A Verified Wireless Safety Critical Hard Real-Time Design

Hernán Baró Graf[1], Holger Hermanns[1], Juhi Kulshrestha[1], Jens Peter[1,2], Anjo Vahldiek[1], Aravind Vasudevan[1,3]
[1]*Saarland University – Computer Science, Saarbrücken, Germany*
[2]*Saarland University, Department of Mechatronics, Saarbrücken, Germany*
[3]*Max Planck Institute for Informatics, Saarbrücken, Germany*

***Abstract*—Wireless communication, hard real time requirements and safety criticality do not go together well. This paper reports on the modelling, design, simulation, implementation and deployment of a small exemplary case that possesses all these features. State-of-the-art verification and simulation means are employed to ensure its proper operation.**

***Keywords*-wireless control, safety criticality, dependability, modelling and validation**

I. Introduction

Classical abstractions used in software engineering leave out "nonfunctional" aspects, such as cost, efficiency and robustness. In particular in the field of embedded software, there is a growing awareness that these abstractions no longer suffice to arrive at dependable designs [8], [12]. Embedded software is subject to complex and permanent interactions with its, mostly physical, environment via sensors and actuators. Wireless sensor and actuator networks such as WirelessHART [4] are setting new standards in this domain. Regardless if wireless or wired, the possibilities of user intervention with such systems are usually very limited and high requirements are therefore put on performance and dependability as the embedded nature complicates tuning and maintenance.

At the same time, embedded software permeates safety and mission critical applications in a spectrum ranging from pace makers to chemical plant control. The future will likely bring an increase in wireless technology also in the context of safety-critical control applications. Wireless communication is known to be inherently unreliable and often is characterised by relatively high message loss rates. When hard real time requirements are to be met despite wireless communication, it becomes even more difficult to come up with safety guarantees. The central problem is that consecutive failures in message transfer may affect the correct functioning.

There are two specific application conditions under which this problem may be overcome: First, it is possible – if the application allows it – to run the control loop on a pace that is slow enough such that the probability of a prohibitive number of consecutive message losses is clearly negligible. Second, if the systems allows for a fail-safe state, it is possible to force the design into that state whenever the number of consecutive losses exceeds some justifiable bound. If not well configured, this may lead to a system that is barely operational, but it is a safe design after all.

But what to do if neither of the two conditions are met? If there is no fail-safe state and if the controller must react within a very limited time window? A responsible designer will likely react very reluctantly to the idea of solving with wireless technology a control problem that is hard real-time, safety-critical and does not offer a fail-safe state to fall back to.

In this paper we are looking at a very tiny control problem of precisely that sort. It is safety-critical, has hard real-time requirements and does not have an obvious fail-safe state. We are looking at the brakes of an ordinary bicycle and are investigating what happens when the mechanical connection is replaced by a wireless sensor and actuator pair. We report here about the modelling, verification, design and construction of such a *wireless bike brake*.

This project was originally conceived as the *mad bike project*, and there is some craziness in this idea after all. However we think that it is a very good case to study the principal possibilities and limitations of wireless control without going to excessive infrastructure costs. Indeed, our investigations allow us to discriminate between different options to solve this and similar problems with different dependability guarantees.

The dependability guarantees we can give are quantitative – or probabilistic – guarantees. We guarantee that the probability of the system to not react within a safe time window is low, where the precise number depends on the assumptions made about timing and individual message loss probabilities. We measure these individual loss probabilities on the real design and use these measurements as input to our model-based analysis. In fact, we consider both a setup with message loss probabilities in the order of 10^{-5} as well as a more challenging scenario with losses occurring for more than 50% of the messages sent.

For the purpose of studying quantitative dependability, we resort to a modelling formalism that supports the specification of real-time as well as probabilities. The principal ingredients of the application are modelled as probabilistic timed automata (PTA), specified in the language Modest [1]. These specifications are submitted to an analysis engine, *mcpta*, a PTA model checker developed for Modest [2], [10]. This model checker enables us to arrive at the above guarantees.

978-1-4577-0351-5/11/$26.00

It can also derive average response time bounds and other measures. It does so by using the PRISM model checker [6] as a back end, encoding time as a variable automatically. The use of probabilistic timed automata model checking is very natural for such a concrete scenario with message losses and hard real-time requirements. However, this is likely the first concrete use case of this kind, mainly because PTA model checkers debuted only very recently [2], [7], [13] and only the Modest checker seems to be stable and available for download [10], providing high level modelling features as well as convenient programming constructs (such as arrays and global and local variable scoping). The Modest tool set also includes a simulation engine *modes*, which we use for validation purposes, namely to link between verification results and empirical measurements.

II. A Wireless Bike Brake

This section discusses the principal aspects of a wireless bike brake, as an example of a safety critical wireless sensor network system that, despite making use of wireless communication, has hard real time requirements.

A. Problem Statement

We consider a bike brake, where the communication between the brake handle and the brake shoe is wireless. The wireless design has led us to use a number of components, the force sensor, sender, replicator(s), receiver, actuator and the alarm system. For a more detailed description of these components, see section II-B.

