Implementation of a MAC Protocol for QoS Support in Wireless Sensor Networks

Petcharat Suriyachai, Utz Roedig, Andrew Scott
InfoLab21, Lancaster University, UK
{p.suriyachai, u.roedig, a.scott}@lancaster.ac.uk

***Abstract*— Future application areas of wireless sensor networks (WSNs) may include industrial process automation, aircraft control systems or patient monitoring in hospitals. Such applications require predictable quality of service in terms of message transfer delay and reliability. Performance of WSN data transport is to a large extent defined by the employed medium access control (MAC) protocol. Currently, no existing MAC protocol is capable of supporting WSN applications that require deterministic data transport performance. In this paper, we present the implementation of a new WSN MAC protocol that is able to give deterministic bounds for message transfer delay and reliability. The protocol is implemented on TinyOS 2.x for a Tmote Sky node using the CC2420 transceiver. Our implementation shows that a deterministic MAC protocol with reasonable energy consumption patterns is practical.**

I. Introduction

Potential applications supported by wireless sensor networks (WSNs) may include industrial process automation, aircraft control systems and patient monitoring in hospitals. These applications necessitate predictable quality of service in terms of message transfer delay and reliability. Timely analysis of sensor data must always be possible to decide whether an action must be performed. To achieve such a goal, WSNs must be dimensioned properly. More specifically, it should be possible to dimension a WSN such that an upper bound D for the end-to-end message transfer delay can be given to guarantee a certain response time for the system. Moreover, the network should be dimensioned such that a lower bound for the end-to-end data delivery reliability R can be given too.

Different methods exist to dimension a WSN such that a bound for D and R can be given (for example, [7]). These dimensioning methods rely on the assumption that an upper bound for the node-to-node forwarding delay d and a lower bound for the node-to-node forwarding reliability r can be achieved. However, in a practical WSN deployment this requirement is not trivial to implement.

Node-to-node forwarding characteristics are defined by the medium access control (MAC) protocol. Unfortunately, no existing MAC protocol is able to provide guaranteed bounds on message forwarding delay and reliability. Most available MAC protocols for WSNs are designed to be energy efficient rather than to provide a deterministic forwarding behavior [5]. Few MAC protocols are designed to optimize the node-to-node forwarding delay and/or reliability [4], [6]. Although these solutions may improve message transport delay and/or reliability, they fail to give guaranteed bounds.

In this paper, we present an implementation of a MAC protocol that can provide guarantees regarding *node-to-node message transfer delay* d and *node-to-node message transfer reliability* r under normal operational conditions. In addition, the protocol can give an upper bound on the node's communication related energy consumption e. Therefore, the protocol can be used in conjunction with existing network dimensioning tools to construct WSNs that are able to provide strict QoS guarantees. The specific contributions of this paper are:

- A description of the MAC protocol and its detailed implementation are given.
- Lessons learned from the implementation process are discussed. We show that state-of-the-art event driven WSN operating systems such as TinyOS are not suitable to host a MAC protocol with deterministic behavior.
- The achievable bounds regarding d , r and e on the selected sensor node platform are studied.

The paper is organized as follows. The next section describes existing MAC protocols that address timely and reliable data delivery. Section III presents our proposed deterministic MAC protocol. Section IV provides a detailed implementation of the MAC protocol on the Tmote Sky platform. Section V reports on the achievable performance and lessons learned from the implementation. Section VI concludes the paper.

II. Related Work

The design of a WSN is often started with the definition of the medium access control protocol as it fundamentally defines the energy consumption properties and capabilities of the network. Additional network mechanisms such as routing or topology control are commonly integrated into the MAC protocol or closely aligned with its design choices. Most MAC protocols described in the literature aim to minimize energy consumption at the expense of high data delivery latency and/or sometimes low reliability (for example, [5]). Hence, these protocols help to conserve battery power but are incapable of delivering an upper bound for message transfer delay or reliability. Consequently, these protocols cannot provide QoS support. Some MAC protocols are designed to minimize delay while optimizing energy consumption (for example, [4]). The resulting MAC protocols provide low latency on data transfer, but they are incapable of providing strict performance guarantees. The f-MAC [3] protocol can provide deterministic message transfer delay but fails to implement any energy saving techniques.

