Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Modeling and Analyzing Real-Time Wireless Sensor and Actuator Networks Using Actors and Model Checking

Ehsan Khamespanah [1,2], **Marjan Sirjani** [3,2], **Kirill Mechitov** [4], **Gul Agha** [4]

[1] University of Tehran, School of ECE, Tehran, Iran
[2] Reykjavik University, School of Computer Science and CRESS, Reykjavik, Iceland
[3] Mälardalen University, School of IDT, Västerås, Sweden
[4] University of Illinois at Urbana-Champaign, OSL, Champaign, United States

The date of receipt and acceptance will be inserted by the editor

Abstract. Programmers often use informal worst-case analysis and debugging to ensure that schedulers 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 behaviors are specified using a Java-based actor language extended with operators for real-time scheduling and delay representation. We show how the abstraction mechanism and the compositionality of actors in 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.

Key words: 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 [21], 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 [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 [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 developed codes 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.

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 to actors. We use Timed Rebeca [28] as the modeling language and its model checking tool Afra [1,18] 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 Rebeca and Timed Rebeca models; we use the timed model checking engine. Afra uses the concept of Floating Time Transition System (FTTS) [18] 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 [19] suggest, this approach can reduce the size of the state spaces by 50 to 90% in comparison with the state spaces which are generated based on the concept of timed transition system. Using FTTS fits with the analyzing the schedulability of events that we are interested in WSAN applications.

In addition to the schedulability analysis, Timed Rebeca provides TCTL model checking facilities for analyzing timed actors against more complicated properties. Later, we will show how TCTL model checking can be used to guaranty the minimum utilization of resources of WSAN applications. In the developed case study, figuring out the minimum utilization of the communication medium (Ether) was the property we were 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 used in several long-term development of several highway and railroad bridges [35]. 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 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 [21], supporting a higher sampling rate or a larger number of nodes without violating schedulability constraints.

Contributions. This paper is an extended version of the workshop paper [17]. In [17], we presented our preliminary results on modeling and analyzing WSAN applications using model checking toolset of Timed Rebeca. Apart from adding more detail about the proposed approaches, this paper extends [17] as follows:

- We show how WSAN applications can be modeled by event graphs and how the elements of an event graph are associated with actors (Sections 2.2).
- We use TCTL model checking engine of Timed Rebeca for the analysis of WSAN applications (Section 6.2).
- We demonstrate the detailed Timed Rebeca source code of B-MAC protocol to illustrate how naturally communication protocols can be modeled using our approach (Section 4).
- We show how the approach can be generalized to support modeling and analysis of a wide range of WSAN applications (Section 5).

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 entities, 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 that are required to ensure correct and efficient operation.

Due to the performance requirements and latencies of operations on sensor nodes, coordination among sensing, data processing, and communication activities is required. 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–such as compensating sensor data for the effects of changes in the temperature–must be executed. Moreover, the timing of radio transmissions from different nodes must be coordinated using a communication protocol.

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 for this class of problems, and it is the approach we explore in this paper.

2.1 Event Graph

At the first step of modeling WSAN applications, we need to explain *event graphs* in which a highly abstracted view of scheduling events can be depicted. Event graphs have a single type of node and two types of edges, i.e.

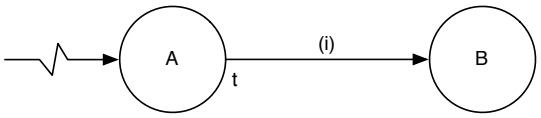


Fig. 1. An example of an event graph

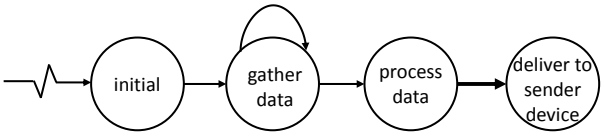


Fig. 2. The event graph of a WSAN sensor behavior

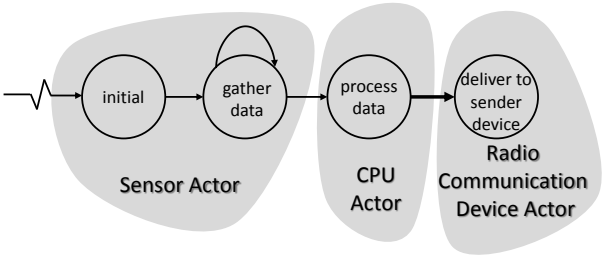


Fig. 3. How events of a WSAN sensor are associated with actors

jagged and ordinary edges. The nodes represent events in a system. Edges correspond to the scheduling of other events [4]. In this graph, the initial event is shown by jagged edges. Edges can optionally be associated with a boolean condition for scheduling an event and/or a time delay which means that an event will be scheduled after the delay. Figure 1 shows an example of an event graph where the event B is scheduled by the event A if condition *(i)* is evaluated to true, at t units of time later than the current time.

Event graphs are widely used in the engineering community for simulation and analysis of complex systems. More specifically, they are used to graphically represent discrete-event simulation models. We use event graphs in this paper only to give a highly abstracted view of how events are scheduled in our case studies. We adopt an alternative notation where conditional edges are thicker, even if the conditions are not specified.

2.2 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 [27], real-time Creol [7], and Timed Rebeca [28].

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.