The bike brake system has very strict real time timing requirements. The time between the rider applying the brake by pressing the handle to the braking force actually being applied, has to be short enough to ensure the safety of the rider. The time for applying the brake includes the time for the force sensor to notice the difference in the force being applied, conveying it to the sender, the sender transmitting these values wirelessly to the receiver and its informing the actuator to apply the braking force. The timings of all these steps in the braking process are also limited by the hardware being used.

A regular bike-rider may ride at 30 km/h, i.e. 8 m/s. We have decided that the communication between handle and shoe cannot exceed 150 milliseconds(ms), based on the fact that the actuator mechanics takes 100 ms to react and that adding both intervals leads to 250 ms, which is equivalent to 2 meters (to react). In the bike experiment we configure an alarm system that continuously monitors the connection and alerts the rider whenever this bound is trespassed.

B. Principal Design

Replacing only the connecting wire between the brake handle and the brake shoe with a wireless connection does not result in a wireless electric brake. We identified the following basic components and their functionalities.

Force Sensor: This sensor is an apparatus which replaces the brake handle and produces a digital or analog signal representing the pulled force.

Sender: The sender is located near to the Force Sensor and has a wired connection to it. It reads the signal and sends it using a wireless connection to the Receiver.

Receiver: This component receives values from the Sender and modulates a control signal for the Actuator to a wire.

Actuator: In general this component produces the brake force based on the control signal of the Receiver.

Alarm System: If any problem occurs, the Alarm System has to notify the rider that the brake is not working in this moment.

Replicator(s): To increase reliability we study the option to introduce a (network of) node(s) for redundancy. A Replicator component acts as a Sender and a Receiver combined, it listens to the Sender (or all Replicators) and echoes the last value obtained to the Receiver (or to all Replicators). A scenario with 4 replicators is depicted below.

Section IV-D provides a deeper discussion of the actual components used.

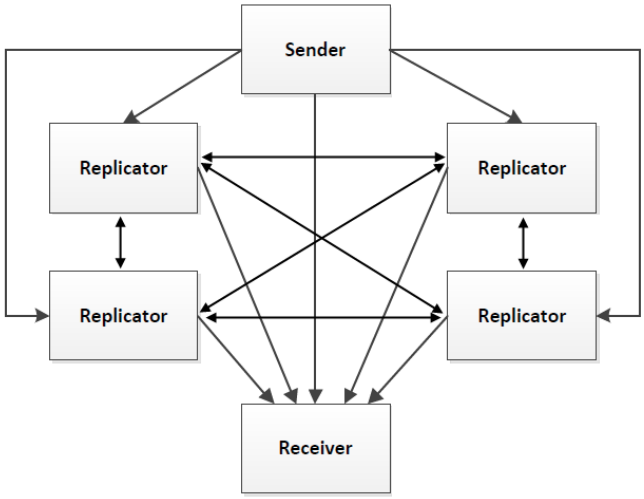

Figure 1
The Wireless Brake Model with 4 replicators

C. Connection Implementation

At the core of the problem is the design of the wireless connection between sender and receiver. As in many other hard real time applications, a TDMA-based communication is a good choice to assure time predictable message delivery. In fact, we are using the MyriaNed wireless nodes as basic components [9], a product manufactured by the Dutch company Chess. These nodes include a micro-controller (Atmel atxmega128A1) and a radio transceiver (Nordic Semiconductor nRF24L01+). The micro-controller features an integrated analog to digital converter which we use to

read a simple force sensor. The radio transceiver operates in the free 2.4 GHz ISM band, so it can be used without a license. The nodes communicate via the Chess gMAC protocol, a proprietary variation of TDMA which is easily configurable. The principles behind the gMAC protocol have already been subject to substantial verification efforts using timed automata [3], [11].

The basic setup consists of a total of two nodes, one acting as receiver and one as sender. After initialisation, they communicate via the gMAC protocol, which in this case is essentially a round-based TDMA protocol. A frame is a collection of TDMA slots and frame time is the time spent per frame, thus the sum of the times spent on each TDMA slot. Each communication frame lasts some n ms and each component sends one message to the other component per frame. Throughout the paper, the words frame and round are used interchangeably.

It is possible to either fix the assignment of components to slots in a static way, or to resort a dynamic slot allocation (DSA), which basically corresponds to running slotted ALOHA. From a reliability perspective, the fixed slot assignment (FSA) is clearly superior, since it eliminates contention on slots. We however decided – perfectly in the spirit of the mad bike project – to let the MyriaNed nodes run in their default DSA configuration. This made us face slot contention (and hence loss probabilities) in the order of 50 %, thereby serving as a natural emulation of a very noisy environment. The remainder of the discussion in this paper is in reference to this setup, except for Section V where we harden the design by switching to the FSA setting.

The minimum setup we consider has 2 slots (for sender and receiver) and we will later explain how redundancy can be achieved by adding further nodes (replicators). Section IV-B covers a discussion about a safe and minimal reaction time and our experimental studies.