The study in [1] assumes a network layout in a hexagonal shape, and a TDMA protocol is constructed on top of this topology. Moreover, this study assumes that only neighboring nodes in the topology interfere, and thus some slots can be used by different nodes at the same time. Based on these assumptions, a carefully designed schedule is devised to obtain the minimum possible bound on message transfer delay. However, the cell-based topology may be impractical to construct even in a planned deployment. Furthermore, message transfer reliability is not addressed by this MAC protocol.

PEDAMACS [2] defines a MAC protocol including topology control and routing mechanisms. An upper bound for the message transfer delay can be analytically determined before network deployment. The sink centrally calculates a transmission schedule for each node, taking interference patterns into account. Subsequently, the sink distributes the scheduling information to all nodes in a single hop using a high-power radio. Thus, a collision-free transmission is achieved within the network. However, the assumption that a high-power sink is available limits the usability of this solution. In addition, PEDAMACS does not address message transport reliability.

A few MAC protocols [8], [9] achieve QoS provisioning by adopting traffic differentiation and prioritization, and thus differ from our work that assumes homogeneous traffic. Moreover, these studies are evaluated via simulation and could provide only *average* delay performance.

III. TDMA-Based MAC Protocol

In order to support QoS in terms of message transfer delay and reliability, we propose a Time Division Multiple Access (TDMA) based Medium Access Control (MAC) protocol that also handles routing. Topology awareness within the MAC layer is vital as the node-to-node transport performance can only be constructed deterministically if the degree of interference from neighboring nodes is known and can be controlled.

A. Goals and Assumptions

We assume that the wireless sensor network has a tree configuration rooted at the sink. Messages requiring deterministic transport are traveling from the sensor nodes to the sink. Control data traveling from the sink to the nodes is considered to be non critical. We assume that a relatively small number of nodes are deployed at chosen locations. An example application scenario that our MAC protocol can be applied to is process monitoring and control in a production plant. In such a setting, the number of nodes is small, and the network topology including node locations can be planned.

The protocol has to ensure that an upper bound for the node-to-node forwarding delay d and a lower bound for the node-to-node forwarding reliability r can be given. This guarantee should be given for data traveling from sensor nodes toward the sink.

B. Data Transfer Delay

All nodes in the network are assumed to be time synchronized; Section IV explains our lightweight time synchronization scheme. The time axis is divided into fixed-length base units called epochs. Each epoch E is subdivided into $k \cdot n$ time slots for a network of at most n sensor nodes. Each node is assigned k exclusive slots per epoch E to successfully transmit one message. Slots are large enough to transmit a packet of maximum payload and to receive an acknowledgment from the next-hop receiver. The protocol is collision-free. A node has k attempts per epoch to deliver a message and to achieve the required node-to-node message transfer reliability. Thus, the worst-case node-to-node transfer latency d for all nodes is defined by the epoch size E:

$$d \leq E = k \cdot n \tag{1}$$

Simple routing is performed by the MAC layer. Sensor readings are routed as *sensor data messages* up-tree toward the sink. If necessary, sensor data messages are queued in the *sensor data FIFO buffer*. *Sink data messages* used to set up and control the sensing tasks can be broadcast from the sink to all nodes in the network. Sink data messages are queued in the *sink data FIFO buffer* when necessary. Sink data messages are transported when no data messages need forwarding. Each node is aware of its position in the tree and knows the slot numbers assigned to its child nodes (to handle sensor data messages) and the parent node (to handle sink data messages).