At the first step of proposing an actor model for the WSAN applications, we need to have a look into the interaction of the components and the events which are triggered and served by them. Based on the specification of the WSAN applications, there are many nodes which have the role of data acquisition and data transmission. For data acquisition, a node has a set of sensors which periodically acquire data from the environment and send the data to the processing unit of the node. The processing unit is responsible for validating the data and storing it into an internal buffer. Upon receiving enough data, the processor unit sends the data to the radio communication unit. The radio communication unit tries to send data via wireless medium, considering a predefined communication protocol. The event graph of this model is depicted in Figure 2. The majority of WSAN applications can be modeled using this graph; although, minor modifications maybe needed.

We split up the event handlers of the events of Figure 2 into three different actors, depicted in Figure 3 and add one additional actor for carrying out miscellaneous tasks unrelated to sensing or communication. This additional actor is necessary for making the model close to its real configuration. The three actors are called `Sensor` (for the data acquisition), `CPU` (processing unit), `RCD` (a radio communication device) and the additional actor is called `Misc`. `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.

Composing the collection of sensor nodes to develop a complete WSAN application requires that the wireless communication medium be specified and a communication protocol is implemented in radio communication devices. Note that the process of sending a packet is controlled by a wireless network communication protocol. 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,

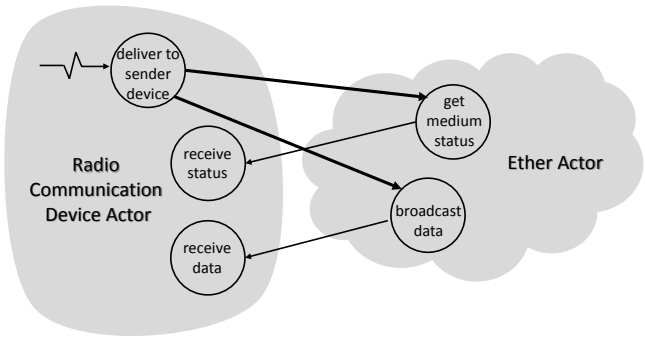


Fig. 4. How events of wireless communication mechanism are associated with actors

implementation of the Media Access Control (MAC) level details of communication protocols is localized. As a result, different implementation of communication protocols can be replaced without significantly impacting the remainder of the model. As shown in Figure 4, `Ether` serves events for receiving the status of the medium and broadcasting data. For the development of different communication protocols, different combination of these two events can be triggered to model the behavior of the protocols.

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 [26] with minimal changes. In a nutshell, using the mentioned association of events with actors, a given WSAN application is modeled by actors as shown in Figure 5.

2.3 Timed Rebeca and the Model Checking Toolset

A Timed Rebeca [28] model (as the real-time extension of the Rebeca modeling language [34,33,32]) consists of a number of *reactive classes*, each describing the type of a certain number of *actors* (called *rebecs*)[1]. Each reactive class declares the size of its message bag and a set of *state variables*. The local state of each actor is defined by the values of its state variables and the contents of its message bag. Following the actor model, communication in the Timed Rebeca models takes place by asynchronous message passing among actors. Each actor has a set of *known rebecs* to which it can send messages. Reactive classes of a Timed Rebeca model may have some constructors. Constructors have the same name as the declaring reactive class and do not have a return value. They are responsible for initializing the actor's state variables and putting initially needed messages in the bag of that actor. A properly written constructor leaves the resulting actor in a valid state. Reactive classes declare the messages to which they can respond. The way an actor responds to a message is specified in a *message server*. The state of an actor can change during the executing of its message servers through assignment statements, makes decisions through conditional statements, communicates with other actors by sending messages, and performs periodic behavior by sending messages to itself. Since communication is asynchronous, each actor has a *message bag* from which it takes the next incoming message. The ordering of the messages in a message bag is based on the arrival times of messages. An actor takes the first message from its message bag, executes its corresponding message server in an isolated environment, takes the next message (or waits for the next message to arrive) and so on. A message server may have a *nondeterministic assignment* statement which is used to model the nondeterminism in the behavior of a message server. Finally, the `main` block is used to instantiate the actors of the model. Note that Timed Rebeca does not support dynamic actor creation, so all the actors of a model must be defined in the main block.

Timed Rebeca adds three primitives to Rebeca to address timing issues: *delay*, *deadline* and *after*. A *delay* statement models the passage of time for an actor during execution of a message server. Note that all other statements of Timed Rebeca are assumed to execute instantaneously. The keywords *after* and *deadline* are used in conjunction with a method call. The term `after(n)` indicates that it takes n units of time for a message to be delivered to its receiver. The term `deadline(n)` expresses that if the message is not taken in n units of time, it will be purged from the receiver's message bag automatically.

A Rebeca model may contain some private methods. These methods can not be called from the other actors and used to make the model of a reactive class more modular. The definition of a method starts with the type of its return value (instead of the `msgsrv` keyword) and its body is the same as the body of a message server.

The model checking toolset of Timed Rebeca models is called Afra. Afra 1.0 supports model checking of Rebeca models against LTL and CTL properties. Afra 2.0 supports deadlock detection and schedulability analysis of Timed Rebeca models; we use Afra 2.0 in this work for the schedulability analysis of WSAN applications. We also benefit from TCTL model checking toolset of Timed Rebeca, which will be integrated to Afra. Timed Rebeca 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 [30,29].