III. Verification and Simulation

We use state-of-the-art model checking techniques based on probabilistic timed automata to understand the principles and limits of the above design, and to get insight into variations of the solution. For these purposes we built a model of the nodes in a formal language that supports concurrency and probabilities. In this section we introduce the formalism, sketch the model and then report on model checking and simulation results, for crucial properties of the design that relate hard real time properties and probabilities.

A. Model Formalism: Modest

Modest is a high-level modelling and description language for stochastic timed systems. It supports a compositional modelling approach, providing all common operators such as nondeterministic choice or parallel composition to combine small models into larger and more complex ones. It combines concepts known from modern programming languages (e.g. exceptions and exception handling) and verification-oriented modelling languages like Promela and LOTOS, and extends these with probabilistic branching, continuous probability distributions and time. One of Modest's design goals is orthogonality [1]: almost all concepts, such as probabilistic branching, time, or continuous distributions, do not depend on each other, so features can be removed to obtain a language describing a submodel of stochastic timed systems. In the context of this paper, we restrict to the probabilistic timed automata fragment of Modest [2].

Language features: We introduce the principal notations needed to understand the models. The basic constituents are processes, that may run in parallel, can possess and manipulate local variables, may share global variables, and may emit or synchronise on user-definable actions such as `send`, `receive`, `reset`, ... The basic constructs are:

- `::` is a process delimiter expressing options. In particular, it can be weighted with probabilities, for example: ": w_i :" for the $palt$ construct appearing later.
- `;` is a sequential composition operator for processes. " `P ; Q` " executes P until it terminates and then it continues with the execution of Q.
- $\{= asgn_1 \dots asgn_k =\}$ is a variable update (atomic assignment), usually occurring after some action.
- `alt` $\{ :: P_1 \cdots :: P_k \}$ is the usual alternative composition. In case several alternatives are *enabled* (e.g. from $P_1 \dots P_k$), one of these alternatives is chosen non-deterministically (e.g. P_i).
- `do` $\{ :: P_1 \cdots :: P_k \}$ repeatedly chooses an alternative P_i in the same nondeterministic manner as alt. It terminates whenever one of the running processes P_i executes a $break$ action.
- `when`(b)`[urgent(`c`)]` P is a guarded command, where b is a boolean expression over variables and clocks, called guard. The execution of P is blocked unless or until the boolean guard b becomes `true`, which may happen because of time advance or because of a change in a shared variable referred to in the guard. The optional `urgent(`c`)` modifier enforces the enabled actions of P to be happen without further delay whenever the boolean expression c holds. It imposes an extra urgency constraint to b. If b and c are identical then we can simply declare `when urgent(`b`)`.

Clocks are special variables that change their values linearly with real time, like in timed automata. They are available as a data type `clock`.

Properties in Modest: In order to analyse a Modest model, a set of properties to be evaluated has to be defined.

In Modest, this is done via a property declaration using Modest's Unified Property Language. This is a variation of well-known temporal logics such as PCTL, TCTL and CSRL, and also allows to refer to accumulation of rewards or bonuses.

B. The Wireless Brake Model

The model is based on the assumptions that synchronicity is attained by the gMAC protocol, which is already implemented in the nodes. In the model described here, the probability that an individual message does not arrive at the receiver side is a constant p for each individual message transmitted. The message size is fixed and one message is enough to send the information needed at any time.

For the first basic model we consider two logical components following the principal design in II-B: Sender and Receiver. The basic unit in Modest is a process, so the overall specification consists of two processes running in parallel, $sender$ and $receiver$. For the sake of better readability, the presented code is slightly different, and simplified, from the one actually used in the verification.

The $sender$ is parametrised with a slot number, SLOTSENDER, that is unique. The decision adopted for synchronising is based on slots so the $receiver$ also gets the $sender$'s slot number as a parameter. This enables the $receiver$ to listen to the $sender$.

The processes have two main actions, communicate and reset in every frame. The $sender$ sends the data through a wireless channel abstracted by a shared variable $data\_s$ indexed by the slot number (for example SLOTSENDER). The $receiver$ is listening and has a probability p of losing the message transmitted, here specified in percent.

```
sender(int SLOTSENDER) {
 clock timer = 0;
 bool brake = true; //local value to be send
 do{
  :: when urgent(timer>=SLOTSENDER)
      send {= data_s[SLOTSENDER]=brake =}
  :: when urgent(timer>=ENDFRAME)
      new_frame {= timer=0 =}
   }
}
```