C. Data Transfer Reliability

The slot assignment of each node can be pre-configured as a planned deployment is assumed. Each node n_i is identified by a unique identification number i with $0 \leq i < n$. Each node n_i owns exclusively k transmission time slots $s_{i,j}$ with $0 \leq j < k$ in each epoch E. The transmission slots are assigned uniformly across the epoch: $s_{i,j} = i + n \cdot j$. Therefore, retransmission slots obtain the maximum temporal distance, helping to counter burst errors in the channel. Nevertheless, other distribution types are acceptable if they render the slot assignment fixed.

Each node must transmit a packet within its first transmission slot in the epoch; if no sink or sensor data is queued for transmission, a simple *control message* is sent to the parent node for the purpose of connectivity testing. A packet with a non-broadcast destination address requires an acknowledgment. As mentioned previously, slots are large enough to contain a packet and its subsequent acknowledgment. If a transmission is not acknowledged, a node will retransmit the message within the next slot of the k transmission slots assigned to the node. If a node does not receive a message in the expected slot, it will start listening on the next transmission slot assigned to this node within the epoch. Figure 1 displays an example for node n_1 in a tree topology when $k = 2$, $n = 20$ and $m = 40$. The node n_1 has two child nodes: n_2 and n_5, and its parent node is n_0. Slots in which the radio of node n_1 needs to be active are shown solid. Slots that might become active in case of packet losses are shown shaded. Slots in which the radio is always in sleep mode are depicted empty.

There are various techniques to determine the value k depending on application requirements and different channel

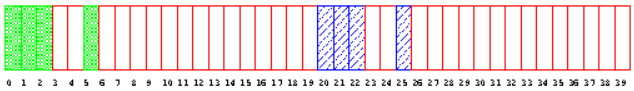

Fig. 1. Slot usage of n_1 whose parent node is $n_0$and child nodes are n_2 and n_5.

models assumed. For example, if the maximum packet size l is known and a simple channel model with independent bit error rate B is assumed, the achievable node-to-node delivery ratio or *message transfer reliability* r can be calculated as:

$$r \geq (1-B)^{8\times l} \times \left(1-(1-B)^{8\times l}\right)^{(k-1)} \quad (2)$$

It is important to emphasize the normality assumption as unexpected conditions, such as signal jamming, would increase the bit error rate B and thus invalidate the proposed calculation. In such an adverse case, the measured message transfer reliability becomes less than the guaranteed theoretical reliability bound r.

D. Energy Consumption

Communication related energy consumption in the context of WSNs is generally specified by the so called duty cycle. The duty cycle is the ratio of radio on time Δ_{on} to the sum of radio on time Δ_{on} and radio off time Δ_{off}. Formally, the duty cycle P is defined as $P = \Delta_{on}/(\Delta_{on}+\Delta_{off})$. The radio on time represents the time the radio is sending, receiving or listening. These three modes of operation require roughly the same energy. The radio off time represents the time the radio is in an energy-efficient sleep mode.

The minimum duty cycle P_i of node n_i depends on the number of child nodes c_i attached to it and the overall epoch length E. Additional energy will be consumed if retransmissions are necessary. A lower bound for the duty cycle of the presented scheme can be given for optimal conditions where retransmissions are not required. A node n_i is only active in the first of the slots assigned to itself, the parent node and child nodes. Thus, its lower bound for the duty cycle and its energy consumption e_{min} is given as

$$e_{min} = P_i = (2+c_i)/E \quad (3)$$

In contrast, an upper bound occurs when all retransmission slots are used, and the node is active in all slots assigned to itself, the parent node and child nodes. Consequently, an upper bound for the duty cycle and its energy consumption e_{max} is easily obtained as:

$$e_{max} = P_i = k\cdot(2+c_i)/E \quad (4)$$

Thus, the energy consumption of a node is bounded by:

$$e_{min} \leq e \leq e_{max} \quad (5)$$

For example, node n_1 in the aforementioned topology with $k = 2$, $n = 20$ has a lower bound for the duty cycle of $P_1 = 4/40 = 10\%$. The upper bound in this example is $P_1 = 8/40 = 20\%$. Obviously, constant operation at the upper bound would dramatically increase energy consumption and consequently reduce network lifetime.

