Piggy-Backing Link Quality Measurements to IEEE 802.15.4 Acknowledgements

Wolf-Bastian Pöttner, Sebastian Schildt, Daniel Meyer and Lars Wolf
Institute of Operating Systems and Computer Networks
Technische Universität Braunschweig,
Braunschweig, Germany
Email: [poettner|schildt|dmeyer|wolf]@ibr.cs.tu-bs.de

***Abstract*—In this paper we present an approach to piggy back link quality measurements to IEEE 802.15.4 acknowledgement frames by generating acknowledgements in software instead of relying on hardware support. We show that the software ACKs can be sent meeting the timing constraints in IEEE 802.15.4. This allows for a standard conforming, energy neutral dissemination of link quality related information in IEEE 802.15.4 networks. This information is available at no cost when transmitting data and can be used as input for various schemes for adaptive transmission power control and to assess the current channel quality.**

I. Introduction

Mechanisms such as adaptive transmission power control or network monitoring in wireless networks require feedback about the current channel quality. This information can either be disseminated out-of-band by using a special packet type, or it can be transmitted in-band by piggy-backing the desired information on existing messages, which reduces overhead and energy consumption.

The idea behind the work presented here is to provide feedback about the reception quality of a frame to the sender by recycling three unused bits in the IEEE 802.15.4 acknowledgement frame. Most packet-based radio chips employed on sensor nodes to date provide functions to automatically generate and send such ACK frames. However, they do not provide means to alter the contents of the ACK. Therefore, ACKs including feedback have to be generated and sent in software. In order to remain compatible with IEEE 802.15.4 compliant nodes, our approach has to meet the stringent time limits of the standard while also using the original frame format. The energy consumption of sending and receiving packets should not be increased to make the approach usable in real-world deployments.

For our prototype implementation we have used the well-known T-Mote Sky sensor node which features the TI CC2420 IEEE 802.15.4 transceiver [7]. Since this radio is used on many other sensor nodes as well, our results should also apply to these nodes. We have used the Contiki[1] operating system on the nodes and base our work on the existing device driver. However, the principles of our solution can also be implemented for other operating systems and hardware platforms.

The remainder of the paper is structured as follows: In section II we introduce some related work. Section III outlines our concept and introduces the constraints set by the hardware and the IEEE 802.15.4 standard. In section IV we present our implementation which we will evaluate in section V. Finally, in section VI we wrap up and present some concluding observations.

II. Related Work

Most published mechanisms to control the radio transmission power rely on receiver feedback regarding the reception quality. ATPC [5] is an adaptive transmission power control scheme that relies on receiver feedback to adapt the outgoing power based on a predictive model. If a node receives a packet where the desired link quality differs from the actual link quality, an explicit notification packet is sent to the sender to be used as input for the predictive model. This transmission of the explicit notification packet provides the necessary feedback but also increases energy consumption and takes longer compared to ACK frames.

Xiao et al. [10] have created a continuous receiver feedback-based adaptive transmission power control algorithm that provides feedback in acknowledgement packets. Transmission power is increased or decreased according to information provided in the ACK packets. The authors do not mention if IEEE 802.15.4 acknowledgement messages are extended or if additional messages are sent. Also no information regarding the timing of the ACKs are given.

TinyOS[2] uses software generated IEEE 802.15.4 acknowledgements in favour of hardware generated to avoid a phenomenon of so-called false acknowledgements [9]. The implementation first reads out the packet from the radio, decides if an ACK should be sent and then signals the CC2420 radio to generate and send an ACK message. However, this method neither allows to embed custom information into the ACK frame nor does it comply with the stringent timing of the standard.

III. Basics

The CC2420 is able to generate and send ACKs for data frames in hardware based on the outcome of the

[1] http://www.sics.se/contiki/

[2] http://www.tinyos.net/

integrated address recognition and the verification of the Frame Check Sequence (FCS, a CRC checksum in the last two bytes of IEEE 802.15.4 frames). The process is fully automatic and application code is not involved which also precludes the modification of the ACK frame. The existing Contiki radio driver for the CC2420 uses the CC2420's AutoAck feature to handle acknowledgements.

