Implementation of a Dominance Protocol for Wireless Medium Access

Nuno Pereira, Björn Andersson and Eduardo Tovar
IPP Hurray Research Group
Polytechnic Institute of Porto, Portugal
{npereira,bandersson,emt}@dei.isep.ipp.pt

Abstract

Consider the problem of scheduling sporadic message transmission requests with deadlines. For wired channels, this has been achieved successfully using the CAN bus. For wireless channels, researchers have recently proposed a similar solution; a collision-free medium access control (MAC) protocol that implements static-priority scheduling. Unfortunately no implementation has been reported, yet. We implement and evaluate it to find that the implementation indeed is collision-free and prioritized. This allows us to develop schedulability analysis for the implementation. We measure the response times of messages in our implementation and find that our new response-time analysis indeed offers an upper bound on the response times. This enables a new class of wireless real-time systems with timeliness guarantees for sporadic messages and it opens-up a new research area: schedulability analysis for wireless networks.

1. Introduction

The sporadic model [1] has proven to be very useful in the design of real-time systems. In this model, the exact time of a transmission request is unknown, but a lower bound on the time between two consecutive transmission requests from the same message stream is known. This model is supported in processor scheduling [2] (where a message stream is called a task) and in wired communication channels [3]. Wireless communication is of increasing interest in the design of distributed real-time systems, and many scheduling algorithms and analysis techniques for wireless communications are available for periodic messages. But for sporadic messages such results are less well developed. Most of the current wireless protocols cannot be analyzed to offer pre-run-time guarantees that sporadic messages meet deadlines, and the protocols that can offer such guarantees rely on polling, which is inefficient when the deadline is short and the minimum time between two consecutive requests is large.

In wired networks, sporadic messages can be scheduled efficiently using the CAN bus [4]. It has a medium-access control (MAC) protocol which is collision-free and prioritized, and hence it is possible to schedule the bus such that if message characteristics (periods, transmission times, jitter, etc.) are known, then it is possible [3] to compute upper bounds on message delays. This MAC protocol belongs to a family called *dominance protocols* or *binary countdown protocols* [5], which works as follows. Messages are assigned unique priorities and when messages contend for the channel, they perform a tournament such that the highest-priority message is granted access. This tournament is performed bit-by-bit of the priority, starting with the most significant bit. If a node (let us call it node *i*) contends with a recessive bit but it detects that another node transmitted a dominant bit then node *i* loses and it does no longer participate in the tournament. Finally, there is only one node that wins and it transmits its message.

The dominance protocol that was implemented in CAN uses open-collector/open drain circuits. Clearly, this does not easily extend to wireless channels. For this reason, researchers in the field recently designed dominance protocols for wireless channels; they are based on modulating priority bits using on-off keying [6-9]. These protocols can support sporadic messages efficiently and to the best of our knowledge, no other published protocol can do this (see [9] for a survey of relevant literature).

Unfortunately, the recently proposed dominance protocols [6-9] for wireless channels all have in common that they were not implemented and tested. Due to non-idealities in transceivers and the nature of the wireless medium, it is not obvious how these protocols should be implemented. There exist priority levels for which the protocols need to switch between transmit and receive modes for every priority bit, and this is potentially wasteful because many transceivers are not designed for frequent switching and hence every switching takes non-negligible time. It is well known that wireless channels typically have significantly higher noise levels than wired channels and that detection of pulses of short duration is difficult [10]. For this reason, wireless communication systems often use long codes [11] and/or spread spectrum modulation to increase the probability of a correctly received message. Unfortunately, these techniques cannot be used to transmit priority bits in the protocols [6-9]: (i) long codes operate on message-level and this is too coarse; (ii) there is the need to demodulate and decode an individual bit so that a decision can be made whether the next priority bit should be transmitted. Spread spectrum modulation cannot be used on

priority bits because it requires nodes that attempt to detect the priority bits be accurately synchronized with the senders: there are many senders and they can all send a priority bit at approximately the same time so a node trying to receive the priority bit cannot be synchronized with all of the senders. This makes it non-obvious whether wireless dominance protocols could work.

In this paper, we implement one of the proposed dominance protocols [9] and we call the implementation *WiDOM*. We evaluate it and find that in our experimental environment, the probability that a message is transmitted collision-free, correctly prioritized and correctly received by all other nodes (that is neither lost nor corrupted) is at least 99.99%. We believe this reliability justifies the development of schedulability analysis techniques for sporadic messages in wireless networks. We do so; we adapt the response-time formulations from the CAN bus to WiDOM and test the validity of the analysis. We find that more than 99.99% of all messages that were proven to meet deadlines also did so in practice.