Two mechanisms can be employed to influence the energy consumption pattern of a node. First, the topology can be chosen such that the desired energy usage pattern emerges. Second, the epoch length E can be extended by adding additional unused slots.

IV. Implementation

The MAC protocol was implemented on TinyOS 2.0.2 for the TelosB platform, which incorporates a CC2420 radio transceiver. The aim was to reuse existing TinyOS components and interfaces as much as possible to facilitate seamless integration of the MAC protocol with existing TinyOS applications and programming infrastructure. However, this was not always possible, and some low-level components had to be modified to obtain the required absolute deterministic MAC behavior. For example, the usage of TinyOS tasks within the MAC component must be avoided as TinyOS cannot guarantee when tasks are executed.

Figure 2 shows the component graph of the MAC layer implementation. The standard TinyOS interfaces to send and receive messages are retained. Applications access the radio resource through the ActiveMessage (AM) layer which comprises the CC2420ActiveMessage component. The MAC protocol component provides basic communication interfaces such as Send and Receive to this AM layer and connects to the physical layer. Our implementation was tailored to the CC2420 radio used in the TelosB platform and, hence, the existing TinyOS CC2420 components were used. It has to be noted that the MAC protocol provides interfaces for functions that are not useful for TDMA protocol (for example, RadioBackoff). These functions are retained purely for compatibility reasons.

A. Frame Format

The MAC protocol uses the frame format defined by the IEEE 802.15.4 standard in order to be compatible with the IEEE 802.15.4 CC2420 radio. In addition, this use allows us to exploit hardware features such as the automatic acknowledgment of the CC2420 radio transceiver.

A MAC frame has a minimum size of $12bytes$ and a maximum size of $122bytes$. The maximum frame size is dictated by the CC2420 design. The MAC header consists of a $2byte$ Frame Control Field (FCF), a $1byte$ Data Sequence Number (DSN), a $2byte$ destination Personal Area Network (PAN) address, a $2byte$ destination address, a $2byte$ source address and a $1byte$ field identifying the TinyOS message type. The payload can have a maximum size of $110bytes$, and the MAC footer contains a $2byte$ Frame Check Sequence (FCS). Another $6byte$ physical layer header is added in front of the MAC frame when the message is transmitted. Thus, messages on the air have a minimum length of $18bytes$ and a maximum length of $128bytes$.

The first $3bits$ of the FCF define the frame type. The MAC protocol uses the three types: *data*, *acknowledgment*

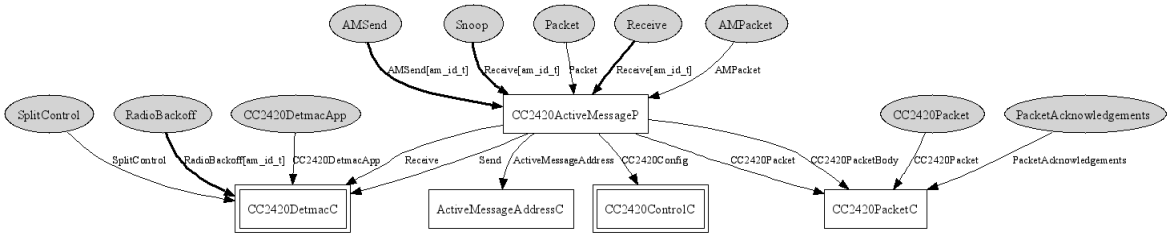

Fig. 2. Component Graph in TinyOS

and *MAC command*, which are defined in the IEEE 802.15.4 standard. Sensor data or sink data messages use the type *data*. Control messages that are transmitted if there is no data to be transferred use the type *MAC command*. Acknowledgments are sent using the type *acknowledgment*. The destination PAN address field is not used by the MAC protocol but is retained for compatibility with the TinyOS message format. The TinyOS message type field is used to identify a TinyOS component. As there may be multiple components sharing one radio, the TinyOS ActiveMessage layer uses the type field for data multiplexing.