Our approach is to switch off the AutoAck feature and generate the IEEE 802.15.4 ACK frames in software. All IEEE 802.15.4 control frames including ACK frames contain a 2-octet Frame Control Field (FCF), where 3 bits in the FCF are currently unused (Reserved). We use these 3 bits to provide feedback about the reception quality to the sender. Link quality is typically measured using the Receiver Signal Strength Indicator (RSSI) or the Link Quality Indicator (LQI). We focus on RSSI, although the results should likewise apply to LQI as well. The mapping between actual RSSI value and the 3 bits is application-dependant and out of scope for this paper.

As there are some strict timing constraints that the sending of ACKs have to adhere to (see III-A), implementing software acknowledgements has to be done in an efficient way. We implemented code that will already start reading the received frame from the CC2420's RXFIFO, while it is still being received. After having read the first four bytes of the incoming packet, the ACK frame is constructed and placed in the TXFIFO of the CC2420. During the time the ACK is constructed and written to the TXFIFO, more data arrives in the RXFIFO. Since we can can read the RXFIFO faster than it is filled by receiving an IEEE 802.15.4 frame, we can make up for the time needed to construct the ACK frame while reading the remainder of the frame.

Upon reception of the last byte and after completion of the copy procedure from the RXFIFO, the prepared ACK will be send or flushed, depending on the outcome of the FCS check.

A significant advantage of this approach is that the packet is completely read out of the buffer shortly after the frame has been received. This can be beneficial for time-critical applications and may also increase throughput. Normally it takes a significant amount of time after the full frame is received by the radio to copy it out of the RXFIFO.

A. IEEE 802.15.4 Timing Constraints

According to the standard IEEE 802.15.4 [8], a sender should wait for up to *macAckWaitDuration* symbols for acknowledgement frames after the original data frame has been transmitted. This period has to be less or equal to *macAckWaitDuration* in order to comply with IEEE 802.15.4. The *macAckWaitDuration* already includes the time for the ACK frame itself (Table 86, [8]). The value is calculated as follows ([8], Section 7.4.2):

$$\begin{aligned} macAckWaitDuration = {} & aUnitBackoffPeriod + \\ & aTurnaroundTime + \\ & phySHRDuration + \\ & \lceil 6 \cdot phySymbolsPerOctet \rceil \end{aligned}$$

For the 2.4 GHz DSSS PHY *macAckWaitDuration* is as follows:

$$macAckWaitDuration = 20 + 12 + 10 + \lceil 6 \cdot 2 \rceil = 54 \quad (1)$$

$$54\ symbols \cdot 62.5 \cdot 10^{-3} s/symbol = 864\mu s \quad (2)$$

For our discussion and evaluation in the remainder of this paper, we refer to the actual time between transmission of the last byte of a data frame until the reception of the last byte of the ACK frame as acknowledgement time or ACK time.

According to [8] the transmission of an ACK frame "shall commence aTurnaroundTime symbols after the reception of the last symbol of the data or MAC command frame". *aTurnaroundTime* is the maximal time allowed for a CC2420 compliant transceiver to switch its radio from TX to RX mode. Commencing to send the ACK earliest after *aTurnaroundTime* makes sure that all compliant transceiver have already switched to RX mode and are able to receive it. For the 2.4 Ghz PHY *aTurnaroundTime* is:

$$aTurnaroundTime = 12\ symbols \quad (3)$$

$$12\ symbols \cdot 62.5 \cdot 10^{-}3 s/symbol = 192\mu s \quad (4)$$

Sending an ACK earlier is harmful, sending it later is acceptable if it can still be received within the limit set by *macAckWaitDuration*. In order to be standard compliant, an implementation should begin sending the ACK frame at latest $864\mu s - 192\mu s = 672\mu s$ after it has received the last byte of the data frame. We have measured the timing of the acknowledgement process using the CC2420 transceiver with hardware acknowledgements enabled as shown in Figure 1. As can be seen the CC2420 begins transmitting the ACK $360\mu s$ after receiving the last symbol of the data frame. An 802.15.4 ACK frame consists of 6 bytes, which leads to a transmission time of $192\mu s$ using the 2.4 GHz PHY at 250 kbit/s. As reference the figure also includes the *aTurnaroundTime* and the *macAckWaitDuration*. For the hardware-generated acknowledgements the measured times are not dependent on the packet size. All experiments have been performed with the IEEE 802.15.4 security features turned off. While the IEEE 802.15.4 security extensions do not modify the timing requirements of the standard, the timing observed during measurements with security enabled might differ from the timing observed in this paper due to additional processing overhead.

B. CC2420 Interface

Figure 2 shows how the CC2420 on the T-Mote Sky interfaces with the host controller. The CC2420 uses an SPI interface to communicate with the application controller which is driven by the MSP430 controller on the T-Mote Sky with 3.9 MHz. In addition to the SPI bus the CC2420 radio chip uses various digital IO pins to provide information about its current state. In receiving mode, the SFD pin goes high after the Start of Frame

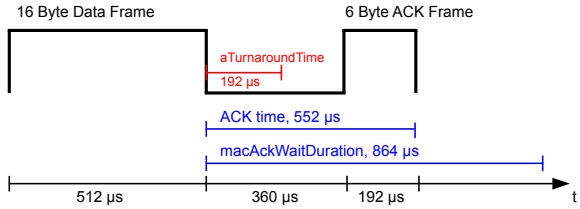

Figure 1: CC2420 Timing using hardware acknowledgements

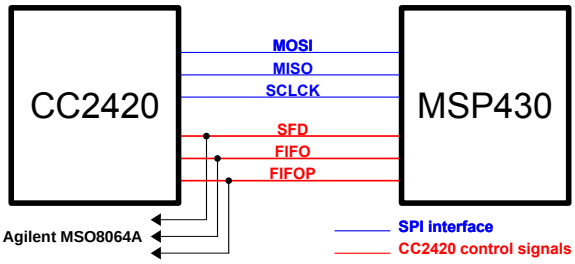

Figure 2: CC2420 interface

Delimiter (SFD) has been received and goes low after address recognition has failed or after the last byte of the frame has been received. The FIFO pin goes high when there is at least one byte waiting in the RXFIFO and goes low when the RXFIFO is empty again. The FIFOP pin is high whenever more than a configurable number of bytes are waiting in the RXFIFO and the CC2420's integrated address recognition was successful or it goes high after a complete frame has been received. The threshold for the FIFOP pin is configurable between 0 (go high immediately after address recognition) and 127 (only indicate completely received frames). More information can be found in the CC2420 data sheet [7].

IV. Implementation

Our implementation aims at being a drop-in replacement for the existing Contiki device driver and consequently uses the same structure. However, instead of waiting for the complete frame to be received by the radio, we read the frame out of the RXFIFO in blocks while it is still being received. After reading the first four bytes, the ACK frame can be prepared and copied into the TXFIFO (see section III).

In order to provide link quality feedback to the sender, this information has to be added to the ACK frame. The CC2420 provides Receiver Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI) information in the last two bytes of the frame, replacing the FCS. However, waiting for these last bytes to create the ACK frame is too time consuming to send out the ACK in time. Therefore we opted for obtaining RSSI information from the CC2420 using the RSSI_VAL register that contains the RSSI averaged over the last 8 received symbols. We obtain this value during the creation of the ACK frame and incorporate it into the three unused bits in the FCF as mentioned in Section III.

Although SPI interfaces operate only byte-wise and do not have the notion of blocks, block-wise reading is still an advantage since the processing overhead on the microcontroller can be reduced by reading blocks and returning them to the driver. Otherwise the overhead from function calls costs too much time and reduces efficiency. However, using a larger block size also increases the amount of data that is possibly left in the RXFIFO when the frame is completely received. While data being read during the packet receive process does not contribute to the ACK time, reading data still remaining in the RXFIFO after the reception of the frame has been completed, will increase the time to send an ACK. The trade-off here is between processing overhead on the microcontroller and the time necessary for reading residual bytes after the complete frame has been received.