3 Schedulability Analysis of a Stand-Alone Node

We now illustrate our approach using a node-level Timed Rebeca model of a WSAN application to check for possible deadline violations. Specifically, by changing the

[1] In this paper we use rebec and actor interchangeably.

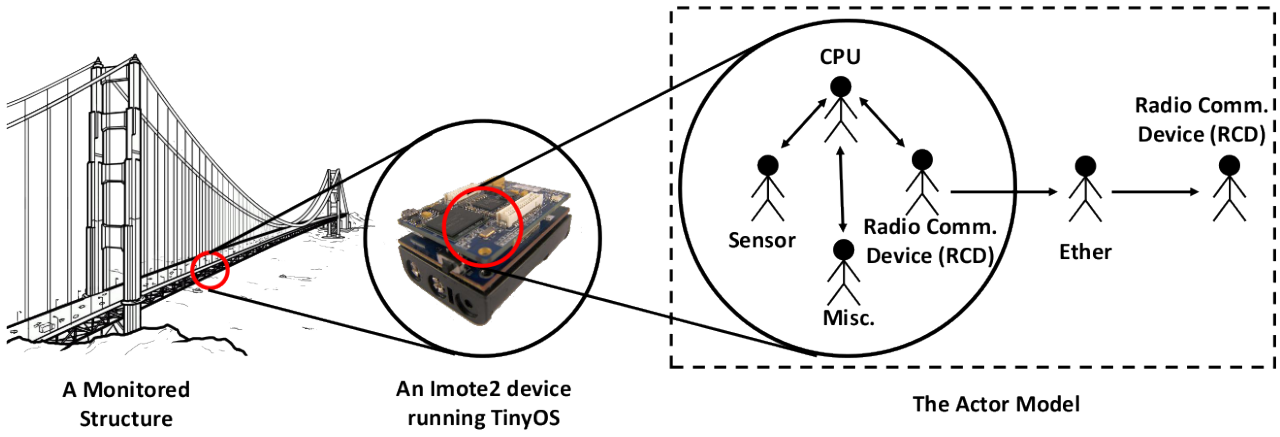


Fig. 5. Modeling the behavior of a WSAN application in its real-world installation in the actor model

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 of Figure 5, the Timed Rebeca model for the four different *reactive classes* is presented in Listing 1 through Listing 3. As shown in Listing 1, the maximum capacity of the message bag of `Sensor` is set to 10, the only actor which `Sensor` knows about is of type `CPU` (line 4), and `Sensor` does not have any state variables. The behavior of `Sensor` is to acquire data and send it to `CPU` periodically. `Sensor` is implemented using a message server `sensorLoop` (lines 10-14) which sends the acquired data to `CPU` (line 12). The sent data must be served before the start time of the next period, specified by the value of `period` as the parameter of `deadline`. Recall that there is a 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 8).

Listing 1. The Timed Rebeca implementation of `Sensor` reactive class

```
1 env int samplingRate = 25; // Hz
2
3 reactiveclass Sensor(10) {
4   knownrebecs { CPU cpu; }
5
6   Sensor() { self.sensorFirst(); }
7   msgsrv sensorFirst() {
8     self.sensorLoop() after(?(10, 20, 30)); // ms
9   }
10  msgsrv sensorLoop() {
11    int period = 1000 / samplingRate;
12    cpu.sensorEvent() deadline(period);
13    self.sensorLoop() after(period);
14  }
15 }
```

After this random offset, the sensor's periodic behavior is initiated. 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 (Listing 2). 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).

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

```
1 env int sensorTaskDelay = 2; // ms
2 env int miscTaskDelay = 10; // ms
3 env int bufferSize = 3; // samples
4
5 reactiveclass CPU(10) {
6   knownrebecs {RCD senderDevice, receiverDevice;}
7   statevars { int collectedSamplesCounter; }
8
9   CPU() { collectedSamplesCounter = 0; }
10
11  msgsrv miscEvent() {
12    delay(miscTaskDelay);
13  }
14  msgsrv sensorEvent() {
15    delay(sensorTaskDelay);
16    collectedSamplesCounter += 1;
17    if (collectedSamplesCounter == bufferSize) {
18      senderDevice.send(receiverDevice, 1);
19      collectedSamplesCounter = 0;
20    }
21  }
22 }
```

As this is a node-level model, communication between nodes is omitted and the behavior of RCD is limited to waiting for some amount of time (line 6 of Listing 3); this represents the sending time of a packet.

Listing 3. The node-level implementation of RCD

```
1 env int OnePacketTT = 7; // ms(transmission time)
2
3 reactiveclass RCD (2) {
4   RCD() { }
5   msgsrv send(RCD receiver, byte numOfPackets) {
6     delay(OnePacketTT * numOfPackets);
7   }
8 }
```

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

Composing the models of stand-alone nodes to have a multi-node model requires that the wireless communication medium Ether be specified and a communication protocol is implemented for radio communication devices. Recall that nodes in the multi-node model periodically send their data to an aggregator node (Listing 5). The sending process is controlled by a wireless network communication protocol. The reactive class of Ether (Listing 4) 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 results in setting the values of senderDevice and receiverDevice to their corresponding actors. So, the status of the 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 13). This way, upon sending data successfully, the value of receiverDevice and senderDevice must be set to null to show that the transmission is completed (lines 30 and 31). Data broadcasting is the main behavior of Ether (lines 16 to 28). Before the start of broadcasting, the Ether status is checked (line 17) and data-collision error is raised in the case of two simultaneous broadcasts (line 26). With a successful data broadcast, Ether sends an acknowledgment to itself (line 20) and the sender (line 22), and informs the receiver of the number of packets sent to it (line 24). 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.