The MAC protocol makes use of automatic acknowledgments provided by the CC2420 transceiver. If the destination of a data frame is not the broadcast address, the protocol then sets the acknowledgment request field in the Frame Control Field. Upon receipt of such a message, the CC2420 transceiver immediately sends an acknowledgment message. The acknowledgment follows the IEEE802.15.4 acknowledgment frame format, and its size is only $11byte$s. The MAC protocol also exploits other hardware support of the CC2420 chip. For instance, the $2byte$ Frame Check sequence (FCS) is automatically appended at the end of the frame and verified by the hardware in the transmit and receive modes, respectively.

B. Slot Size

The selected slot size T_s should be as small as possible to reduce the epoch size E and ultimately to reduce the end-to-end delay. To determine the necessary slot size, two different cases must be distinguished: the slot used for transmission and the slot used for reception.

If the slot is used for transmission, time to complete the following steps must be allocated. First, the message has to be transferred from the MAC layers data FIFO buffer into the buffer of the CC2420 transceiver (transfer time t_{ts}). Then, the message has to be transmitted (transmit time t_{xm}). For unicast transmissions, the receiver needs some time t_{pm} to process the message and initiate the transmission of an acknowledgment message (processing time t_{pm}). The acknowledgment is transmitted (transmit time t_{xa}). Finally, the node needs time to transfer and process the acknowledgment from the CC2420 transceiver and to perform the associated actions for received/missed acknowledgment (processing time t_{pa}). A small guard time is required at the beginning and end of the slot to compensate for clock drifts between nodes (guard time t_g). Note that the propagation delay is neglected as it is small compared to the other time values. Thus, the minimum size of a transmission slot is given as $T_{st} = t_g + t_{ts} + t_{xm} + t_{pm} + t_{xa} + t_{pa} + t_g$.

If the slot is used for reception, time to complete the following steps must be allocated. First, the message is received (transmit time t_{xm}). The incoming message must be processed (processing time t_{pm}), and if necessary an acknowledgment has to be transmitted (transmit time t_{xa}). Thereafter, the micro controller has to be informed, and the message is transferred from the CC2420 buffer to the micro controller where it is either enqueued in the MAC layer data FIFO buffer for forwarding or in the MAC layer application FIFO buffer (transfer time t_{tr}). Again, a small guard time is required at the beginning and the end of the slot (guard time t_g). Thus, the minimum size of a reception slot is given as $T_{sr} = t_g + t_{xm} + t_{pm} + t_{xa} + t_{tr} + t_g$.

The required slot size can now be determined by aligning T_{st} and T_{sr}. Thus, we obtain

$$T_s = t_g + t_{ts} + t_{xm} + t_{pm} + t_{xa} + t_{tr} + t_g \qquad (6)$$

as t_{pa} on sender side and t_{tr} on receiver side are executed in parallel and $t_{tr} > t_{pa}$.

A lower bound for T_s can be given by determining the best theoretical possible execution time for all steps that have to be carried out in a slot. To calculate this lower bound, we first assume that the message processing time is negligibly small ($t_{pm} = 0$). Second, we assume that the message transfer times t_{ts} and t_{tr} between CPU and CC2420 are solely determined by the speed of the SPI interface interconnecting them (again, we neglect processing time). For the TinyOS maximum message size of a $28byte$ payload, a $39byte$ MAC frame is transferred into the transceiver buffer at the transmitter side while a $41byte$ MAC frame is transferred out of the transceiver buffer at the receiver side. Two additional bytes account for the FCS, which is automatically generated by the CC2420 at the transmitter side. The node's SPI interface can support at a maximum rate of $500kbits/s$ on both sides. Therefore, we obtain $t_{ts} = 0.62ms$ and $t_{tr} = 0.66ms$. The CC2420 transmission speed is $250kbits/s$, and with a payload size of $28bytes$ we obtain $t_{xm} = 1.47ms$. The acknowledgment has a fixed total size of $11bytes$ and $t_{xa} = 0.35ms$. We assume perfect time synchronization for this calculation, and we use $t_g = 0ms$. Thus, a theoretical lower bound for the slot size on the TelosB mote can be given as $T_s = 3.10ms$. Note that switching between sleep and active states can be