We configure the CC2420 to use hardware address recognition and disable the CC2420 AutoAck feature. To realize the block-wise reading of the incoming packet, the FIFOP threshold is set to a preconfigured block size (see also section V-B). After the first bytes of a frame have been received, the FIFOP interrupt fires and the interrupt service routine then sends a signal to the driver process which will eventually be polled by the scheduler. The process starts by reading the first 4 bytes including the length field, the FCF and the DSN. With this information, the ACK frame can be created and is subsequently copied into the TXFIFO of the CC2420. The read process now goes into the reading loop until all bytes of the frame have been read (or the CC2420 signals an error).

The reading loop waits for the FIFOP pin to go up and then reads a complete block from the RXFIFO. If less bytes than the block size are left in the RXFIFO, the driver resorts to byte-wise reading of the residual data. Once all bytes from the RXFIFO have been read out, the driver can verify the CRC checksum. The CC2420 radio replaces the FCS with a flag indicating whether the CRC checksum was successfully verified, so the actual calculation of the CRC does not have to be done in software. Based on the CRC flag, the driver decides to either send the prepared ACK frame or to flush the TX buffer.

V. Evaluation

In order to evaluate the usability of software-generated acknowledgements in a WSN deployment, we look at 4 points:

1) The influence of reading block sizes onto the timing
2) The influence of packet sizes onto the timing
3) Number of unrecognised acknowledgement frames
4) Energy consumption

A. Experimental Setup

The general setup for all the following measurements are two T-Mote Sky sensor nodes in close vicinity (distance $<$ 1m). One node is configured to send 60 unicast packets per second to the second node using

Contikis Rime [1] stack. The second node acknowledges these packets either using the CC2420 AutoAck feature or using our software-generated acknowledgment frames. The Contiki stack was configured to use the CSMA MAC layer with retransmissions enabled and both nodes were rebooted prior to each measurement run. Unless otherwise noted the following measurements were conducted in a university lab with active IEEE 802.11 hardware in the surroundings; possible influences can neither be excluded nor quantified. For the purpose of this measurement we have soldered wires onto the SFD, the FIFO and the FIFOP pin of the CC2420 and connected them to digital inputs on an Agilent MSO8064A oscilloscope as shown in Figure 2. In our evaluation on the receiving node we monitored the SFD pin to get accurate timing data for received and sent frames.

The time between the end of the send process of the original frame and the end of the receive process of the ACK frame at the sender node can be measured by looking at the SFD pin. In receive mode it goes high after the Start of Frame Delimiter has been received and goes low again after the complete frame has been received or address recognition failed. In transmit mode it goes high after the SFD has been transmitted and goes low again after transmission of the frame is completed. We have measured the time between the edges of the SFD signal using the configurable trigger functionalities of the Agilent MSO8064A. The integrated measurement functions have been used to gather mean, max and min values and write them to a file. As the oscilloscope has a finite buffer and calculates mean, max and min only over the contents currently residing in the buffer, depending on the packet size, the following results are based on 200-600 ACK frames being measured.

Packet loss has been measured in software. The sender node counts the number of outgoing packets and the number of incoming ACKs while the receiver node counts the number of incoming packets. We can then calculate the number of lost or unrecognised ACK frames using $Packets_{sent} - Packets_{Received}$ while excluding the number of lost data frames.

B. Block Sizes

Our first concern was the optimal block size for reading information from the radio via SPI.