At the end of every frame, ENDFRAME, the timers are reset and the $receiver$ updates the counter $lost$, that keeps the number of consecutive message losses according to what happened in the frame. In case this counter reaches a established maximum MAXLOST, a flag is raised. This happens immediately, assured by an $urgent$ action $crash$. In the other case, nothing happens (abstracting from the true operational behaviour) but resetting the clock. This is modelled via a tau action, which stands for some action invisible to the outside. To avoid unnecessary delays, we let the model progress as early as possible, and therefore frame the guarded decisions with urgency constrains, i.e. $when\ \ urgent$.

```
receiver(int SLOTSENDER) {
 clock timer = 0;
 bool msg = false;  //local stored value
 int lost = 0;   //number of messages lost
 do{
  :: when urgent(timer>SLOTSENDER)
      palt {
        :100-p: {= msg=data_s[SLOTSENDER] =}
        :    p: {= msg=false =}
       }
  :: when urgent(timer>=ENDFRAME)
      alt{
       :: when(msg==false) inc {= lost++ =}
       :: when(msg==true) reset {= lost=0 =}
       };
        alt{
          :: when(lost>=MAXLOST) crash {=timer=0=}
          :: when(lost<MAXLOST) tau {=timer=0=}
         }
   }
}
```

In this process we can see that there is a probabilistic choice, $palt$, representing the possibility of losing the message with probability p (in percent). The nondeterministic choice alt allows any of the following enabled actions to be fired. In the $receiver$ the disjoint guards on the actions ensure only one action to be enabled at a time.

C. Introducing the Replicators Wireless Network

Adding redundancy into a system is a classical concept to improve the overall reliability. To experiment with this idea of a replicator network discussed above, we introduce a generic $replicator$ node, that acts as a $receiver$ and a $sender$ at the same time. At a glance it is obvious that any type of redundancy will improve the results, but it should be considered that adding a node induces more communication, which in a TDMA setting is naturally accommodated by extending the number of slots per frame accordingly. This means that the more nodes we introduce as senders the longer the frames become. If we want comparable results and we know that we need a reaction within 150 ms we must keep a fixed number of slots. For practical reasons we established 12 slots in 150 ms and we divide the frames accordingly to the number of transmitting nodes.

To study the effect of replication, we add in parallel a set of nodes as $replicators$, extending the model with 1, 2 and 4 $replicators$. The change to the model is minimal. The behaviour of the $replicator$ modelled here is simple (and simplified for the sake of presentation). It basically listens to the $sender$ during the slot SLOTSENDER and with probability p it may not hear the message. Then at SLOTREPLICATOR it replicates the value last received and, again, with a certain probability p the $receiver$ may hear it.

We show a *replicator* module in a "1-replicator" setting. We also add an action to listen (with probability p) messages from the *replicator* without loosing the information already received from the *sender*, thus we use an OR operator. The other actions remain unchanged.

```
replicator(int SLOTSENDER, int SLOTREPLICATOR) {
 clock timer = 0;
 bool msg = false;  //local stored value
 do{
  :: when urgent(timer>SLOTSENDER)
       palt {
        :100-p: {= msg=data_s[SLOTSENDER] =}
        :    p: {= msg=false =}
         }
  :: when urgent(timer>=SLOTREPLICATOR)
      replicate {= data_s[SLOTREPLICATOR]=msg =}

  :: when urgent(timer>=ENDFRAME)
     reset {= timer=0 =}
  }
}

receiver(int SLOTSENDER, int SLOTREPLICATOR) {
...
 :: when urgent(timer>SLOTREPLICATOR)
    palt {
     :100-p:{= msg=(msg OR data_s[SLOTREPLICATOR])=}
     :    p:tau
    }
 :: when urgent(timer>SLOTSENDER)
...
}
```

For 2 and 4 *replicators* we did similar changes in the *receiver* but now we also need to pass all the slots to the *replicators* and *receiver* to enable them to listen to other *replicators*. Since we are interested in not loosing more than MAXLOST messages in a row, every receiving node keeps the difference between the frame where it last received message and the actual frame. When this difference in the *receiver* reaches MAXLOST an alarm is fired.

D. Model Checking

We build our model using the Modest checker tool, which allows us to model-check various properties of interest. Until further notice, we assume the TDMA frame to contain 2 slots and set its length to $n = 25$ ms (i.e. the slot's length is 12.5 ms). The parallel composition of the nodes results in a system running on a global time line.

Requirements: there are a couple of requirements that relate probabilities and time in these case studies. We focus on a crucial one now and another later for simulation and experiments. As discussed above, the safety of the rider requires that the *receiver* (at the brake shoe) must receive a command from the *sender* within a limited time interval of t ms. We assume that t is a multiple of the length n (in ms) of a communication frame. We can obviously fit $\frac{t}{n}$ frames into the interval of length t. The model is set up in such a way that a $crash$ action is triggered immediately if the count of consecutive message losses exceeds or equals MAXLOST. Thus, if MAXLOST$= \frac{t}{n}$, the risk of not receiving any message within t ms corresponds to eventually crashing in an interval of length t. This requirement is expressed by the following property:

```
property PMAX_crash = Pmax(<> crash && time <= t)
```

The temporal operator *eventually*, represented by $<> a$, usually denoted $\diamond a$, states that at some point during the execution the event a will have happened. This property asks for the maximum probability of such a behaviour ($Pmax$). Since PTA are in general non-deterministic, the maximum quantification isolates the highest probability over all resolutions of non-determinism.