performed within the inactive slot before or after a block of active slots and consequently does not need to be included in the calculation.

Unfortunately, the theoretical minimum slot size cannot be achieved in practice as processing times have to be taken into account. For example, the TinyOS interface to the SPI bus introduces a large processing overhead that reduces the achievable SPI bus transfer rates dramatically. Measurements show an achievable SPI transfer speed of $173.91kbits/s$ ($46\mu s/byte$). For a payload of $x\,byte$, we determined the following values as practical: $t_g = 0.15ms$, $t_{ts} = 0.31ms + (x{+}11){\cdot}46\mu s$, $t_{xm} = 0.40ms{+}(x{+}13){\cdot}32\mu s$, $t_{pm} = 0.21ms$, $t_{xa} = 0.35ms$ and $t_{tr} = 1.28ms + (x + 5) \cdot 46\mu s$. With the limited TinyOS SPI rate, other determined values and a payload size of $28bytes$, the minimum feasible slot size is $T_s = 6.97ms$.

In our experiments, a slot size of $T_s = 9.765ms$ is used to accommodate a payload of up to $50bytes$. TinyOS hardware clocks operate in binary multiples of a tick, and therefore 1024 TinyOS milliseconds correspond to one real second. Consequently, the selected slot size is 10 TinyOS milliseconds long, which corresponds to $9.765ms$.

The slot size could be reduced by organizing the slot allocation differently. t_{ts} and t_{tr} could be performed outside the slot in adjacent inactive slots. However, this would require that the topology is arranged to guarantee all active slots are surrounded by inactive slots. This potential optimization is currently not used in our protocol implementation.

C. Time Synchronization

Our TDMA-based protocol requires time synchronization among nodes. The protocol could be used together with an already available time synchronization method provided that the mechanism would not obstruct data transmission when required. However, to obtain a lightweight implementation, we decided to use the existing constant exchange of data and control messages for time synchronization.

Each node in the network synchronizes its clock with the parent node in the tree topology. When a node receives a message (or overhears the message if it is traveling up-tree) generated by its parent, it can calculate the current time in the epoch as it knows which slot in the epoch is used by the parent node. All nodes can use one message in each epoch for time synchronization as each node has to transmit a message in every epoch. Thus, time synchronization is carried out frequently, which ensures an accurate time base in the network. We are aware that our time synchronization method will be problematical for topologies with a long hop distance as synchronization errors will propagate. However, the described MAC protocol is intended to be used in relatively small and controlled deployments. As observed in our experiments, time synchronization can be achieved accurately enough with the described method.

After being turned on, the sink node initializes the system's clock and broadcasts one message in its assigned slot every epoch. After a non-sink node is turned on, it listens for a message from its parent and discards all other packets. Upon receiving the parent's message, the node determines the epoch start and begins the radio duty cycle (normal MAC operation). To find a reference point to start its MAC timer, the MAC protocol uses the system's current time t_c and a timestamp t_{sfd} at which the Start Frame Delimiter (SFD) of the packet is received. The start time t of the slot can be calculated using the known transceiver set-up time and transmission duration of the SFD $t_{xsfd} = 0.40ms$, the selected packet length and Equation (6) $t = t_c - (t_{sfd} - t_{xsfd}) - t_{ts} - t_g$. If there is no packet loss, the synchronization of the network can be completed within H epochs, with H being the depth of the tree. After a successful time synchronization, a node signals to the application layer that the radio is ready and may be used for data transmission.