Listing 4. The Timed Rebeca implementation of Ether reactive class

```
1 env int OnePacketTT = 7; // ms(transmission time)
2
3 reactiveclass Ether(5) {
4   statevars {
5     RCD senderDevice, receiverDevice;
6   }
7
8   Ether() {
9     senderDevice = null;
10    receiverDevice = null;
11  }
12  msgsrv getStatus() {
13    ((RCD)sender).receiveStatus(
14      receiverDevice != null);
15  }
16  msgsrv broadcast(RCD receiver, int packets) {
17    if(senderDevice == null) {
18      senderDevice = (RCD)sender;
19      receiverDevice = receiver;
20      self.broadcastingIsCompleted()
21        after(packets * OnePacketTT);
22      ((RCD)sender).receiveResult(true)
23        after(packets * OnePacketTT);
24      receiver.receiveData(receiver, packets);
25    } else {
26      ((RCD)sender).receiveResult(false);
27    }
28  }
29  msgsrv broadcastingIsCompleted() {
30    senderDevice = null;
31    receiverDevice = null;
32  }
33 }
```

We now extend RCD to support communication protocols. Listing 5 shows the Timed Rebeca implementation model of TDMA protocol. 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 32) and schedules handleTDMASlot message to leave the slot (line 31). 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, some details of RCD are omitted in Listing 5. The complete source code (which implements the B-MAC protocol) is available on the Rebeca web page [1].

Listing 5. The Timed Rebeca implementation of TDMA protocol in RCD

```
1 env int OnePacketTT = 7; ms (transmission time)
2
3 reactiveclass RCD (10) {
4   knownrebecs { WirelessMedium medium; }
5   statevars {
6     byte id;
7     int slotSize, sendingData;
8     boolean busyWithSending, inActivePeriod;
9     RCD receiverDevice;
10  }
11
12  RCD(byte myId) {
13    id = myId;
14    inActivePeriod = false;
15    sendingData = 0;
16    busyWithSending = false;
17    receiverDevice = null;
18    ...
19  }
20  msgsrv send(RCD receiver, int data, int
        packetsNumber) {
21    assertion(receiverDevice == null);
22    receiverDevice = receiver;
23    sendingData = data;
24    self.checkPendingData();
25  }
26  msgsrv handleTDMASlot() {
27    inActivePeriod = !inActivePeriod;
28    if(inActivePeriod) {
29      int remainedTime = tmdaSlotSize -
            currentMessageWaitingTime;
30      assertion(remainedTime > 0);
31      self.handleTDMASlot() after(remainedTime);
32      self.checkPendingData();
33    } else {
34      self.handleTDMASlot() after((tmdaSlotSize *
            (numberOfNodes - 1))-
            currentMessageWaitingTime);
35    }
36  }
37  msgsrv checkPendingData() { ... }
38  msgsrv receiveStatus(boolean result) { ... }
39  msgsrv receiveResult(boolean result) { ... }
40  msgsrv receiveData(RCD receiver, int data, int
        receivingPacketsNumber) {
41    if (receiver == self) {
42        delay(receivingPacketsNumber *
              OnePacketTransmissionTime);
43    }
44  }
45 }
```

Using TDMA, an execution period is defined and each node in the network has one or more time slots to transmit a packet or a series of packets in an execution period. If a node has data available to transmit during its allotted time slot, it may be sent immediately. Otherwise, packet sending is delayed until its next allotted time slot. This way, the packet transmission of one sensor node does not interfere with the other sensor nodes. Having more sensor nodes only results in having shorter time slots, so the presence of sensor nodes can be abstracted and modeled as making time slots shorter. Using this abstraction, compositional verification of WSAN applications against schedulability and deadlock-freedom properties become feasible as only one node which is in communication with the central node has to be considered for networks in any size.

B-MAC protocol is designed for low power Ad-Hoc networks in which some sender nodes send data to a receiver. Like the other low power protocols, B-MAC uses periodically sleep/wakeup cycles. During wakeup times, the node listens for incoming data transmissions. If there is no data received, the listen state is interrupted after passing a predefined duration. Otherwise, the node stays in listen state for complete data transmission. The sleep periods of nodes may differ, makes B-MAC an asynchronous communication protocol. When a node wants to send, it turns on the radio and starts sending an announcement. This announcement is long enough to make sure that it has overlap with the wakeup time of the data receiver. Afterwards the sender transmits data to the target address and starts sending data. In order to reduce the amount of needed energy, clear channel assessment is used with the aim of better separation between signals and noise on the channel. B-MAC has an application interface for flexible configuring parameters. A good value for this sometimes depends on the use case, so, this can be adjusted by a higher layer application.

We now extend RCD to implement B-MAC protocols, depicted in Listing 6. In contrast to TDMA, in B-MAC RCD tries to detect free channel status and send data upon receiving a request from CPU (line 20). In the case of detecting free channel, the data is sent immediately (line 24). This way, collision may occurs; so, RCD has to wait for some amount of time and resend data (line 23). B-MAC protocol does not need complicated and expensive synchronization methods. It also avoids data fragmentation. So, it would be more complicated to coordinate long messages and B-MAC expects short messages, which is common for informations of WSAN nodes.

