `Misc` (carrying out miscellaneous tasks unrelated to sensing or communication). `Sensor` collects data and send it to `CPU` for further data processing. Meanwhile, `CPU` may respond to messages from `Misc` by carrying out other computations. The processed data is sent to `RCD` to forward it to a data collector node actor. We model the communication medium as an actor (`Ether`) and the receiver node also by the actor `RCD`. Using the actor `Ether` facilitates modularity: specifically, implementation of the Media Access Control (MAC) level details of communication protocols is localized, making it is easy to replace component sub-models for modeling different communication protocols without significantly impacting the remainder of the model. During the application design phase, different components, services, and protocols may be considered. For example, TDMA [8] as a MAC-level communication protocol may be replaced by B-MAC [23] with minimal changes.

Although schedulability analysis of WSAN applications can be challenging in the absence of a real-time scheduler, we reduce the problem of checking for deadline violations 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.

2.2 Timed Rebeca and the Model Checking Toolset

A Timed Rebecca (TR) model consists of reactive classes and a main program which instantiates actors (called *rebecs* in TR). As usual, actors have an encapsulated state, a local time, and their own thread of control. Each actor contains a set of state variables, methods and a set of actors it knows. An actor may only send messages to actor that it knows. Message passing is implemented by method calls: calling a method of an actor (target) results in sending a message to the target. Each actor has a message bag in which arriving messages may be buffered; the maximum capacity of the bag is defined by the modeler.

Timing behavior in TR is represented using three timing primitives: `delay`, `after`, and `deadline`. A `delay` term models the passing of time for an actor. The primitives `after` and `deadline` can be used in conjunction with a message send: `after` n indicates it takes n time units for the message to be delivered to its receiver; `deadline` n indicates that if the message is not taken in n time units, it should be purged from the receiver's bag.

Afra 1.0 supports model checking of Rebeca models against LTL and CTL properties. Afra 2.0 supports deadlock detection and schedulability analysis of TR models; we use Afra 2.0 in this work. TR and Afra toolset have previously been used to model and analyze realtime actor based models such as routing algorithms and scheduling policies in NoC (Network on Chip) designs [27, 26].

3 Schedulability Analysis of a Stand-Alone Node

We now illustrate our approach using a node-level TR model of a WSAN application to check for possible deadline violations. Specifically, 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. Then, we show how the specification of a node-level model can be naturally extended to network-wide specifications.

Following the mapping in Figure 1, the TR model for the four different *reactive classes* in Fig. 2 through Fig. 4.

As shown in Fig. 2, the maximum capacity of the message bag of `Sensor` is set to 10, the only actor `Sensor` knows about is of type `CPU` (line 4), and `Sensor` does not have any state variables (line 5). The behavior of `Sensor` is to periodically acquire data and send it to `CPU`. `Sensor` is implemented using a message server `sensorLoop` (lines 13-17) which sends the acquired data to `CPU` (line 15). 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 that there is a

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

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

  Sensor() {
    self.sensorFirst();
  }
  msgsrv sensorFirst() {
    self.sensorLoop() after(?(10, 20, 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; // ms
env int miscTaskDelay = 10; // ms
env int bufferSize = 3; // samples

reactiveclass CPU(10) {
  knownrebecs { RCD 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`

nondeterministic initial offset after which the data acquisition becomes a periodic task. To represent this property, `Sensor` which sends a `sendLoop` message to itself; the message is nondeterministically delivered after one of 10, 20, and 30 (line 11). After this random offset, a sensor's periodic behavior is initiated (line 13). Note that in line 1, the sampling rate is defined as a constant. A similar approach is used in the implementation of the `Misc` reactive class.