The remainder of this paper is structured as follows. Section 2 provides the necessary background on the target platform, introducing the main aspects relevant for the implementation. Section 3 presents our implementation of the protocol in TinyOS, outlining its main software components. Next, in Section 4, the response-time formulation for our protocol is presented. The evaluation of the implemented protocol, in Section 5, entails the description of several experiments performed in order to test the properties of our protocol. Section 6 discusses the performance and the use of the protocol and compares it with previous work. Finally, Section 7 provides conclusions and future work.

2. The Platform

We implemented the dominance protocol on an embedded computer platform known as MicaZ [12]. It is a sensor network platform, offering a low power microcontroller, 128 kbytes of program flash memory and an IEEE 802.15.4 compliant radio transceiver CC2420 [13], capable of 250 kbits/s data rate. The MicaZ platform is supported by TinyOS [14], an open-source operating system designed for wireless sensor networks. This platform was found to be an attractive alternative for the implementation of our experiments because of the following relevant characteristics: (i) it allowed us to replace the existing MAC protocol in TinyOS easily; (ii) the timers available were sufficiently precise for our application; (iii) the radio can be put into a specific test mode, where it is possible to transmit an unmodulated carrier for an arbitrary duration; (iv) the radio has built-in RSSI (Receive Signal Strength Indicator)/energy detection functionality and Clear Channel Assessment (CCA) is available through a digital output pin; (v) the spread spectrum modulation used makes data frames resistant to noise and distortion. Due to (v), the main term that affects message transmission reliability is collisions. Our protocol is (as we will see) collision-free.

The CC2420 provides a packet level interface for sending and receiving, meaning that packets are sent by writing to the chip's memory over a bus and issuing a send command. Reception is signalled when the chip triggers an interrupt, and at that time the communication stack should read the bytes out of the chip's memory.

Dominance protocols in wired media require that a node can simultaneously transmit while it detects the transmissions from other nodes. Unfortunately, this is not possible in most radio transceivers, including the CC2420 because the transmitted energy is much higher than the received energy. For this reason, the CC2420 can only be either in transmission mode or in reception mode and it can take up to 192 μs to switch between these two modes.

WiDOM needs to transmit a carrier wave and the CC2420 radio can do this, either a modulated carrier or an unmodulated carrier in a transmitter test mode. The RSSI obtained when a node sends an unmodulated carrier is 9dBm stronger than the RSSI obtained when a node sends a modulated carrier (when the node transmits a data packet) [13]. Hence, for WiDOM, we use the unmodulated carrier in the tournament; the modulated carrier is transmitted only as a result of transmission of data.

WiDOM also needs to detect whether other nodes transmit a carrier wave. For this, it uses the CC2420 support for CCA. The CCA functionality of the CC2420 radio computes the average RSSI over the last 128 μs. To make a decision, this average is compared to a configurable threshold and then CC2420 sets the CCA digital output pin accordingly. This pin is sampled by our software communication stack to detect if other nodes are sending carrier pulses. After the radio is in receive mode, it takes 128 μs to make the first valid CCA operation.

Our protocol is heavily dependent on timers. The MicaZ's ATmega128 microcontroller provides two 8-bit timer/counter and two 16-bit timers. For our implementation, we use the 8-bit Timer/Counter2 for timing, since this is the timer used in CC2420 TinyOS communication stack, which we are partially replacing.

We built a component that abstracts the timing by providing an interface for setting timeouts and to deliver the clock ticks to a carrier-detection component that uses them to drive the CCA pin polling. This component configures the timer to give interrupts every 34.722 μs. For this reason, when we choose timeouts, these will be selected as multiples of 34.722 μs.