We here report on a series of verification runs for the above requirement that are summarised in the table below. In this setup, we run the model checker for different time constraints t: 150 ms, 1 s and 10 s. We configure the model using 0, 1, 2 and 4 replicators, Rep, to compare reliability (thus in 150 ms we have 6, 4, 3 and 2 frames respectively). We also need to vary the corresponding maximum consecutive losses (in frames), i.e. MAXLOST $= 6, 4, 3$ *or* 2. From the experiments we adjust the probability of a single message lost p to 51%, which corresponds to the average obtained from the experiment logs, for the chosen setup (DSA). The results were produced within 1 to 5 minutes, depending on t, in a dual-core notebook with 3 GB of RAM.

Rep\t	150 ms	1 sec	10 sec
0	0.0175963	0.2796740	0.9732774
1	0.0221659	0.3091858	0.9844734
2	0.0304205	0.3709793	0.9935737
4	0.0639192	0.5259302	0.9996357

TABLE I
MAXIMUM PROBABILITIES OF CRASHING WITHIN t

From table I we can easily deduce that the probabilities of having a failure increase with the time interval (approaching 1 in the infinite). It was unexpected that when having 4 replicators, also replicating to each other, the crashing probability is higher than with 1 replicator (or no replicator). This is a result of the fact that, despite the redundancy provided by the replicators, the slot added for each replicator node introduces an *expensive* delay in the communication, thus the probability of crashing within t, for a fixed t, increases. For instance within $150ms$ the "basic" model is better than the "4-replicators" model by a factor of 3.6. Nevertheless, when increasing the time interval, and probabilities got closer to 1, this factor is significantly reduced.

The problem here is that with adding nodes the number of frames available in the time window t goes down and MAXLOST goes down accordingly. This is not compensated by the additional redundancy.

So, as we will confirm later with the simulation and experiments, short fast frames from sender to receiver behave

better than longer frames with replicating nodes in between. But, there is also another aspect out of the scope of this paper that takes into consideration the battery life of the nodes and also the costs of the extra nodes This means that in short frames there are no slots for idle time whereas in longer frames the sending nodes have more idle slots. Idle times are known to increase (often significantly) battery life and efficiency. On the other hand, having replicators implies having more nodes and the system becomes more expensive. Possibly, future bikes will anyway be equipped with a network of wireless nodes for other reason (for gear shifting, rider monitoring for instance) with better performance and the use of this network as replicators for the brakes could become beneficial.

E. Simulation

The Modest tool set also comprises a discrete event simulator, *modes*. Simulation allowed us to get numbers as average and interval analysis. We therefore employed simulation as a link between model checking and experiments. First we statistically derive the maximum probability of crashing, as we did via the model checker, for a number or scenarios, now via simulation. For the 0, 1 and 2 replicators model, we established a number of simulation runs of 10.000, to not take long and provide good confidence intervals.

$Rep \backslash t$	150 ms	1 sec	10 sec
0	0.01550	0.27220	0.97640
1	0.02230	0.30470	0.98180
2	0.03250	0.36540	0.99350
4	0.06339	0.52470	0.99976

TABLE II
SIMULATED MAXIMUM PROBABILITIES OF CRASHING WITHIN t

Table II exhibits a good approximation of the actual results obtained in model checking. We use the simulation technology for further experiments, also to compare with the empirical results (reported on later). For this, we simulate the system and count how many crashes can occur within a given time interval. The *modes* engine allows us to introduce a watch expression over a counter and check for the number of crashes at a certain time point. It then calculates the resulting average over all simulation runs. The property we desire is expressed as:

```
property AVG_crash = Xmax(num_crashes | time == t);
```

For the "4 replicators" model we run 100.000 simulations using *modes*, taking 6 hours to complete, with the following results where:

S.D.: stands for Standard Deviation
L.C.L.: stands for Lower Confidence Limit
U.C.L.: stands for Upper Confidence Limit

Comparing the model checking for the "4-replicators" model and its corresponding simulation, as can be seen in last row of table I and the first and second columns of table III, we can conclude that the model checking results were precise (and faster) and that is expected that the simulation is also providing valid results for the experiment.

Property	Result	S.D.	L.C.L.	U.C.L.
PMAX 150ms	0.06339	0.243664	0.061427	0.065388
PMAX 1s	0.52470	0.499392	0.520641	0.528757
PMAX 10s	0.99976	0.015490	0.999609	0.999871
AVG 1s	0.76572	0.896036	0.758438	0.773002
AVG 10s	8.42719	2.988175	8.402905	8.451475

TABLE III
RESULTS FROM THE "4 REPLICATORS" MODEL

The attentive reader may ask why there are no average values for 150 ms, it is simply because it is very close to 0. The cases where a crash happened within 150 ms were far less than the cases with 0 crashes. This is an expected detail if we notice that the probability of a crash in 150 ms is around 6% in the worst case and that the average crash in a second (1 sec $\simeq$ 6.6 times 150 ms) is less than 1.

It is also worth noting that the average for 10 seconds is more than 10 times the average of 1 second. Why is this the case? Well, the simulator runs up to a certain stop point (1 sec or 10 sec) and starts the counter from 0 in the next run, so for 10 seconds there is a possibility of counting crashes involving two intervals of 1 second, where for 1 second intervals this crash is not seen (analogously between 150 ms and 1 sec).

