An Autonomous Framework for Supporting Energy Efficiency and Communication Reliability in WSNs

Osama Khader  
Technische Universitaet Berlin  
Berlin, Germany  
thaker@tkn.tu-berlin.de

Andreas Willig  
University of Canterbury  
Christchurch, New Zealand  
andreas.willig@canterbury.ac.nz

Adam Wolisz  
Technische Universitt Berlin, and  
University of California, Berkeley  
awo@ieee.org

Abstract—We propose a novel decentralized and self-learning framework to support both communication reliability and energy-efficiency for periodic traffic applications in WSNs. Our autonomous framework comprises three main components: estimation and identification of multi-flow traffic, dynamic multi-flow wakeup scheduling, and asynchronous channel hopping. With asynchronous channel hopping the frequency hopping pattern is determined by each source node autonomously, and forwarders have to identify and follow the pattern. We also propose a light and efficient controller to eliminate the collision caused by multi-flow overlap at forwarders. We present design and evaluation of our autonomous framework using real world measurements, and realistic trace-based simulation. The results show that our asynchronous channel hopping solution improves the packet reception rate compared to the single channel solutions without the need of an expensive signaling and time synchronization overhead. We also show that with this scheme the average energy consumption yields a \( \approx 50\% \) lower than the single channel solutions. This paper is to the best our knowledge, the first to explore channel hopping without maintaining a tight time synchronization protocol.

I. INTRODUCTION

In many application areas of embedded wireless networks, for instance in building automation or industrial control, source nodes send data packets periodically to a gateway or sink node across a set of forwarder nodes [21], [22], [7]. For cost-effective, quick and scalable deployment, sensor nodes often run on batteries, and therefore have only a limited amount of energy. The sensed data should be transmitted reliably and in a timely fashion to the sink. At the same time the operation of the whole network and of individual nodes should be energy-efficient. Therefore, reporting the sensed data reliably while consuming the minimum amount of energy is of great concern.

One of the key approaches to achieve energy-saving is to let the forwarding nodes switch to an energy-conserving sleep state whenever possible. In this sleep state parts of the node hardware, especially the wireless transceiver, are switched off. This disables the communication ability of a node but leads to significant energy savings, since for most of the currently available sensor node platforms the wireless transceiver is the dominant source of energy consumption. The fraction of time where the node is awake is called its duty cycle, and from the perspective of energy-efficiency this duty cycle should be kept as small as possible. For a source node generating the periodic data there is no problem: the node wakes up, samples its sensor, transmits a packet and returns to sleep mode until time for next sampling has come. In a multi-hop network other nodes are needed to forward the packet to a sink node. To be most energy-efficient, a forwarder should wake-up just before a periodic packet arrives, do the necessary forwarding work and enter sleep mode again. However, in general the time differences between packet arrival times seen by a forwarder are not ideally regular, but have a random component, for example due to the usage of randomized MAC protocols, time-varying cross-traffic (resulting in queueing effects), or operating system imperfections (i.e. interrupts handling). The deviation from perfect periodicity is also referred to as jitter. Intuitively, one might expect that the amount of jitter is a function of the number of hops a packet traverses.

On the other hand, reliability in WSNs is challenged by multi-path fading and narrow band interference. Low communication reliability causes packets to be lost, and therefore retransmission of lost packets is usually needed, which in turns leads to higher energy-consumption. One important approach to improve reliability is to exploit frequency diversity by channel hopping, i.e. periodically changing the communication channel. Channel hopping is known to substantially improve communication reliability in WSNs [3], and therefore it has been adopted in recent standards for industrial wireless sensor networks, for example Wireless-HART and ISA100.11a, see [17], [11], [13] [1]. Both Wireless-HART and ISA100.11a rest on a TDMA approach with slow-hopping, i.e. slot-by-slot frequency hopping. These protocols use an explicit time synchronization protocol in order to be able to switch between different channels and communicate. They also require to have a centralized coordinator and extensive signaling overheads. Moreover, because these protocols require communication schedules to be computed and distributed in advance, it is relatively expensive (energy-wise) to adapt the network to new topologies or load situations.

The key motivation for this paper is the observation that the full TDMA operation including time synchronization, maintenance and schedule dissemination represents too much overhead for lightly loaded networks. Nonetheless we want to support periodic transmissions and we want to leverage frequency hopping. In order to address these challenges, we have developed a distributed and self-learning framework integrating asynchronous channel hopping, estimation of periods and dynamic multi-flow wakeup scheduling. Two key ideas are used. First, there is no explicit time synchronization, but instead each forwarder learns the traffic period and jitter distribution from observing the traffic in distributed manner. Based on this information a forwarder determines suitable times for
sleeping and for waking up to receive the next packet—this approach has been introduced in an earlier publication of ours (see [15]), however in this paper we extend the work to multi-flow and multi-channel scenarios. Secondly—this is the main novel contribution of this paper—the source nodes and all forwarders switch channels for each new periodic packet, and source nodes are independent of each other, i.e. they choose their own transmission periods and channels autonomously. We assume that the physical layer offers a number of different orthogonal frequency channels—the prime example (being adopted in this paper) are transceivers following the IEEE 802.15.4 physical layer in the 2.4 GHz ISM band. A forwarder thus uses the estimated traffic periods also for figuring out the times when it needs to switch the channel. In order to integrate these approaches, some significant challenges have to be addressed. First, enabling nodes to switch between different channels without maintaining a time synchronization protocol is difficult, and to the best of our knowledge this has not been addressed in the WSN literature so far. Secondly, period estimation and the scheduling of wakeup times will have to deal with jitter in the packet inter-arrival times. If a packet arrives before the forwarder wakes up or after it has returned to sleep, it is lost. This opens up a trade-off between loss rates and the sleeping activities of the forwarder: when the forwarder wakes up “early”, the packet loss rate will be low but the forwarder spends more energy, and vice versa.