Based on this fact, the presence of sensor nodes can be abstracted and modeled as the possible number of collisions before a data communication is performed successfully. Using this abstraction, efficient verification of WSAN applications become feasible as only one sensor node which is in communication with the central node has to be considered for networks in any size. Any data transmission of this sensor node may encounter a collision. The maximum number of the collisions is the number of sensor nodes in the model. So, in the Re-

beca code for RCD, for each data transmission we have a non-deterministic choice between a successful transmission or a collision. During model checking, in the case of collision, data transmission with zero, one, ..., up to n collisions are considered where n is the number of sensor nodes. The Timed Rebeca model of this protocol is available on the Rebeca web page [1].

Listing 6. The Timed Rebeca implementation of B-MAC protocol in RCD

```
1 env int OnePacketTT = 7; ms (transmission time)
2
3 reactiveclass RCD (10) {
4   knownrebecs { WirelessMedium medium; }
5   statevars {
6     byte id;
7     int sendingData;
8     RCD receiverDevice;
9   }
10
11  RCD(byte myId) {
12    id = myId;
13    sendingData = 0;
14    receiverDevice = null;
15  }
16  msgsrv send(RCD receiver, int data, int
        packetsNumber) {
17    assertion(receiverDevice == null);
18    receiverDevice = receiver;
19    sendingData = data;
20    medium.getStatus();
21  }
22  msgsrv receiveStatus(boolean result) {
23    delay((numberOfNodes/2) * (OnePacketTT + 1));
24    medium.broadcast(receiverDevice,sendingData,
          packetsNumber);
25    delay(OnePacketTT * packetsNumber);
26  }
27  msgsrv receiveResult(boolean result) { ... }
28  msgsrv receiveData(RCD receiver, int data, int
        receivingPacketsNumber) { ... }
29 }
```

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 or reducing the sampling rate, to prevent such situations.

5 Generalization of the Approach for Any WSAN Application

Here, we summarize the modeling approach and describe the way of extending it to make the approach applicable for the other WSAN applications. It is noteworthy that the actor-based approach is aligned with the structure of WSAN applications with different types of behaviors. Loosely coupled actors as the units of concurrency, with asynchronous message passing, and event-driven computation, are natural candidates for modeling such systems. The semantic gap between the model and the real-world system that has to be modeled is small, this so-called fidelity of the model to the system makes modeling easier and also makes the model easy to understand. Also, the possibility of building an understandable model with the least needed effort shows the usability of the model.

There is a simple rule for mapping a WSAN application to the actor model. Each entity in the WSAN application that is running concurrently, and is serving or creating events, has to be mapped into an actor. Therefore, two events which are served concurrently will be served by two different actors. In contrast, if there are two triggered events which are sent to the same entity and are served sequentially, then this entity is mapped to an actor which serves both events.

The simplest scenario is when there is no concurrent activity within each node in a WSAN application. In this scenario, the network communication among nodes are directly mapped to asynchronous communication among actors, and intra-node activities of each node are mapped into the event handlers of the corresponding actor. In a more general case, we recognize five different actors in a WSAN application, Sensor, CPU, Misc, RCD, and Ether. Here, the events created by sensors and miscellaneous activities have to be served by CPU and then sent to other nodes in the network.

One may want to make the model even more general as there may be more than one sensor in each node (e.g. one for humidity and one for temperature measurements). In this case, two different actors are needed to model the concurrent (data acquisition and) event creation of sensors. Also, in the case of using multicore CPUs inside a node, more than one CPU actors have to be associated with a node and the incoming tasks from sensors and miscellaneous activities have to be dispatched among them. Note that handling a multicore CPU in a node requires developing a task scheduler which must be run on one of the cores and dispatches the incoming tasks to appropriate CPUs. A modeler may perform these activities using event graphs or associating events with actors directly.

In addition to the mapping of entities to actors, we also need to model different communication protocols among nodes. The current implementation of Ether, explained in Section 4, serves two events which facilitates modeling of any wireless node-to-node communication protocol. Basically, no modification is needed in this code for supporting other node-to-node communications protocols. However, one may want to develop a WSAN application in which a node broadcasts data to some other nodes. In this case, Ether must be extended to support

multi-node data broadcasting, multicasting, or anycasting.

6 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.[2] 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 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 [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.

[2] 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 the Rebeca homepage [1].

6.1 Finding the Maximum Sampling Rate

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 6. 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 6, 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, nonlinearly 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.

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.