This last point is important to remember, since it will become one of the differences between the simulation results and the experiments result. The experiments run for a long time collecting data in the logs, while the simulation has several runs over the same time interval, so inter-interval crashes, meaning that in the case the $lost$ counter starts counting in one interval and finishes in the next interval in the experiments are missed in the simulation since it is reset in every interval.

IV. REAL CHECKING

Aside from the abstract modelling and verification tasks, the wireless bike project also involved building the concrete embedded control software running on the MyriaNed node, as well as constructing a prototype of the bike. To bring the concepts of the software design across, we discuss some of its aspects below, concentrating on the ideal case where communication is non-faulty.

A. Initialisation and Calibration

Initially the Sender normalises the signal of the Force Sensor to the range between 0 and 100. In each communication round the Sender reads the signal of the Force Sensor and normalises it. The Receiver reads the normalised value from the received message and adjusts the phase modulation for the Actuator accordingly. Instead of modulating the voltage, it does so in a digital way: each millisecond an interrupt calls a function which sets the signal to ON or OFF, and the

percentage of ON signals over a period of 100 ms is always kept according to the most recent normalised value.

An alarm system is put in place that is triggered by the Receiver (in the form of a red LED on the bike handle). In each round, the Receiver sends a heartbeat including the number of continuous frames where it has not receive any message, i.e. the number of consecutive frames without any communication from Sender. Based on these messages or their losses the status of the alarm LED is updated.

B. Experimental Validation

Multiple tests were conducted on the real nodes to determine the safe minimum frame time. These tests were made with no other MyriaNed nodes running in the vicinity, but there could have been other sources of electronic disturbances (e.g., mobile devices, microwave, copy machine). The tests were not made in an isolated environment on purpose to emulate real life situations.

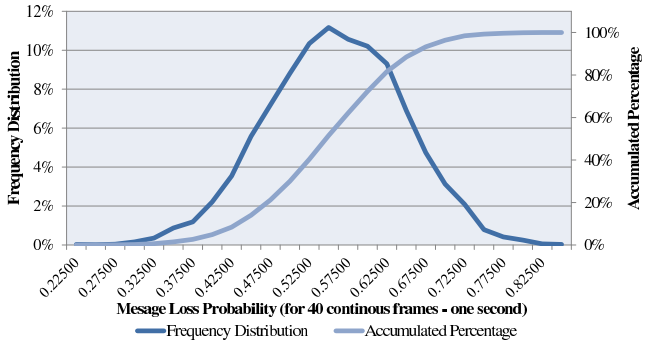

FIGURE 2
MESSAGE LOSS RATE FOR 25 MS FRAME TIME (108 MINUTES, 270,000 MESSAGES)

Minimum frame time: The run configuration for the base case of the experiments is as follows. Two MyriaCore nodes are used with DSA and 2 TDM slots per frame, and a payload of 32 bytes. To reduce interference at low frame times due to interrupts, I/O (to record the messages received and sent) is performed only once every ten frames, and the size of the I/O is only 10 bits. The approximate distance between the nodes is 1 meter. Since the objective is to check the more appropriate frame time, tests are conducted on different frame times.

Owing to the gMac protocol implementation and constraints posed by the hardware, we consider 12.5 ms for a slot to be safe. Because of this, a 25 ms frame time was used for a two node setup. As discussed, the DSA scheme gives rise to an average loss probability of 51%. Figure 2 shows that the message losses stay reasonably uniform within the guard boundaries of 0.4 and 0.7 (93 % of all message losses rates calculated), and this appears suitable as there are only rare outliers and inconsistencies (2.5% below, 3.5% above). In the worst case (82.5 % loss rate), which occurred only twice, still 7 messages are received within one second. This study seems to reflect the randomness in the DSA slotted ALOHA well.

Replicator Network: To verify the modelled and simulated results, we implemented a protocol in which the communication between a single sender and receiver is enhanced by a network of replicators. Table IV depicts the results. For this experiment the time to crash is fixed to 150 ms as reasoned before. The entire network is sending 12 messages within this time (slots). Each setup was tried in a different arrangement. Each experiment runs for 9:30 min of which only the last 9 min where used, due to start up abnormalities. The observed message loss rates of the individual nodes are similar to the previous discussed ones.

In this setup less replicators reduce the number of crashes due to the higher update rate of the sender. After detecting the crash, we observe the additional number of continuously lost frames as *length of crash*. We use these values to calculate the average length (of a crash). In this sense, the replicator network pays off a little bit more than no replicator. Increasing the number of replicators has the tendency to increase the average length of crashes due to the longer frames up to a point where high redundancy (several replicators) helps getting a good new message to the receiver, i.e. faster recovery. When we look at the message loss rate from the sender to the receiver, adding more replicators decreases the message loss rate by up to 25% in our experiments, although some messages may be too old. If we compare the distinctly received messages at the receiver from the sender the best results are achieved without any replicator, and this is consistent with the model checking section.