The behavior of `CPU` as the target of `Sensor` and `Misc` events is more complicated (Figure 3). Upon receiving a `miscEvent`, `CPU` waits for `miscTaskDelay` units of time; this represents computation cycles consumed by miscellaneous tasks. Similarly, after receiving the `sensorEvent` message from `Sensor`, `CPU` waits for `sensorTaskDelay` units of time; this represents cycles required for intra-node data processing. Data must be packed in a packet of a specified `bufferSize`. The number of collected samples + 1 is computed (line 16) and when the threshold is reached (line 17), `CPU` asks `senderDevice`, to send the collected data in one packet (line 18). As this is a node-level model, communication between nodes is omitted. The behavior of `RCD` is limited to waiting for some amount of time (line 6); this represents the sending time of a packet.

```
1 env int OnePacketTransmissionTime = 7; // ms
2
3 reactiveclass RCD (2) {
4   RCD() { }
5   msgsrv send(RCD receiverDevice, byte numberOfPackets) {
6     delay(OnePacketTransmissionTime);
7   }
8 }
```

Fig. 4. The node-level implementation of `RCD`

Note that computation times (`delay`'s) depend on the low-level aspects of the system and are application-independent; they can be measured before the application design. For schedulability analysis, we set the `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 a network model requires that the wireless communication medium `Ether` to be specified in order to model the communication protocol it supports. Then both the node-level and multi-node models must be considered. Recall that nodes in the multi-node model periodically send their data to an aggregator node (Fig. 1). The sending process is controlled by a wireless network communication protocol. The reactive class of `Ether` (Fig. 5) has three message servers: these are responsible for sending the status of the medium, broadcasting data, and resetting the condition of the medium after a successful transmission. Broadcasting data takes place by sending data to a `RCD` which is addressed by the `receiverDevice` variable. So, we can easily examine the status of the `Ether` 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 `Ether` (lines 15 to 26). Before the start of broadcasting, the `Ether` status is checked (line 16) and data-collision error is raised in case of two simultaneous broadcasts (line 24). With a successful data broadcast, `Ether` sends an acknowledgment to itself (line 19) and the sender (line 20), and informs the receiver of the number of packets sent to it (line 21). In addition to the functional requirements of `Ether`, there may be non-functional requirements. For example, the Imote2 radio offers a theoretical maximum transfer speed of 250 kbps. When considering only the useful data payload (goodput), this is reduced to about 125 kbps.

We now extend `RCD` to support communication protocols. Fig. 6 shows the model of TDMA protocol implementation. 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 a node has data available to transmit during its alloted time slot, it may be sent immediately. Otherwise, packet sending is delayed until its next transmission slot. The periodic behavior of TDMA slot is handled by `handleTDMASlot` message server which sets and unsets `inActivePeriod` to show that whether the node is in its alloted time slot. Upon entering into it's slot, a device checks for pending data to send (line 31) and schedules `handleTDMASlot` message to leave the slot (line 30). On the other hand, when `CPU` sends a packet (message) to a `RCD`, the message is added to the other pending packets which are waiting for the next alloted time slot. `tdmaSlotSize` is the predefined size of the tdma slots, and `currentMessageWaitingTime` is the waiting time of this message in the bag of its receiver.

For the sake of simplicity, the details of `RCD` are omitted in Fig. 6. The complete source code (which implements the 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 the TDMA time slot length, to prevent such situations.

```
env int OnePacketTT = 7; // ms (transmission time)

reactiveclass Ether(5) {
  statevars {
    RCD senderDevice, receiverDevice;
  }

  Ether() {
    senderDevice = null;
    receiverDevice = null;
  }
  msgsrv getStatus() {
    ((RCD)sender).receiveStatus(receiverDevice != null);
  }
  msgsrv broadcast(RCD receiver, int packetsNumber) {
    if(senderDevice == null) {
      senderDevice = (RCD)sender;
      receiverDevice = receiver;
      self.broadcastingIsCompleted() after(packetsNumber * OnePacketTT);
      ((RCD)sender).receiveResult(true) after(packetsNumber * OnePacketTT);
      receiver.receiveData(receiver, packetsNumber);
    } else {
      ((RCD)sender).receiveResult(false);
    }
  }
  msgsrv broadcastingIsCompleted() {
    senderDevice = null;
    receiverDevice = null;
  }
}
```

Fig. 5. Reactive class of the `Ether`

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.[1] 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 potential damages to the structure by measuring changes in strain and vibration response. SHM can also be employed with *structural control*, where it is fed into algorithms that control centralized or distributed control el-