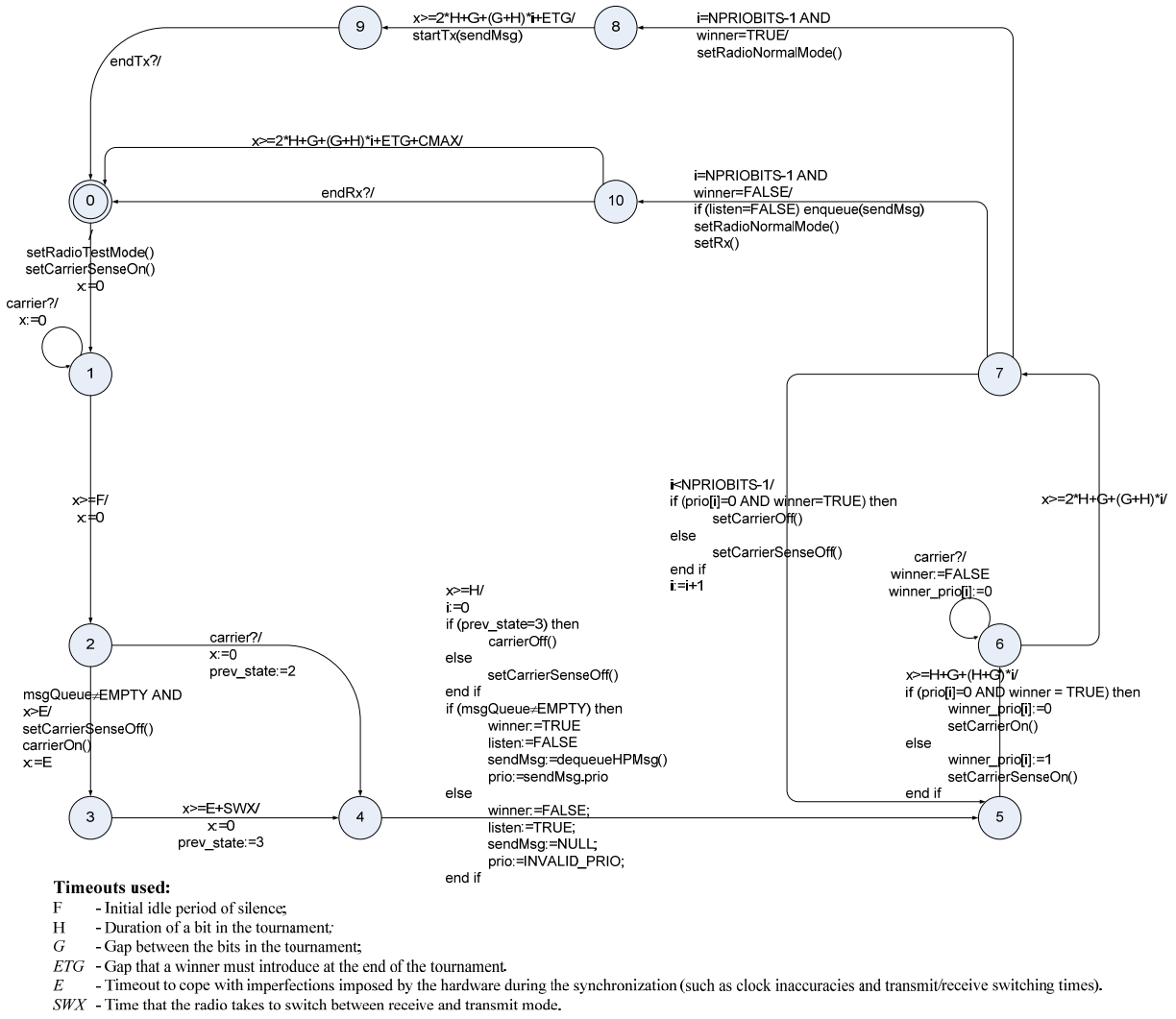

Figure 1. Automaton describing the implementation of the protocol.

3. The Protocol

To simplify presentation of the implementation, we present it using a timed-automata like notation. States are represented as vertices and transitions are represented as edges. An edge is described by its guard (a condition which has to be true in order for the protocol to make the transition) and an update (an action that occurs when the transition is made). In Figure 1, we let "/" separate the guards and the updates; the guards are before "/" and the update is after. We let "=" denote test for equality and let ":=" denote assignment to a variable. For those transitions with an update having many lines of code, it is assumed that the lines are executed sequentially.

Our dominance protocol assumes that all nodes can hear each other. The main idea of the protocol is that a message is assigned a static priority and when a message contends for the channel, nodes perform a tournament such that the highest-priority message is granted access to the channel. This tournament is performed bit-by-bit, starting with the most significant bit (States 5, 6 and 7 in Figure 1). A bit is assigned a time interval. If a node contends with a dominant bit then a carrier wave is transmitted in this time interval; if the node contends with a recessive bit, it transmits nothing but listens. This makes it possible for a node with a recessive bit to detect that another node has transmitted a dominant bit, and hence the node with the recessive bit withdraws. After the tournament, a node is a winner if it requested to transmit a message, and when it listened, it never heard a dominant bit. The winning node will then transmit (States 8 and 9 in Figure 1). A node which loses will receive, but if it does not receive a message it will timeout (transition 10→1 in Figure 1). In order for this scheme to work, nodes must agree on which time interval to use. This requires a convention, something that is easy to state and which we do. Before the beginning of a tournament, we must also establish a common reference point in time (this is ensured by States 0-5 in Figure 1).