Figure 3 shows the block size in bytes on the x-axis and the acknowledgement time in μs on the y-axis. The three data lines indicate radio packet sizes of 18, 66 and 128 bytes including the FCS. The maximum permissible time *macAckWaitDuration* is marked in red. The measurement set up is described in Section V-A. As expected, the acknowledgement time for a block size of 1 is significantly higher than *macAckWaitDuration* for all packet sizes due to processing overhead. Smaller packet sizes generally have higher acknowledgement time than larger packets which can be explained by the fact that reading data over SPI is faster than receiving data over IEEE 802.15.4. The time that is necessary for address recognition and the overhead of Contiki are independent of the packet size, so that the driver can catch up for

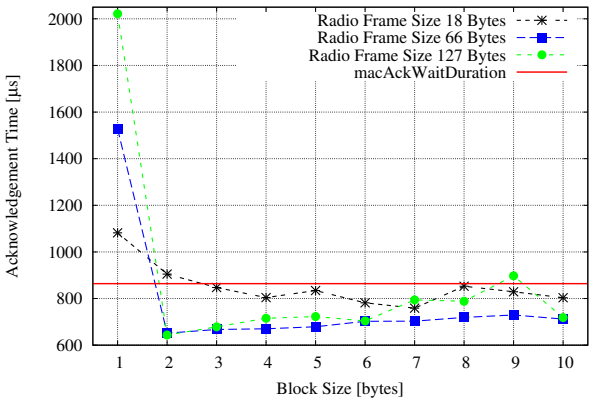

Figure 3: Acknowledgement time for different block sizes with software generated ACKs.

larger packets while there is not much potential for catching up for smaller packet sizes. Interestingly, larger packets have the lowest ACK time for a block size of 2 since the overhead is low and a maximum of 2 bytes can wait in the buffer once the receive process is finished. Small packets have lower ACK times for larger blocks which can also be explained by the driver catching up during the SPI transfer. By taking the arithmetic mean of the ACK times for the three packet sizes, block sizes of 4 and 6 bytes have the lowest ACK time with the average for 6 bytes being marginally lower. Since smaller radio frames experience a higher ACK time than larger radio frames, especially for small block sizes, we think that an SPI read block size of 6 byte is the best trade-off between overhead and ACK time.

C. Radio frame sizes

To evaluate whether software-generated acknowledgements meet the time constraints of IEEE 802.15.4 (see section III-A), we have measured the acknowledgement time for different radio packet sizes. Also, we have measured the respective times for hardware-generated acknowledgements to have a comparison baseline. We have used the Contiki Rime stack to generate radio frames of 16 - 126 bytes plus 2 bytes FCS.

Figure 4 shows the radio packet sizes in bytes on the x-axis and the acknowledgement time in μs on the y-axis. The boxes plot the minimum, the arithmetic mean and the maximum while the whiskers depict the standard deviation; the maximum permissible ACK time is marked by the red line. The measurement set up is described in Section V-A and these measurements use a block size of 6 bytes. As previously seen, the software ack time for 18 byte frames is higher than the ACK time for larger packet sizes. In general, the software ACKs are relatively stable with some small glitches and a marginal standard deviation. The hardware generated acknowledgements show no variation or dependence on the frame size whatsoever and are perfectly stable. As expected, software generated ACKs are between 140 μs and 225 μs slower than the hardware generated pendant

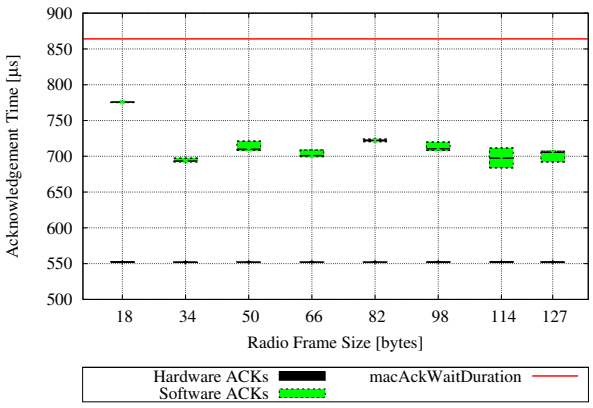

Figure 4: ACK times for hard- and software generated ACKs

Packet size	18 bytes	34 bytes	66 bytes	128 bytes
HW ACKs	0.00 %	0.00 %	0.00 %	0.00 %
SW ACKs	0.08 %	0.08 %	0.08 %	0.01 %