[1] The TR 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 the Rebeca homepage [1].

```
env int OnePacketTT = 7; ms (transmission time)

reactiveclass RCD (3) {

  knownrebecs { Ether ether; }

  statevars {
    byte id;
    int slotSize;
    boolean inActivePeriod;

    int sendingPacketsNumber;
    RCD receiverDevice;
    boolean busyWithSending;
  }

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

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

Fig. 6. Reactive class of the RCD

ements 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 [18], 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 [13].

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 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 on up to 30 nodes—where 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 sensors. The accuracy of estimating safe limits for sampling and data transmission delays directly impacts the system's efficiency.

To illustrate the applicability of this work, we considered applications where achieving the highest possible sampling rate that does not result in any missed deadline is desired. This is a very common requirement in WSAN applications in the SHMC domain in particular. We begin by setting the value of `OnePacketTT` to 7ms (i.e., the maximum transmission time of this type of applications) and fixed the value of `sensorTaskDelay`, `miscPeriod`, and `miscTaskDelay` to some predefined values. In addition to the sampling rate, the number of nodes in the network and the packet size remain variable. By assuming different values for the number of nodes and the packet size, different maximum sampling rates are achieved, shown as a 3D surface in Figure 7. As shown in the figure, higher sampling rates are possible when the buffer size is set to a larger number (there is 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 7, the cases with the maximum buffer size (i.e., 9 data points) 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.

A server with Intel Xeon E5645 @ 2.40GHz CPUs and 50GB of RAM, running Red Hat 4.4.6-4 as the operating system was used as the model-checking host. We varied the size of the state space from < 500 to $> 140K$ states, resulting in model checking times ranging from 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 observed. For example, the largest state spaces correspond to configurations where `sensorTaskDelay`, `bufferSize`, and `numberOfNodes` are set to large values.

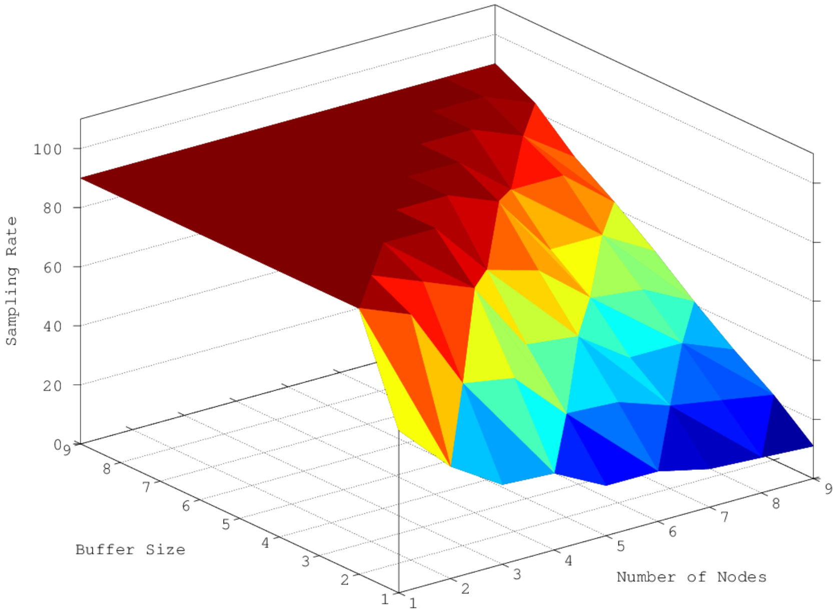


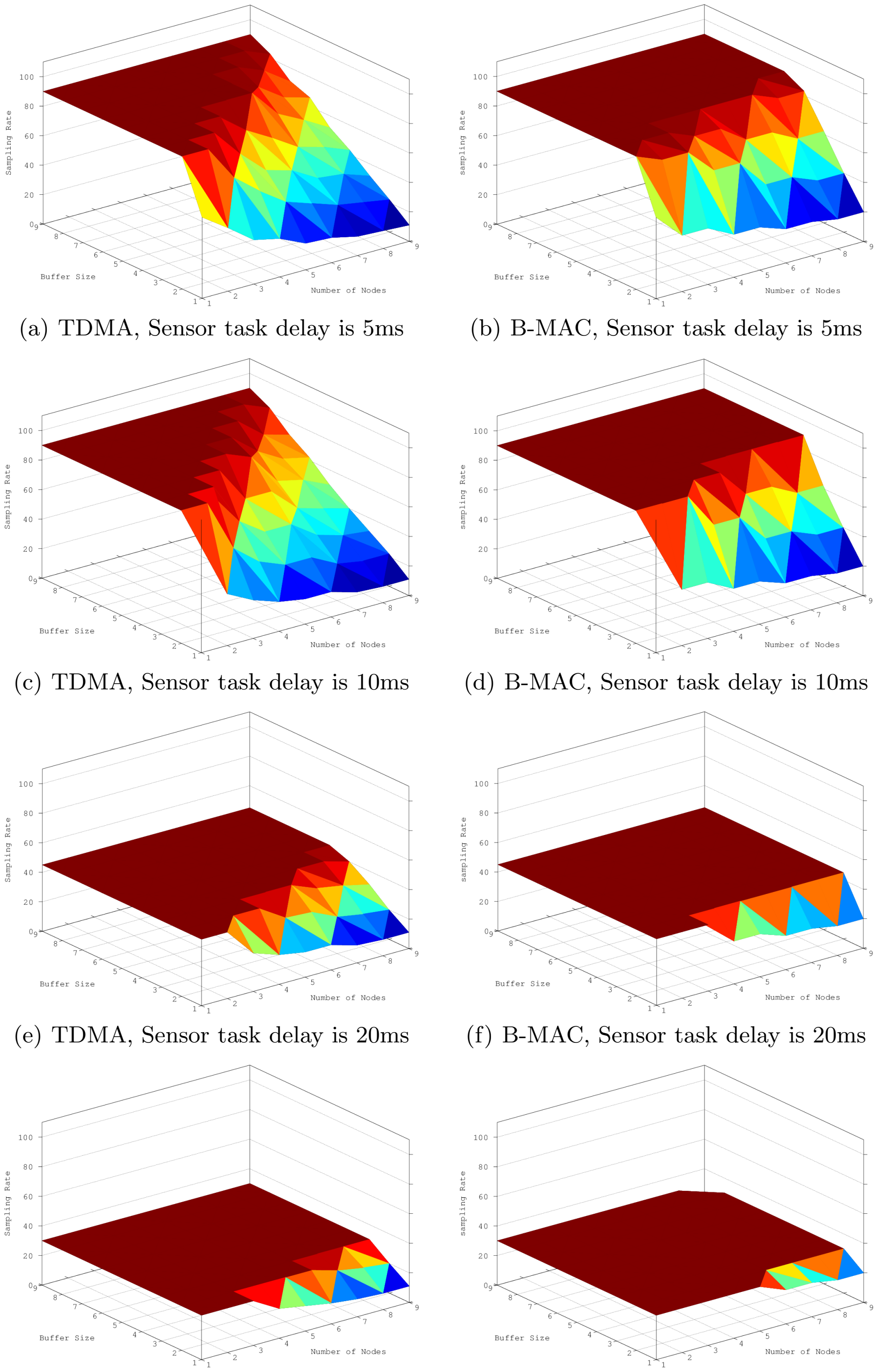
Fig. 7. The maximum sampling rate in case of using TDMA protocol and setting the value of `sensorTaskDelay` to 2ms

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 maximum sampling rates found for each configuration is depicted in Figure 8; they show that increasing the value of `sensorTaskDelay` as the representor of intra-node activities, decreases the sampling rate dramatically. They also show that using B-MAC results in achieving higher sampling rates in comparison to TDMA.

The parameters used in our analysis of configurations were determined through a real-world installation of an SHMC application. Our results show that the current manually-optimized installation can be tuned to an even more optimized one: by changing the configuration, the performance of the system can be safely improved by another 7% percent.