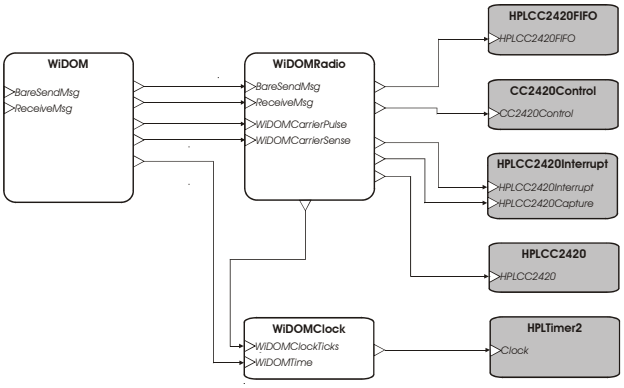

Figure 2. TinyOS component assembly.

Rectangles are implementation modules of components. Components names are in bold, the corresponding module has the same name with an 'M' appended. For simplicity, only the most relevant modules and interfaces are depicted, and configurations wiring components are omitted. Shaded rectangles are TinyOS components reused in the implementation. Triangles pointing into a rectangle are provided interfaces. Triangles pointing out represent used interfaces. The names of the provided interfaces are in italics.

The protocol automaton (Figure 1), refers to several timeout values. These timeout parameters were selected according to constraints given in [9], using NPRIOBITS=10: *E*=312 µs, *F*=21770 µs, *G*=555 µs, *ETG*=520 µs, *H*=1145 µs, *L*=5 µs, where *L* is the maximum computational time for a transition in the automaton. The timeout parameters are based on the assumption that a carrier pulse must have a duration (timeout named *TFCS*) of 486 µs so that the other nodes may detect it, and *SWX*=192 µs. The value of *TFCS* was experimentally obtained; further details are given in Section 6 and the relationship between these parameters can be found in [9].

The protocol has been implemented in TinyOS using *nesC* [15]. The main TinyOS software components of the implementation are presented in Figure 2, which provides a simplified overview of the implementation component assembly [14, 15]. Components `CC2420Control`, `HPLCC2420FIFO`, `HPLCC2420Interrupt` and `HPLCC2420` are part of the Hardware Presentation Layer (HPL) for the CC2420, and are regular TinyOS components reused by the implementation. These components provide basic functionalities for handling the radio. They are used by `WiDOMRadio` to do operations as starting/stopping the radio, configuring the radio parameters, and performing all the operations needed for sending/receiving packets.

The `HPLTimer2` is also an HPL component from TinyOS, which directly abstracts the 8-bit Timer/Counter2 on the MicaZ's ATmega128 microcontroller. This component is used by `WiDOMClock` to drive the timing of the protocol. `WiDOMClock` configures

```
interface WiDOMTime {
   command uint32_t get();
   command result_t reset();
   command result_t setAlarm(uint32_t when, uint8_t type);
   command bool isAlarmSet();
   command result_t cancelAlarm();
   async event void alarm(uint8_t type);
}
interface WiDOMClockTicks {
    command result_t postTicks(bool posting);
    async event void tick();
}
interface WiDOMCarrierSense {
   command result_t start();
   command result_t stop();
   event result_t channelBusy();
}
interface WiDOMCarrierPulse {
   command result_t carrierTXModeStart();
   command result_t carrierTXModeEnd();
   command result_t on();
   command result_t off();
}
```

Figure 3. WiDOM implementation-specific interfaces.

the timer prescaler to deliver the timer interrupts every 34.722 µs, which is used to drive the timing maintained by this component. The `WiDOMClock` has two functionalities. It presents an interface (interface `WiDOMTime`, in Figure 3) to the module that runs the WiDOM protocol (`WiDOM` component) which enables the MAC protocol to maintain its timing in a manner equivalent to the way 'x' is used in the protocol automaton from Figure 1. As seen in Figure 3, this interface provides commands to get the current time (the same as reading the value of 'x' in the protocol automaton) and resetting the time ('x:=0', in the protocol automaton). Additionally, this interface allows the setting of timeouts (called *alarms*) when the time reaches a certain value ('x>=' conditions in the automaton). Notice that when an alarm is set, an alarm type is specified. This is delivered when the alarm fires.

The second functionality offered by `WiDOMClock` is providing the clock ticks it receives from the hardware timer to other components. More specifically, because carrier sensing is made by polling the CCA pin of the CC2420, the `WiDOMRadio` issues a command to `WiDOMClock` component to get the clock ticks every time it has to perform carrier sensing. These clock ticks are then used to drive the polling of the CCA pin.