Deviations in a hardware clock can accumulate and adversely impact any TDMA protocol. To deal with this problem, the protocol makes a clock adjustment in each epoch. Upon receiving a packet from its parent, the node uses the aforementioned method to re-calculate the start time of the slot. Thereafter, the MAC timer is adjusted accordingly. Unfortunately, the clock adjustment needs processing time and can therefore only be performed when the MAC layer is not busy. To simplify implementation, it was decided to prolong the epoch E by an additional unused slot ($E = k{\cdot}n{+}1$). This slot at the end of an epoch is used for time re-synchronization. Note that the additional slot is not needed if the total number of actual nodes in the topology is less than n, the maximum number of nodes the MAC protocol can support. If no transmissions are received from the parent node for 5 consecutive epochs, the node assumes that time synchronization was lost. In this case, the node switches the transceiver into a permanent listen state until a packet is received from the parent node and re-synchronization is obtained.

V. Evaluation

To test the achievable performance bounds, a network of 15 nodes structured as a binary tree was deployed. We found that the calculated bounds regarding d, r and e using Equations (1), (2) and (5) could actually be met in the deployment, as presented in Table I

Performance Parameters	Bounds
d	$156.24ms$
r	99%
e	25%

TABLE I
NODE-TO-NODE PERFORMANCE BOUNDS OBSERVED

A node-to-node delay bound of $d = 156.24ms$ was achieved as a slot size of $T_s = 9.765ms$ was selected to accommodate up to $50bytes$ of payload and the maximum number of nodes was set to $n = 16$ in the implementation. The deployment scenario required $k = 1$ to achieve a node-to-node transmission reliability of more than 99%. Since a binary tree was used, a duty cycle of $P = 4/16 = 25\%$ was accomplished. This duty cycle is considerably higher than

that commonly provided by energy-efficient MAC protocols having a similar transmission delay but taking the contention-based access route. Clearly, the deterministic behavior of our protocol is obtained at the expense of energy consumption.

A. Lessons Learned

TinyOS uses split-phase operations. Here, the program calls an operation, the call returns immediately and the called abstraction issues a callback when it completes. The called abstraction may use a TinyOS task for the callback, which is then placed in a task queue for execution. For example, to switch the radio on, the MAC protocol calls a specific function. At the end of this function a task is posted. When the task is executed, a callback function in the MAC layer is executed, which informs the MAC layer that the radio is now successfully switched on. As other tasks might be executed before this particular task, there is no guarantee how long it will actually take to switch the radio on. The TDMA protocol requires a deterministic behavior, and consequently split-phase operations including tasks cannot be supported in the MAC layer. Basically, no tasks can be used during the time-critical operations of the MAC layer. To comply with this requirement, two existing TinyOS code elements had to be modified to remove tasks. First, the task that signals successful activation of the radio was removed. It was determined that radio activation is always completed within $t = 2.56ms$. Thus, instead of using the task to signal when the radio is ready, it is assumed that after $t = 2.56ms$ radio activation is complete. Second, the task that signals successful locking of the SPI interface was removed as the only component using SPI in the prototype system is the radio.

In summary, significant modifications to TinyOS are necessary to implement a deterministic MAC protocol efficiently. These modifications require changing the TinyOS programming model (for example, split-phase operation), and therefore TinyOS might not be the best choice of programming environment for implementing QoS-support MAC protocols.

B. Possible Improvements

The epoch size E depends on the number of nodes n in the network and, hence, the protocol does not scale to large deployments. However, the protocol is suitable for many realistic deployments where data delivery must be timely and reliable. For example, a process automation application deployed on a factory floor might use only a few dozen nodes, and many nodes will sometimes be in communication range of each other. Thus, exclusive slots for each node are required to guarantee collision-free operation. To make the protocol usable for larger deployments, the protocol could reuse slots but must guarantee that two nodes sharing the same slot are never in communication range of each other. In addition, different transmission frequencies could be applied to create separate collision domains (for example, the CC2420 radio supports operation on 16 different carrier frequencies).