(a) Lost Acknowledgement Frames

Power Cons.	Avg.	Run 1	Run 2
HW ACKs	0.7450 mWh	0.7322 mWh	0.7578 mWh
SW ACKs	0.7231 mWh	0.6912 mWh	0.7550 mWh

(b) Power Consumption

Table I: Comparison between software and hardware generated acknowledgements using a contention-based MAC

but still lie within the time bounds.

D. ACK detection rate

Software-generated ACKs are likely more fragile than their hardware-generated pendants since they rely on Contikis cooperative multitasking. Consequently, we wanted to investigate how many acknowledgement frames are actually being recognised by the sender. In this test we used the Contiki Rime stack to generate 10000 packets on the sender side. We have performed runs with varying radio packet sizes. The results show the arithmetic mean of 3 runs per size. The measurement was performed in the basement of an office building with active IEEE 802.11 hardware. Influences of any kind cannot be excluded.

Table Ia shows the ratio of unrecognised or lost acknowledgements in percent for different radio packet sizes using hardware and software generated acknowledgements. The ratio only counts data packets which were successfully sent by the sender and received by the receiver but for which the sender did not receive an ACK in time. The table shows that generally not a single hardware-generated ACK frame was lost or not recognised while between 0.01 and 0.08 % of the software-generated acknowledgement frames were lost. We cannot say, what caused the loss of ACKs in these particular cases, but it is likely that timing is the problem. This thesis is supported by the fact that the largest packet size exhibits significantly lower ACK losses which could be caused by the radio driver catching up time that was previously lost.

	Avg.	Min.	Max.
Hardware ACKs	0.027 %	0.022 %	0.032 %
Software ACKs	0.060 %	0.052 %	0.063 %

Table II: Unicast Packet Loss using a TDMA MAC

E. Packet loss

The GINSENG project uses a TDMA-based Medium Access Control [6] based on IEEE 802.15.4 radios that is specifically optimised for industrial process automation and control. In-time delivery of messages and the overall delivery reliability are of major importance in such networks.

We have also evaluated our software-generated acknowledgement approach using the GINSENG TDMA MAC system in the basement of an office building with active IEEE 802.11 hardware. The testbed uses 15 nodes and a sink node and the following results are accumulated values from all nodes in the network. We have conducted 5 measurement runs of one hour each, separately for hardware- and software-generated ACKs with an average of 132506 unicast frames being generated during each run. Our monitoring system counts a packet as lost when no ACK frame is received in time but we did not have any means to distinguish the loss of a data from the loss of the respective ACK frame.

Table II shows the average, minimum and maximum packet loss during all 10 measurement runs. The number of lost packets when using hardware ACKs averages at 0.027 % and can be seen as a baseline that may be caused by interference with other transmitters or by TDMA timing problems. Software-generated ACKs introduce additional packet losses which average at 0.060 %. When we subtract the number of lost packets when using hardware ACKs from the number of lost packets for software ACKs, the loss rate averages at 0.033 % which is well within the range found in the previous measurement.

F. Power consumption

To see whether the power consumption of software-generated ACKs is different from the hardware-generated pendant, we measured power consumption for both cases. We have conducted two runs with 1000 packets of 34 bytes each that were transferred from sender to receiver. We have measured the power consumption of the receiver by monitoring a step-up converter as explained in [4] that was inserted into the USB power supply lines of the T-Mote Sky. The measurement was started after the first packet was sent and stopped after the second to last packet was sent.

Table Ib shows the average power consumption as well as the individual results for the two runs. Considering the accuracy of the measurement method, no real difference in the power consumption between hardware- and software-generated acknowledgements can be seen.