C. Experiments vs Simulation

At a first glance we see some discrepancies between the numbers obtained by the experiments and the simulations. Nevertheless we may have some pointers to address this. In the first place, as we explained in the simulation section III-E, there is a difference between counting crashes per interval (as done in simulation) and counting crashes for the whole experiment and then averaging those numbers. To shed some light on this, we look at the “4 replicators” model (and experiments 8 and 9) and depicted a small case with counter resets at the beginning of each interval. We checked the property “Average crashes within 10 second” and then compare it with simulation results in this table V.

Type	Avg. # of crashes in 10 sec
Simulations	8.42719
Experiment 8	12.6296296
Experiment 9	17.5925926

TABLE V
SIMULATION VS. EXPERIMENTS

Values are not similar (although closer), but that is ex-

Experiment #	# of Replicators	Frame time (in ms)	# of crashes	Avg. # of crashes in 1 s	Avg. length of crash(ms)
1	0	25	169	0.31	22.93
2	0	25	183	0.34	22.40
3	1	37.5	453	0.84	44.62
4	1	37.5	582	1.08	52.19
5	1	37.5	365	0.68	33.39
6	2	50	778	1.44	49.81
7	2	50	407	0.75	30.22
8	4	75	805	1.49	27.02
9	4	75	1104	2.04	19.70

TABLE IV
CRASH EXPERIMENTS

pected. It is important, though, that the order of magnitude is the same. And considering the standard deviation (2.988175) in the simulation, the 95% of the samples occurred in the interval $8.42719 + / - 3 * 2.988175$, this shows that $Experiments$ 8 and 9 are within the interval.

$Experiment$ 9 is a little bit further away, but this leads us to our second point. Changes in the relative position of the nodes between each other changed the results considerably. So even with the same code and same nodes, difference in the position of the nodes lead to considerable changes (60%).

D. Prototype Deployment

We here report on the features of a first wireless bike brake prototype built by our team. It has several innovative experimental features. As described above in the basic model (and more effective), the MyriaNed nodes establish a wireless communication between Sender and Receiver, and this is used to transfer heartbeat signals from Receiver to Sender: When the number of consecutive losses exceeds the allowed threshold, a red LED is activated at the alarm subsystem. Furthermore, instead of placing a force sensor inside the brake handle, we use an ordinary bike bell as the interface to the rider: For this purpose, we place a sensor (potentiometer) measuring the rotation of the bike bell. The Sender reads the measured value, normalises it and sends it to the Receiver which modulates the control signal for the Actuator. The Actuator is an electric drive engine (operating in 4,5-15V, 540 ER) mounted close to the brake shoe. It pulls and releases the brake shoe via a mechanical connection. Additionally, an external stationary battery supports the engine. We developed the mounting construction and an electronic board.

In practise the best case reaction time of this wireless electric brake is about 125 ms. The engine, sampling and A/D converters require a maximum of 100 ms altogether to adjust to the current control signal and the wireless communication consumes the remaining 25 ms. With a message loss probability of about 51% (which is an average according to our experiments), we can deduce from the first table that the reaction time of the brake will exceed the pre-specified upper bound of 250 ms (150 ms for the wireless connection) with a probability of almost 2%. As seen in table I where the probability of crashing in 10 seconds is 97.33%. Notably, these results are based on the DSA scheme, we consider the FSA scheme in the next Section.

V. THE HARDENED DESIGN

In the case of the FSA scheme, which does avoid any randomness in slot assignment, long-term measurements show a very stable behaviour with an individual message loss probability estimate in the order of about 10^{-5}, more precisely $p = 0.003\%$. This estimate is easy to obtain from a set of long lasting observations of a MyriaNed connection properly set up. However, the tininess of this estimate makes it impractical to study the entire braking behaviour (in reality or in simulation) in a reasonably small amount of experiments or simulation runs, because a message loss is now a very rare event. One might think of counteracting this with rare event simulation technology [5]. Instead we exploit the convenience of the model checking engine that we have in place, and readily arrive at the results displayed in table VI.

$Rep\backslash t$	150 ms	1 sec	10 sec
0	7.29000 E-28	2.55143 E-26	2.87946 E-25
1	8.74780 E-32	2.01199 E-30	2.30067 E-29
2	1.09349 E-31	1.96829 E-30	2.16512 E-29
4	3.64498 E-27	4.37397 E-26	4.81137 E-25

TABLE VI
CALCULATED MAXIMUM PROBABILITIES OF CRASHING WITHIN t IF USING FIXED SLOT ASSIGNMENT

As we can see, the overall design becomes highly reliable if based on the FSA setting. Indeed the model checker reveals that the probability of crashing within 10 seconds is below 10^{-24}. The idea of a replicator network is shown to be clearly counterproductive. In a nutshell, using FSA the mad bike brake project indeed delivers a verified wireless safety critical hard real-time system.