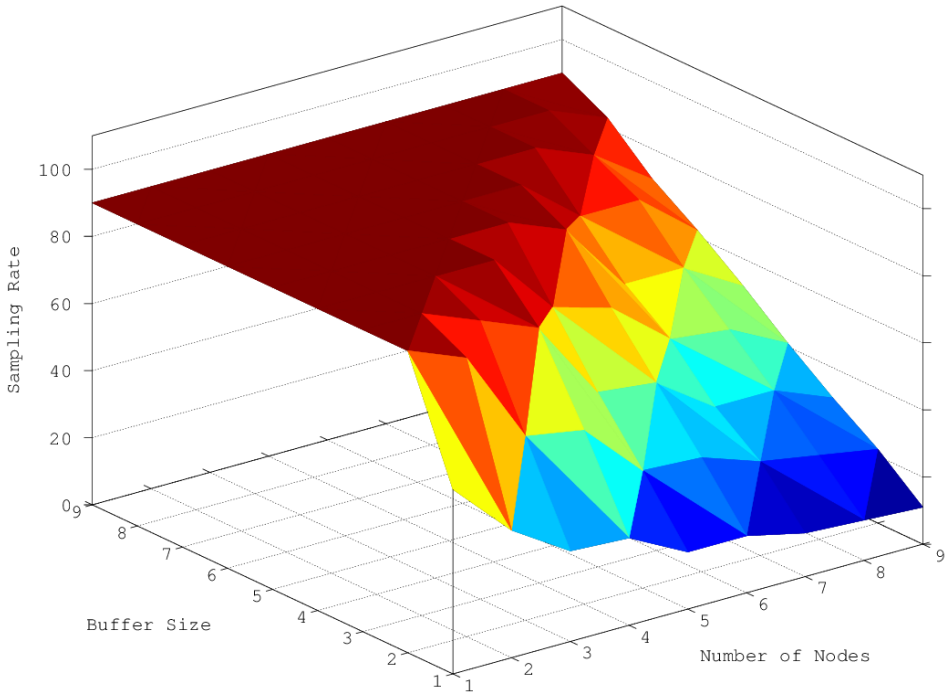


Fig. 6. The maximum sampling rate in case of using TDMA protocol and setting the value of `sensorTaskDelay` to 2ms

Configuration	State Space Generation				Model Checking Time	
	states ORG	states RED	Gain	Time	ORG	RED
25-5-3-10	1,741	402	77%	<1s	<1s	<1s
33-6-4-2	1,934	451	77%	<1s	<1s	<1s
25-5-4-10	3,718	945	75%	1s	<1s	<1s
30-6-4-2	9,353	2,774	71%	1s	<1s	<1s
25-6-4-2	34,503	10,368	70%	2s	<1s	<1s
20-6-4-2	57,621	17,714	69%	3s	<1s	<1s

Table 1. The size of the state spaces, the gained reductions, and the time consumption of TCTL model checking of a WSAN model with different configuration.

6.2 TCTL Model Checking of WSAN Applications

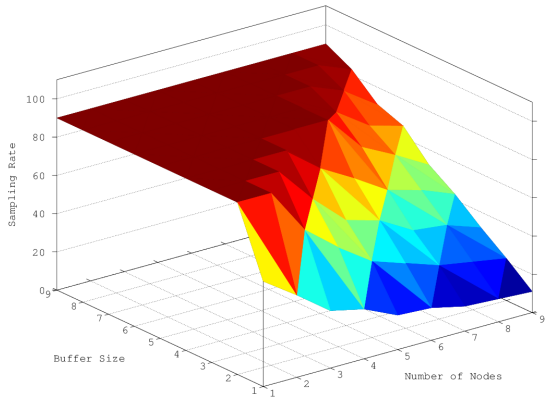
In addition to the schedulability analysis of Timed Rebeca models, they can be model checked against TCTL properties. In case of the WSAN model, checking for utilizing the communication medium in at least each 50 time units is the sample property we examined. This property can be formulated like:

$$\mathbf{AG}^{\leq 50}(\,\mathbf{A}(\mathit{freeChannel})\,\mathbf{U}^{\leq 50}(\neg \mathit{freeChannel})\,)$$

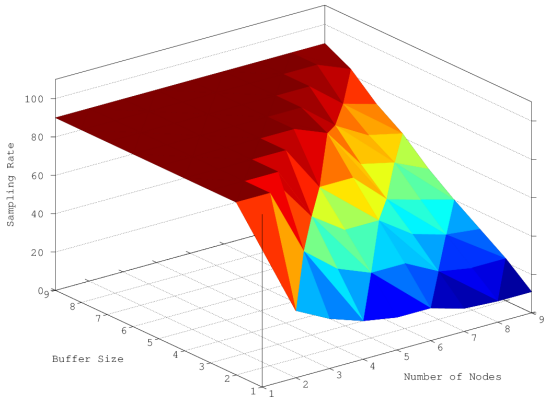
We verified the WSAN application in a limited number of configurations, varying the value of the sampling rate, the number of nodes, the packet size, and the sensor task delay.

We also applied "Folding Instantaneous Transitions" reduction technique, to make the model checking of WSAN applications against TCTL more efficient. The idea of folding instantaneous transitions is developed based on the fact that the instantaneous transitions take no time to execute; so, the system cannot "stay" in the states whose outgoing transitions are instantaneous. Hence, these states are not observable to the verifier (as an external observer). Note that as generally assumed in the modeling of timed systems, instantaneous transitions take priority over non-instantaneous ones. So, any state which has an instantaneous outgoing transition cannot have non-instantaneous transitions. Hence, there are two types of states: the ones whose outgoing transitions are all instantaneous (called *transient states*), and the ones which have no outgoing instantaneous transition (called *progress-of-time states*). Folding instantaneous transitions eliminates all instantaneous transitions as well as all transient states from state spaces. Therefore, there is a transition between two states of the resulted transition systems if and only if the two states are consecutive progress-of-time states in the original transition system.