The `WiDOMRadio` component is responsible for providing all radio functionalities needed by the MAC protocol. It handles the interactions with the HPLs for sending/receiving packets, and provides to the `WiDOM` component a simple send/receive interface (interfaces `BareSendMsg` and `ReceiveMsg` are commonly used TinyOS interfaces for these purposes). A packet indicated for sending to this component will be sent without any kind of arbitration. Packets received are indicated to the protocol only after they have been entirely received and fetched from the radio chip. As described in the former paragraph, `WiDOMRadio` also provides an interface for

performing carrier sensing. This interface, called `WiDOMCarrierSense`, is shown in Figure 3 and simply enables the `WiDOM` component to turn carrier sensing on and off, whenever necessary.

The `WiDOMRadio` also provides an interface for sending carrier pulses. This interface (`WiDOMCarrierPulse`, detailed in Figure 3) simply enables the WiDOM protocol to turn the sending of a carrier pulse on and off. However, because the radio must be in a transmitter test mode in order to be able to send an unmodulated carrier, the interface also presents commands to switch the radio into this transmitter test mode and to get out from this mode into a normal transmit mode, to be able to transmit data.

Finally, we describe the `WiDOM` component itself. This component implements the WiDOM protocol within a function that receives an integer representing a message type. These message types can be any of the events that make the automaton evolve: every time the `WiDOMClock` signals an alarm, this function is invoked, passing as message type the alarm fired; Whenever a message is queued for sending, the protocol function is invoked with an associated message type; Similarly, if `WiDOMRadio` signals the reception of a packet, the end of a transmission, or detection of a busy channel. The protocol automaton maintains a state variable indicating the current state and, for each state there is a switch statement, where the behaviour is implemented. When the WiDOM protocol function is invoked, the code for the current state is executed. Within each state there is a switch statement for each of the messages that may be received in that state and make the protocol evolve.

4. Response-time Calculations

Let us now introduce the response-time calculations for the WiDOM protocol. This analysis is based on the analysis of non-preemptive static-priority scheduling used in the CAN bus, but subtleties about the synchronization in the protocol requires our schedulability analysis to deal with aspects that the CAN bus did not have to deal with. Consider the sporadic model [1] with a system of n message streams: $\tau_1, \tau_2, \ldots, \tau_n$. Each message stream τ_i is characterized by T_i, D_i and C_i with the interpretation that (i) T_i is the minimum time between two consecutive transmission requests from τ_i, (ii) every time a message from τ_i is requested to be transmitted it needs to finish the transmission at most D_i time units after the request and (iii) C_i denotes the time required to transmit a message from message stream τ_i. We assume $\forall i: D_i \leq T_i$.

Every message has associated protocol overhead that should be added to C_i. The time to transmit a message and performing the tournament when nodes are already synchronized is denoted C_i'. The time to transmit a message and performing the tournament when nodes are not already synchronized is denoted C_i''. We now compute C_i' and C_i'' from C_i and state the response-time equations.

We use a message size of 64 bytes of data (the length of data in a packet is included). Adding 3 bytes used for preamble, the time to transmit a message is given by:

$$C_i = (64+3) \times 8 \times \frac{1}{256000} = 2093\mu s \qquad \textbf{(1)}$$

Inspecting the automaton in Figure 1 (and applying reasoning from [9]), yields:

$$C_i' = C_i + 2H + G + (G+H) \times (npriobits - 1) + 2L + ETG = C_i + 18675\mu s = 20768\mu s \qquad \textbf{(2)}$$

and taking into account also the initial idle time (state 2 in Figure 1) yields:

$$C_i'' = C_i' + F + E + SWX = C_i' + 22274\mu s = = 43042\mu s \qquad \textbf{(3)}$$

Let us assume that the release jitter is equal to zero. We also assume that the granularity of the time is Q_{bit}=4/250000s=16 µs. This is because the radio uses Direct-Sequence Spread-Spectrum such that every 4 bits is modulated as 16 chips and the data rate is 250 kbits/s (this is equivalent to 2Mchip/s). Using these assumptions we obtain that the response time can be calculated (similar to [3]) as a sum of the waiting time w_i and C_i''.

$$R_i = w_i + C_i'' \qquad \textbf{(4)}$$

where C_i'' is defined as in (3). The waiting time is:

$$w_i = B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + F + E + SWX + Q_{bit}}{T_j} \right\rceil \times C_j'' \qquad \textbf{(5)}$$

where *hp(i)* is the set of all message streams with a higher priority than τ_i.

Observe that (5) differs from the analysis used in the CAN bus. With WiDOM, it is necessary to add *F+E+ SWX* which is the time before the next message is dequeued after the previous message has been transmitted. Figure 4 provides the intuition behind this.