6 Related Work

Three different approaches have been used for analysis of WSANs: system simulation, analytical approach, and formal verification.



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

System Simulation. Simulation of WSAN applications is useful for their early design exploration. Simulation toolsets for WSANs have enabled modeling of networks [17], power consumption [28], and deployment environment [31]. Simulators can adequately estimate performance of systems and sometimes detect conditions which lead to deadline violations. But even extensive simulation does not guarantee that deadline misses will never occur in the future [5]. For WSAN applications with hard real-time requirements this is not satisfactory. Moroever, none of available simulators is suitable for the analysis WSAN application software.

Analytical Approach. A number of algorithms and heuristics have been suggested for schedulability analysis of real-time systems with periodic tasks and sporadic tasks with constraints, e.g. [20]. Although these classic techniques are efficient in analyzing schedulability of real-time systems with periodic tasks and sporadic tasks, their lack of ability to model random tasks make them inappropriate for WSAN applications.

Formal Verification. Real-time model checking is an attractive approach for schedulability analysis with guarantees [5]. Model checking tools systematically check whether a model satisfies a given property [4]. 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. suggest an extension of timed automata to support schedulability analysis of real-time systems with random tasks [21]. 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 [10].

TIMES [3] is a toolset which is implemented based on the approach of Feresman et al. [9] 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 [7] 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 lead to a state space explosion even for small size examples.

Real-Time Maude is used in [22] for performance estimation and model checking of WSAN algorithms. The approach supports modeling of many details such as communication range and energy use. The approach requires some knowledge of rewrite logic. Our tool may be easier to use by engineers unfamiliar with rewriting logic: our language extends straight-forward C-like syntax with actor concurrency constructs and primitives for sensing and radio communication. This requires no formal methods experience from the WSAN application programmer, as the language and structure of the model closely mirror those of the real application.

7 Conclusion

We have shown one of the applications of real-time model checking method in analyzing schedulability and resource utilization of 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 developer, our approach provides a much more accurate view of an WSAN application's behavior and its interaction with the operating system and distributed middle-ware 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. This would greatly reduce development time as many potential problems with scheduling and resource utilization may be identified early.