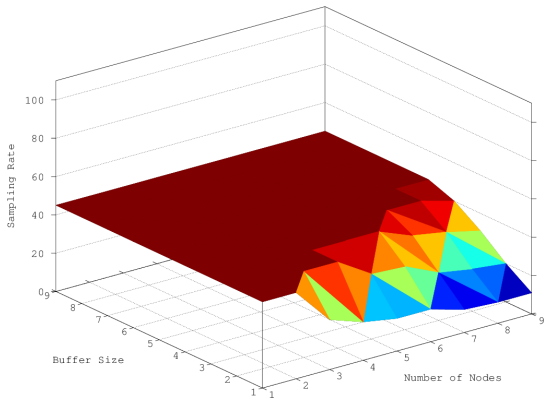
The results of these experiments are depicted in Table 1. In each row, the configuration (the numbers which are separated by a dash) is a combination of the sampling rate, the number of nodes, the packet size, and



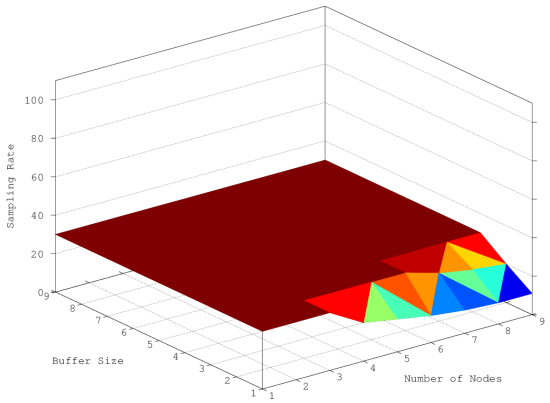
(a) TDMA, Sensor task delay is 5ms



(b) TDMA, Sensor task delay is 10ms

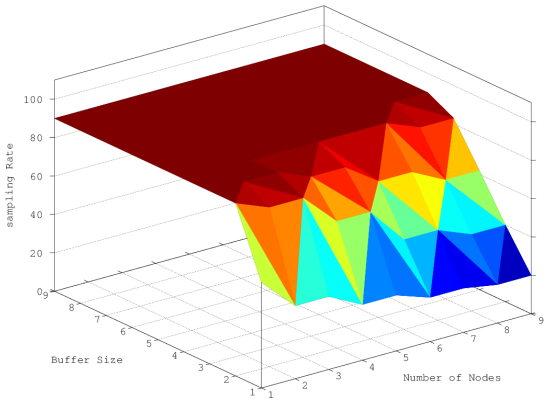


(c) TDMA, Sensor task delay is 20ms

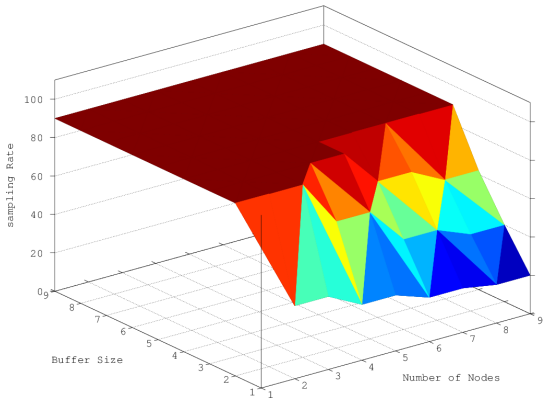


(d) TDMA, Sensor task delay is 30ms

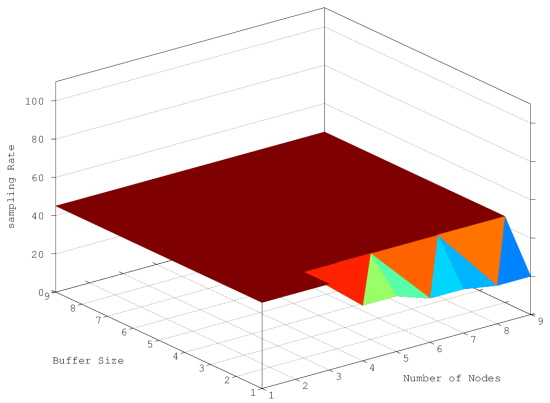
Fig. 7. Maximum possible sampling rate in case of different communication protocols, number of nodes, sensor internal task delays, and radio packet size



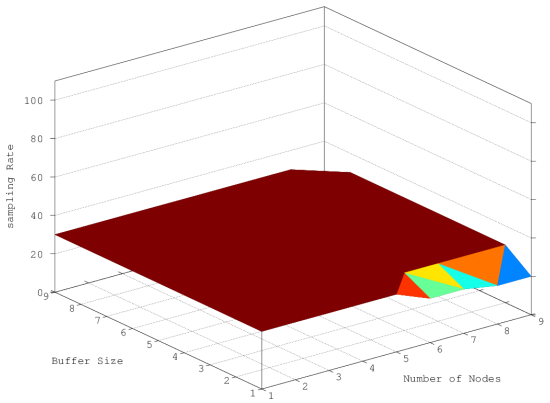
(a) B-MAC, Sensor task delay is 5ms



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



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



(d) 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

the sensor task delay of the experiment, respectively. We choose the state space size and the model checking time consumptions as the performance metrics of the model checking. The values of these metrics are depicted in a table for each case study. In the tables, ORG is used to refer to the original state spaces and RED is used to refer to the reduced state space (i.e., Folding Instantaneous Transitions). We also reported the spent times for the state space generation. There is no difference between the spent times for the generation of the original state spaces and the reduced state spaces as the reduction technique is applied during state space generation with a very small footprint, so only one number is reported as the spent times.