Nevertheless, we have observed that Contiki's software-based online energy estimation [2] reports approximately 14 % increased transmit energy consumption when using our approach. This is caused by the fact that by default, the estimation mechanism only counts frames that are sent in software. Since ACKs in Contiki are generated in hardware, the energy estimation does not "see" them. Consequently, our approach enables more accurate energy accounting compared to hardware acknowledgements when using Contiki's built-in energy estimation.

G. Limitations

A severe limitation of the block-wise approach lies in the way an incoming frame is signalled to the driver process. An incoming interrupt is converted into a Protothreads interprocess signal [3] and the frame is eventually read out of the RXFIFO (and finally acknowledged) whenever the operating system scheduler decides to schedule the driver process. Since Contiki uses cooperative multitasking strategies, one cannot guarantee that ACK frames are always generated in-time. This depends on the overall work load of the sensor node and can be avoided by creating the ACK frame directly in the interrupt service routine. However, this would involve changing the design of the radio driver and was not considered in this paper. Also in order to achieve fast ACK respone times, the current reading routine basically performs busy-waiting while receiving, which will limit the the computational resources available to other Contiki tasks.

VI. Conclusions and Future Work

In this paper we have presented an approach to generate IEEE 802.15.4 acknowledgement frames in software on the T-Mote Sky sensor nodes while complying with the timing constraints set by the IEEE 802.15.4 standard. By reading the packet from the RXFIFO while it is still being received and by using a block-wise reading technique we were able to lower the processing overhead to send ACKs in time. By obtaining link quality information during the reception of a frame and not waiting for the packet footer, our approach is able to lower the critical time until the ACK is transmitted even further.

In the evaluation we have found a block size of 6 bytes to be the best trade-off between processing overhead and ACK time. We have also evaluated the ACK time for different packet sizes and found that software-generated acknowledgements stay within the time bounds for all our measurement runs. However, we have found that software ACKs are not as reliable as hardware ACKs because up to 0.07 % of software generated ACKs are not recognised by the receiver. Finally, we have shown that software ACKs do not increase power consumption compared to hardware ACKs but improve the accuracy of Contiki's built-in online energy estimation mechanism.

Additional work has to go into more efficient ways of encoding link quality feedback information into the three available bits. Further analysis has to show why the software-generated ACKs have a slightly higher probability of being lost.

Acknowledgements

This work has been partially supported by the European Commission under the contract FP7-ICT-224282 (GINSENG) and by the NTH School for IT Ecosystems.

References

[1] Adam Dunkels. Rime — A Lightweight Layered Communication Stack for Sensor Networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, January 2007.

[2] Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th workshop on Embedded networked sensors*, EmNets '07, pages 28–32, New York, NY, USA, 2007. ACM.

[3] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 29–42, New York, NY, USA, 2006. ACM.

[4] Prabal Dutta, Mark Feldmeier, Joseph Paradiso, and David Culler. Energy Metering for Free: Augmenting Switching Regulators for Real-Time Monitoring. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 283–294, Washington, DC, USA, 2008. IEEE Computer Society.

[5] Shan Lin, Jingbin Zhang, Gang Zhou, Lin Gu, John A. Stankovic, and Tian He. ATPC: adaptive transmission power control for wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 223–236, New York, NY, USA, 2006. ACM.

[6] Petcharat Suriyachai, James Brown, and Utz Roedig. Time-critical data delivery in wireless sensor networks. In *6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '10)*. IEEE, June 2010.

[7] Texas Instruments Incorporated. 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. http://www.ti.com/lit/gpn/cc2420.

[8] The Institute of Electrical and Electronics Engineers, Inc. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). IEEE 802.15.4-2006, September 2006.

[9] TinyOS Wiki. CC2420 Hardware and Software Acks. http://docs.tinyos.net/index.php/CC2420_Hardware_and_Software_Acks.

[10] Shuo Xiao, A. Dhamdhere, V. Sivaraman, and A. Burdett. Transmission Power Control in Body Area Sensor Networks for Healthcare Monitoring. *Selected Areas in Communications, IEEE Journal on*, 27(1):37 –48, 2009.