Figure 4b shows two computer nodes N_1 and N_2 and how their states change as time progresses. We can see that a message is dequeued when nodes have made the transition to State 5. Let us assume that there is a message stream τ_1 on N_1 and a message stream τ_2 on N_2 and that τ_1 has higher priority than τ_2. If τ_1 and τ_2 make a transmission request at the same time when N_1 and N_2 are in State 1, then we will get one response time of τ_2 and if they make transmission requests just before both nodes make a transition to State 5, τ_2 will get another response time. But neither of them maximize the response time of τ_2.

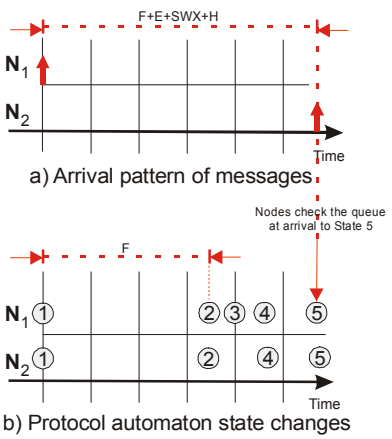

Figure 4.Worst-case arrival pattern and protocol automata state changes.

Instead, it is the arrival pattern illustrated in 4a that maximizes the response time of τ_2.

B_i can be computed as follows:

$$B_i = \begin{cases} 0 & if\ lp(i) = \varnothing \\ \max\{C_j' : j \in lp(i)\} & if\ lp(i) \neq \varnothing \end{cases} \quad (6)$$

where *lp(i)* is the set of all message streams with a lower priority than τ_i. Note that the analysis considers the initial idle time between states 1-5 (Figure 1) to be part of the "message" when we compute interference. This initial idle period should not be included when computing the blocking in (6).

Let us apply the response time analysis to calculate the values according to (1)-(6) in the following example (the response times will be tested empirically in Section 5).

Example 1. Consider m=10 computer nodes with one message stream on each node. Message streams are given values, as shown in Table 1 (all values are given in µs).

Table 1. Message streams for Example 1.

	i=1	*i=2*	*i=3*	*i=4*	*i=5*
T_i	64 000	256 000	512 000	1 024 000	2 048 000
C_i	2 093	2	2 093	2 093	2 093
C_i'	20 768	20	20 768	20 768	20 768
C_i''	43 042	43	43 042	43 042	43 042
B_i	20 768	20	20 768	20 768	20 768
R_i	63 810	192	451 188	967 692	2 000 700
	i=6	*i=7*	*i=8*	*i=9*	*i=10*
T_i	8 192	6 384	32 768 000	32 768 000	32 768 000
C_i	2 093	2 093	2 093	2 093	2 093
C_i'	20 768	20 768	20 768	20 768	20 768
C_i''	43 042	43 042	43 042	43 042	43 042
B_i	20 768	20 768	20 768	20 768	0
R_i	4 109	8 198	14 353 754	28 686 740	30 731 988

We assume that deadline monotonic is used, and assume D_i=T_i. It can be seen that the periods are harmonic. We apply (1)-(6). Observe (Table 1) that the scheduling theory predicts that all deadlines will be met because we obtain $\forall i: R_i \leq T_i$.

5. Experimental Evaluation

Having seen the implementation, we now turn our attention to evaluating its performance. We have the following hypotheses:

1. The implementation of the protocol offers collision-free medium access for data messages.
2. The implementation of the protocol offers prioritized medium access.
3. The response-time analysis equations in (1)-(6) can be used to analyze the response-times of the implementation of the protocol.

§Hypothesis 1 and Hypothesis 2. In order to test Hypothesis 1 and Hypothesis 2, four experiments were set up. We let d denote the maximum distance between any two nodes. We positioned m nodes in a circle such that for every node, the distance to its neighbors with the minimum distance is maximized. The experiment runs as follows. A special node (which is not included in the m nodes) transmits a carrier wave and all other nodes boot. All nodes request to transmit a message and they enter state 1 (from Figure 1). These nodes stay in state 1 (Figure 1) until the special node stops transmitting the carrier. We made the experiment with m=2 nodes and with m=10 nodes. For the case m=2 nodes, all nodes make a new request to transmit a message `random`(0, 255) ms (This means generate a uniformly distributed random number with a minimum value 0 and maximum value 255) after the previous request. For the case m=10 nodes, all nodes make a new request to transmit a message `random`(0,1023) ms time units after the previous request. We also varied the diameter, d=1m and d=10m. Nodes are given numbers from 1 to 10 and their priority is equal id. We send messages with a length of 64 bytes.