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 cases. Development of a probabilistic extension is currently underway.

Acknowledgments

The work on this paper has been supported in part by the project "Timed Asynchronous Reactive Objects in Distributed Systems: TARO" (nr. 110020021) of the Icelandic Research Fund, by Air Force Research Laboratory and the Air Force Office of Scientific Research under agreement number FA8750-11-2-0084, and by National Science Foundation under grant number CCF-1438982. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The authors acknowledge the helpful comments by the anonymous referees and by Karl Palmskog.

References

1. Rebeca Formal Modeling Language. `http://www.rebeca-lang.org/`.
2. Gul A. Agha. *ACTORS - a model of concurrent computation in distributed systems*. MIT Press series in artificial intelligence. MIT Press, 1990.
3. 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.

4. Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking.* MIT Press, 1999.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. Illinois SHM Services Toolsuite. `http://shm.cs.illinois.edu/software.html`.
14. Marijn R. Jongerden, Alexandru Mereacre, Henrik C. Bohnenkamp, Boudewijn R. Haverkort, and Joost-Pieter Katoen. Computing optimal schedules for battery usage in embedded systems. *IEEE Trans. Industrial Informatics*, 6(3):276–286, 2010.
15. 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.
16. 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.
17. 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.

18. 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.
19. Giuseppe Lipari and Giorgio Buttazzo. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *Journal of Systems Architecture*, 46(4):327–338, 2000.
20. Jane W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.
21. 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.
22. 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.
23. Joseph Polastre, Jason L. Hill, and David E. Culler. Versatile low power media access for wireless sensor networks. In Stankovic et al. [30], pages 95–107.
24. 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.
25. 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.
26. Zeinab Sharifi, Siamak Mohammadi, and Marjan. Sirjani. Comparison of NoC Routing Algorithms Using Formal Methods. To be published in proceedings of PDPTA'13, 2013.
27. 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.
28. 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. [30], pages 188–200.
29. Billie F. Spencer Jr., Hongki Jo, Kirill Mechitov, Jian Li, Sung-Han Sim, Robin. Kim, Soojin Cho, Lauren Linderman, Parya Moinzadeh, Ryan Giles, and Gul Agha. Recent advances in wireless smart sensors for multi-scale monitoring and control of civil infrastructure. *Journal of Civil Structural Health Monitoring*, pages 1–25, 2015.
30. 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.
31. Sameer Sundresh, WooYoung Kim, and Gul Agha. Sens: A sensor, environment and network simulator. In *Proceedings 37th Annual Simulation Symposium (ANSS-37 2004), 18-22 April 2004, Arlington, VA, USA*, pages 221–228. IEEE Computer Society, 2004.