Aside from the wireless communication aspect, the current prototype is however still a very rough design. The mechanics is clumsy, and the brake force is applied too rigorously. This is also due to inaccurate sensing in the bell and the absence of force feedback. The applied force of the electric engine is inadequate. So, we are currently working on a second prototype, that will incorporate several

improvements to enhance the reaction time guarantees and to amplify rider convenience:

- a hydraulic disc brake with a more direct apparatus to apply and adjust the brake force, combined with an anti-lock braking system (ABS).
- a force sensor in the brake handle with a force feedback system.

VI. Conclusion

This paper has discussed the design, modelling, verification, simulation, construction, and deployment of a prototypical bike with wireless brakes. For the safety of the rider, we determine that is imperative that the brake shoe reacts within no more than 250 ms to a command issued by the brake handle. For the current prototype, the delay by the mechanical and conversion components is about 100 ms, which leaves some 150 ms for a successful communication between the wireless partners. Measurements show that an unfortunate configuration in DSA mode can lead to message loss probabilities around 50%. According to our model checking results, this implies that a bare communication delay of 150 ms cannot be guaranteed in 1 out of 50 brake attempts. Using a replicator network to add redundancy to the communication is revealed to be counterproductive by the model checker, and by experiments, if one takes the increased round timing into account. This insight is non-obvious, and is obtained by state-of-the-art PTA model checking and simulation and later confirmed with several experiments. We suppose it is of general interest to designers of wireless dependable systems to be able to study whether simple replication mechanisms can improve the safety guarantees.

Finally, the key to arrive at a safe design is to drastically reduce the individual message loss probabilities. For the MyriaNed system, this is achieved – maybe not surprising – by avoiding randomness is slot assignment, using the fixed slot allocation scheme FSA. The model checker enables us to readily prove that this twist results in a design with very high reliability guarantees, far beyond the "five-nines" yardstick 99.999%. In a nutshell, our model checking studies clearly hint at the potential tradeoffs when designing such systems.

Acknowledgements

We are grateful to Bert Bost, Frits van der Wateren, and Marcel Verhoef (Chess) for inspiring discussions and continuous support with the MyriaNed configuration. Arnd Hartmanns and Jonathan Bogdoll (Saarland University) have clarified subtleties of the Modest languages, and have supported the development of the models via their tool set.

This research is supported by the Seventh Research Framework Programme of the European Commission as part of the "Quasimodo" project, grant agreement number 214755, and by the German Research Council (DFG) as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS.

References

[1] H. Bohnenkamp, P. D'Argenio, H. Hermanns, JP. Katoen. MODEST: A Compositional Modeling Formalism for Hard and Softly Timed Systems. In *IEEE Transactions on Software Engineering*, volume 32(10), pages:812-830, 2006.

[2] A. Hartmanns, H. Hermanns. A Modest Approach to Checking Probabilistic Timed Automata. In *Proceedings of the 6th International Conference on the Quantitative Evaluation of Systems, QEST 2009*, pages: 187-196, IEEE Computer Society Press, September 2009.

[3] F. Heidarian, J. Schmaltz and F.W. Vaandrager. Analysis of a Clock Synchronization Protocol for Wireless Sensor Networks. In A. Cavalcanti and D. Dams, editors, *Proceedings FM 2009: Formal Methods*, volume 5850 of Lecture Notes in Computer Science, pages: 516-531, Springer-Verlag, November 2009.

[4] Highway Addressable Remote Transducer (HART) Protocol. http://www.hartcomm.org/ – accessed on Apr 28, 2011.

[5] S. Juneja, P. Shahabuddin. Rare-event simulation techniques: An introduction and recent advances. In *Handbook of Simulation,* volume 13 of Handbooks in Operations Research and Management Science, pages: 291-350, 2006.

[6] M. Kwiatkowska, G. Norman and D. Parker. PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. In *ACM SIGMETRICS Performance Evaluation Review*, pages: 40-45, March 2009

[7] M. Kwiatkowska, G. Norman and D. Parker. Stochastic Games for Verification of Probabilistic Timed Automata. In *Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of Lecture Notes in Computer Science, pages: 212-227, Springer-Verlag, September 2009.

[8] E.A. Lee Embedded Software. In M. Zelkowitz, editor, *Advances in Computers*, vol. 56, Academic Press, 2002.

[9] MyriaNed®: large wireless sensor and control network. http://wsn.chess.nl/ – accessed on Apr 28, 2011.

[10] Modest Toolset. http://www.modestchecker.net/ – accessed on Apr 28, 2011.

[11] M. Schuts, F. Zhu, F. Heidarian and F.W. Vaandrager. Modelling Clock Synchronization in the Chess gMAC WSN Protocol. In *Proceedings of the Workshop on Quantitative Formal Methods: Theory and Applications, QFM 2009.* Electronic Proceedings in Theoretical Computer Science 13, pages: 41-54, November 2009.

[12] Special issue on embedded systems. *IEEE Computer Science*, volume 33, 2000.

[13] UPPAAL PRO - Uppaal for Probabilistic Timed Automata. http://www.cs.aau.dk/~arild/uppaal-probabilistic/ – accessed on Apr 28, 2011.