To implement the deterministic behavior of the MAC protocol, modification of the existing low level components was required to avoid the use of tasks in critical sections. As shown, the TelosB platform could theoretically support a slot size of $3.10ms$. However, due to the overheads introduced by TinyOS the best achievable slot size is currently $6.97ms$. This could be reduced by modifying the SPI programming abstraction in TinyOS, responsible for most of the introduced overheads.

VI. Conclusion

In this paper, a TDMA-based MAC protocol[1] is proposed to serve potential WSN applications that demand predictable quality of service regarding message transfer delay and reliability. The collision-free protocol exploits topology knowledge and is integrated with a routing mechanism and retransmission scheme to ensure that an upper bound for the node-to-node forwarding delay d and a lower bound for the node-to-node forwarding reliability r can be given. In addition, the energy consumption can be bounded as well. Detailed implementation of the protocol on TinyOS 2.x for a Tmote Sky node using the CC2420 transceiver is provided. Moreover, some features of TinyOS are discussed to illustrate its limitations in hosting a MAC protocol with deterministic behavior.

As for future work, the protocol could be integrated with a dimensioning tool which utilizes its node-to-node delay bound d to determine an end-to-end delay bound D.

References

[1] K. Shashi Prabh, "Real-Time Wireless Sensor Networks," Ph.D. Thesis, Department of Computer Science, University of Virginia, Charlottesville, VA, USA, 2007.

[2] S. Coleri-Ergen and P. Varaiya, "PEDAMACS: Power Efficient and Delay Aware Medium Access Protocol for Sensor Networks," *IEEE Trans. Mobile Comput*., vol. 5, pp. 920-930, Jul. 2006.

[3] U. Roedig, A. Barroso and C. J. Sreenan, "f-MAC: A Deterministic Media Access Control Protocol Without Time Synchronization," in *Proc. 3rd European Workshop Wireless Sensor Networks*, Zurich, Switzerland, 2006, pp. 276-291.

[4] G. Lu, B. Krishnamachari and C. Raghavendra, "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Sensor Networks," in *Proc. 4th Int. Workshop Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks*, Santa Fe, NM, USA, 2004, pp. 224-231.

[5] W. Ye, J. Heidemann and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks,"*IEEE/ACM Trans. Netw.*, vol. 12, pp. 493-506, Jun. 2004.

[6] R. Biswas, V. Jain, C. Ghosh and D.P. Agrawal, "On-Demand Reliable Medium Access in Sensor Networks," in *Proc. 7th IEEE Int. Symp. a World of Wireless, Mobile and Multimedia Networks*, Buffalo, NY, USA, 2006, pp. 251-257.

[7] J. Schmitt and U. Roedig, "Sensor Network Calculus - A Framework for Worst-Case Analysis," in *Proc. 1st Int. Conf. Distributed Computing in Sensor Systems*, Marina del Rey, CA, USA, 2005, pp. 141-154.

[8] N. Saxena, A. Roy and J. Shin, "A QoS-based Energy-aware MAC Protocol for Wireless Multimedia Sensor Networks," in*Proc. IEEE 67th Vehicular Technology Conf.: VTC2008-Spring*, Marina Bay, Singapore, 2008, pp. 183-187.

[9] Y. Lui, I. Elhanany and H. Qi , "An Energy-Efficient QoS-Aware Media Access Control Protocol for Wireless Sensor Networks," in*Proc. 2nd IEEE Int. Conf. Mobile Ad-hoc and Sensor Systems*, Washington, DC, USA, 2005, pp. 189-191.

[1] This work has been partially supported by the European Commission under the contract FP7-ICT-224282 (GINSENG).