Every node has a sequence counter, initialized to 1. The sequence counter is transmitted in every message and then the sequence counter on the node is incremented. Whenever a node received a message it compares the received sequence counter to the previously sequence number received from the same node. If the new sequence number is one greater than the previous sequence number then the receiver concludes that the transmission was collision-free; otherwise the receiver takes the difference between the sequence counters, subtracts one; this is the number of lost messages. Since a collision causes a lost message, this gives us an upper bound on the number of lost messages due to collisions.

We also tested whether prioritization is functioning.

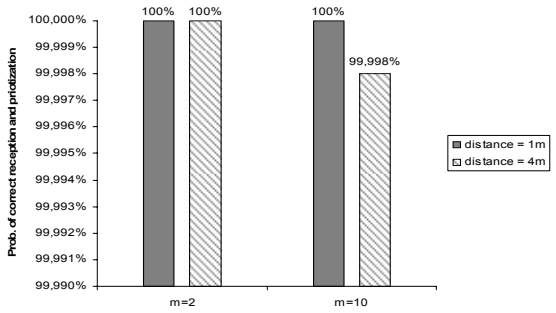

Figure 5. Prioritization and collision free test results.

We did it as follows. When a node sends a message it sends its priority in the data packet. All nodes receive this packet (if they did not receive, it would be considered as a collision, see Hypothesis 1) and if the priority of the winner was less than the priority of this node then it is considered as a prioritization error.

The experiments were run until a total of 50 000 messages were sent, and the data that we obtained is presented in Figure 5. We can see that more than 99.99% of all messages were collision-free and prioritized (observe that messages might get lost due to noise). This corroborates Hypotheses 1 and 2.

§Hypothesis 3. In order to test Hypothesis 3 we set up the following experiment.

One special node (which is not included in the m nodes) sends a carrier for approximately 1 minute. During this minute, the other nodes boot and enter state 1 (from Figure 1). Then the special node stops transmitting the carrier. This causes (i) all m nodes to reset their timers and (ii) all m nodes to request to transmit a message. Then messages are requested to be sent sporadically such that a message stream τ_i made a new transmission request T_i + random(0,5*T_i) ms after the previous request. Every node had exactly one message stream, thus n=m. A node which receives a message reads the current time and calculates the response time. Periodically (every 3 minutes), the special node will send a carrier for a long duration in order to synchronize clocks on the m nodes again. This is performed immediately after receiving a message in order to avoid disturbing a tournament. When the node finishes sending the carrier, all m nodes reset their timers and the queuing time of previously queued messages. This maintains the clocks of all m nodes more tightly synchronized during the length of the experiment and this is necessary in order to measure the response times of messages because the reception and enqueuing times of a message are measured on different nodes.

The experiment was run for the task set in Example 1, until 20 000 messages were transmitted. The results obtained are given in Table 2 (all values are given in µs). Let r_i denote the maximum measured response time.

Table 2. Response times obtained experimentally for settings of Example 1.

	i=1	*i=2*	*i=3*	*i=4*	*i=5*
r_i	60 625	166 840	252 326	289 236	330 590
	i=6	*i=7*	*i=8*	*i=9*	*i=10*
r_i	371 840	454 583	578 368	619 687	702 500

We observe that all measured response times are less than the calculated upper bounds on response times in Example 1. This corroborates Hypothesis 3.

6. Discussion

Recall that the aim of this paper is to show that dominance protocols for wireless channels can be implemented and this is important for real-time communication. The issue of obtaining long range communication is a subject pertaining to telecommunication. It may require other techniques for modulating the prioritization bits or other techniques to configure/interface with the transceiver. However, doing so is beyond the scope of this paper. In order to show the potential range, we have made experiments on detecting pulses of carriers when the receiver is always in receive mode. These experiments suggest that a communication range of 15 m may be possible when WiDOM is used. This is promising, considering that the manufacturer states the range of the CC2420 transceiver is 20-30 meters indoors (all experiments were done indoors).

We set up an experiment to test the ability of our target platform to detect pulses. We set up two nodes: one sender and one receiver, with non-obstructed line-of-sight and they were separated 15 meters apart. The experiment was conducted in an indoor office environment. The nodes were put at the ends of a corridor. The default values as described in the manufacturer's manual for the radio parameters were used and the experiment was conducted with new batteries in the nodes. We used different durations of the pulses sent and we selected the durations as a multiple of 34.722 µs. For every duration, we transmitted 100 000 pulses, counted the number of detected pulse and computed the estimated probability of an undetected pulse. The result is shown in Figure 6.

Observe (in Figure 6) that with duration of 277 µs or less it happens (and it happens often) that the receiver does not detect the pulse. With 486 µs all pulses are detected. Hence, we implemented our protocol based on that assumption.