As shown in Table 1, the time consumption of the model checking is less than one for all cases and changing the configuration of the model does not end in large state spaces. However, the effectiveness of the reduction technique is reduced in configurations which result in bigger state spaces. This is because of the fact that changing the configuration of WSAN in this way does not increase the number of messages which are sent at the same time. So, the chance of finding transient transitions is decreased as there is no increment in the number of simultaneously executing instantaneous transitions.

7 Related Work

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

System Simulation. Simulation of WSAN applications is useful for their early design exploration. Simulation toolsets for WSANs provide modeling and analysis of networks [20], power consumption [31], and deployment environment [37]. 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 [6]. 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. [23]. 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 of models with guarantees [6]. Model checking tools systematically check whether a model satisfies a given property [5]. 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 [24]. 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. Jaghoori et al. in [15] and [16] presented a framework for schedulability analysis of real-time concurrent objects. The proposed 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 [25] 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.

8 Conclusion and Future Work

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.

In this paper, we only addressed the schedulability analysis of WSAN components and did not consider the interference on the wireless channel issues (the details of communication protocols). We assumed that there is a reliable wireless infrastructure in the application which provides guaranteed delivery of messages, which is a reasonable assumption for a wide range of deployments of structural health monitoring and control systems. However, this work can be extended by taking the details of communication protocols into account together with noises and unreliability of wireless communication which results in errors. This way, only Ether and RCD actors have to be modified to contain the details of the protocols. Note that the implementation of the chosen MAC protocol as well as the interaction of the processing hardware with the transmitter has to be added to RCD to take hardware and software into account and provide combined analysis of the underlying hardware infrastructure as well as the application software. Other different assumptions, including fairness in access to B-MAC, time drift of actors, and uncertainties, can be added. Note that extending the number of modeled MAC layer protocols also can be performed as a future work of this paper. Comparing the efficiency of MAC protocols in different cases to study their characteristics will be one of the outcomes of this extension.

On the other hand, some WSAN applications also exhibit probabilistic behaviors which are not discussed in this paper. Also, in many soft real-time systems it is desirable to know whether the application does not violate any deadlines with at least a given probability. This is particularly important when deadline violation probability is very small, but requires significant extra resource allocation to be avoided completely. In resource-constrained WSAN environments, the price of such safety guarantee may be too high. To address these cases, we are going to extend this work by using Probabilistic Timed Rebeca [14] for modeling WSAN application and benefiting from combining performance evaluation with functional verification of models. This way, we develop one model for the purposes of model checking, performance evaluation, and probabilistic model checking.

Acknowledgments

The work on this paper has been supported in part by the project "Self-Adaptive Actors: SEADA" (nr. 163205-051) 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. 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. Arnold H. Buss. Modeling with event graphs. In John M. Charnes, Douglas J. Morrice, Daniel T. Brunner, and James J. Swain, editors, *Proceedings of the 28th conference on Winter simulation, WSC 1996, Coronado, CA, USA, December 8-11, 1996*, pages 153–160. IEEE Computer Society, 1996.
5. Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking.* MIT Press, 1999.
6. 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.
7. 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.
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. Ali Jafari, Ehsan Khamespanah, Marjan Sirjani, Holger Hermanns, and Matteo Cimini. Ptrebeca: Modeling and analysis of distributed and asynchronous systems. *Sci. Comput. Program.*, 128:22–50, 2016.
15. Mohammad Mahdi Jaghoori, Frank S. de Boer, Tom Chothia, and Marjan Sirjani. Schedulability of asynchronous real-time concurrent objects. *J. Log. Algebr. Program.*, 78(5):402–416, 2009.
16. Mohammad Mahdi Jaghoori, Frank S. de Boer, Delphine Longuet, Tom Chothia, and Marjan Sirjani. Compositional schedulability analysis of real-time actor-based systems. *Acta Informatica*, pages 1–36, 2016.
17. Ehsan Khamespanah, Kirill Mechitov, Marjan Sirjani, and Gul A. Agha. Schedulability analysis of distributed real-time sensor network applications using actor-based model checking. In *Model Checking Software - 23rd International Symposium, SPIN 2016, Co-located with ETAPS 2016, Eindhoven, The Netherlands, April 7-8, 2016, Proceedings*, pages 165–181, 2016.
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. 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.
25. 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.
26. Joseph Polastre, Jason L. Hill, and David E. Culler. Versatile low power media access for wireless sensor networks. In Stankovic et al. [36], pages 95–107.
27. 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.
28. 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.
29. Zeinab Sharifi, Siamak Mohammadi, and Marjan. Sirjani. Comparison of NoC Routing Algorithms Using Formal Methods. To be published in proceedings of PDPTA'13, 2013.
30. 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.
31. 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. [36], pages 188–200.
32. Marjan Sirjani, Frank S. de Boer, and Ali Movaghar-Rahimabadi. Modular verification of a component-based actor language. *J. UCS*, 11(10):1695–1717, 2005.
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. 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.
36. 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.
37. 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.