Detection of pulses is well known to be a difficult problem [10] and it is known that false positives can occur. For this reason we run the experiment again but the sender does not transmit any pulses. We wait for the duration of the experiments and detect the number of detected pulses. We find that no pulses are detected and

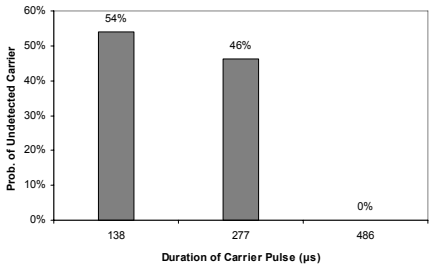

Figure 6. Probability of undetected carrier as a function of the carrier pulse duration.

use that to obtain an estimate of the probability of a false positive detection. It is zero.

7. Conclusions and Future Work

We have shown that a wireless dominance protocol can be implemented. It is collision-free and prioritized and hence it allows us to compute the response times. The protocol is intended for short-range communication in small geographic areas and for this purpose, our protocol works reliably. We observed that the tournament functions correctly in more than 99.99% of the cases. The overhead is to a large extent due to the transition time between transmission and reception. This is a technological parameter that can be improved with better hardware as witnessed by the fact that the Hiperlan standard [16] required a switching time of 2µs. If such a transceiver was available and offered the flexibility to design the MAC protocol in software, then the overhead of the protocol could be reduced by two orders of magnitude. We are currently looking for such hardware.

Finally, we point out that besides from being naturally useful for scheduling sporadic real-time traffic, WiDOM also supports certain types of group communication [17].

8. Acknowledgements

We are grateful to the reviewers for suggested improvements of the paper. This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para Ciência e Tecnologia - FCT) and the ARTIST2 Network of Excellence on Embedded Systems Design.

9. References

[1] Mok, A. "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment Electrical Engineering and Computer Science, In Electrical Engineering and Computer Science, Cambridge, Mass., 1983, Massachusetts Institute of Technology, Cambridge, Mass., 1983.

[2] Baruah, S.K., Mok, A.K. and Rosier, A.K., "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". In IEEE Real-Time Systems Symposium, 1990, 182-190.

[3] Tindell, K., Hansson, H. and Wellings, A., "Analysing real-time communications: controller area network (CAN)". In 15th Real-Time Systems Symposium (RTSS'94), 1994, 259-263.

[4] Bosch, "CAN Specification, ver. 2.0, Robert Bosch GmbH, Stuttgart," 1991. Online:http://www.semiconductors.bosch.de/pdf/can2spec.pdf.

[5] Mok, A.K. and Ward, S. "Distributed Broadcast Channel Access". Computer Networks, 3. 327-335, 1979.

[6] You, T., Yeh, C.-H. and Hassanein, H.S., "CSMA/IC: A New Class of Collision-free MAC Protocols for Ad Hoc Wireless Networks". In 8th IEEE International Symposium on Computers and Communication, 2003, 843-848.

[7] You, T., Yeh, C.-H. and Hassanein, H.S., "A New Class of Collision - Prevention MAC Protocols for Ad Hoc Wireless Networks". In IEEE International Conference on Communications, 2003.

[8] You, T., Yeh, C.-H. and Hassanein, H.S., "BROADEN: An efficient collision-free MAC protocol for ad hoc wireless networks". In IEEE International Conference on Local Computer Networks, 2003.

[9] Andersson, B. and Tovar, E., "Static-Priority Scheduling of Sporadic Messages on a Wireless Channel". In International Conference on Principles of Distributed Systems (OPODIS´05), Pisa, Italy, 2005.

[10] Tobagi, F.A. and Kleinrock, L. "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution". IEEE Trans. on Communication, 23 (12). 1417-1433, 1975.

[11] Shannon, C.E. "A mathematical theory of communication". Bell System Technical Journal, 27. 379-423 and 623-656, 1948.

[12] http://www.xbow.com/, XBow Inc.

[13] http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf, In.

[14] Hill, J. "System Architecture for Wireless Sensor Networks Computer Science Department, In Computer Science Department, 2003, University of California, Berkeley, 2003.

[15] Gay, D., Welsh, M., Levis, P., Brewer, E., Von Behren, R. and Culler, D., "The nesC language: A holistic approach to networked embedded systems". In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'03), 2003, 1-11.

[16] "Broadband Radio Access Networks(BRAN);HIPERACCESS; PHY protocol specification", 1998.

[17] Andersson, B., Pereira, N. and Tovar, E., "Using a Prioritized MAC Protocol to Efficiently Compute Aggregated Quantities". To appear in 5th Intl Workshop on Real Time Networks (RTN'06), Dresden, Germany, 2006.