Thirdly, certain forwarders might be placed on the routes for several distinct sources and must adapt both its sleep/wakeup windows and also the frequency, especially in situations where packets of different source flows “collide” at a forwarder. To the best of our knowledge, this paper is the first attempt of proposing a distributed and self-learning asynchronous multi-channel hopping for supporting both energy-efficiency and communication reliability.

The remainder of the paper is structured as follows. Section II presents an overview of the autonomous framework and its wake-up times scheduling approach. It also presents experimental jitter measurements and results. The asynchronous channel hopping mechanism is presented in Section III. In this section we also introduce the multi-flow overlapping mechanism. The methodology, performance metrics, and experimental setting are explained in Section IV. Section V presents the result of the autonomous framework. Related work is discussed in Section VI and finally, Section VII concludes the paper with some future works.

II. DISTRIBUTED WAKEUP SCHEDULING SCHEME

In this section we briefly review the distributed wakeup scheduling scheme for supporting periodic traffic in wireless sensor networks that we have introduced in [15]. In this paper the distributed wakeup scheduling is extended to support multi-flow traffic. Readers may refer to our technical report [16] for more details.

A. Distributed Wakeup Scheduling Scheme

We consider a multi-hop wireless sensor network in some source nodes generate periodic traffic, which needs to be carried to a sink node. The network also comprises forwarder nodes which help with forwarding the traffic from the sources to the sink. Each source wakes up periodically, samples the environment through its local sensors, transmits a packet carrying the sensor data to the next forwarder on its path to the sink, and goes back to sleep. Each source node can have its own period and there is no common time reference in the network, nor is there any other synchronization mechanism present. We only require that sources have unique identifiers which they include in their packets.

A forwarder has the goal to stay most of its time in a deep sleep state. Since the source traffic is periodic, a forwarder can expect to see traffic that is approximately periodic, but with non-negligible jitter. The jitter can for example result from the operation of the underlying MAC protocol (e.g. when random backoff times are used like in the IEEE 802.15.4 CSMA MAC) or from cross-traffic and queueing effects in upstream nodes. Hence, the interarrival time distribution seen by a forwarder is not impulse-like, but instead has some variability around its average (which is the source period). When the forwarder knows the interarrival time distribution, it can obtain the period as the average of this distribution. Furthermore, knowing the lower \( \alpha/2 \) and upper \( \alpha/2 \) quantiles (for \( 0 < \alpha < 1 \)), the forwarder can schedule its sleeping activities such that it wakes up at the lower \( \alpha/2 \) quantile, waits for a packet to forward, forwards it (if any), and goes back to sleep at the upper \( \alpha/2 \) quantile. Following this approach guarantees that no more than a fraction of \( \alpha \) packets are lost from the forwarders sleeping activities.

Clearly, in the absence of any synchronization and signaling, the forwarder does not know the interarrival time distribution a priori. Therefore, a forwarder alternates between two different states: In the learning state a forwarder is switched on all the time and observes all packets from a source. After a number of observations the forwarder is able to estimate the period and the relevant quantiles. Once these estimates are reliable enough, the forwarder enters the other state, called the operational state. In the operational state the forwarder follows the sleep/wakeup cycle sketched above, where it wakes up at the lower \( \alpha/2 \) quantile and returns to sleep at the upper \( \alpha/2 \) quantile. Furthermore, the forwarder observes the packet loss rate in the operational state and continues to update the estimates of the period and the quantiles (we refer to this as statistics update). If the packet loss rate grows too large, the forwarder returns to the learning state in order to re-estimate period and quantiles. This allows forwarders to adapt to changes in topology or load scenario. The duration of the learning state and the precise consecutive packet loss rate threshold triggering the transition back from the operational into the learning state are design parameters of the scheme.

What we have described so far does not consider frequency hopping, which we explain in detail in Section III. For the time being, it suffices to mention that a forwarder in the learning state initially starts listening on a randomly chosen channel. Having received the first packet on this channel, the forwarder knows on which channel the next packet from the same source will be transmitted, as the source switches channel for each new packet and the channel hopping sequence is well known to all nodes.

B. Estimating the Jitter Distribution

To implement the previously described approach, a forwarder needs the ability to estimate the jitter distribution and
derive both its average (giving the estimated period) and the upper and lower $\alpha/2$ quantiles. In general, the jitter distribution will depend on the choice of underlying protocols (e.g., CSMA vs. ALOHA vs. TDMA), the presence or absence of retransmissions, and also on the hop distance of the forwarder from the source, as the jitter can be expected to accumulate with increasing hop number.

In the following we report the results of jitter measurements carried out with the standard protocol stack coming with the TinyOS 2.0 operating system. We provide evidence that for this protocol stack the jitter distribution at various hop distances is well modeled by a normal distribution. With this insight the estimation process is simplified: a forwarder only needs to measure the average and standard deviation of the packet interarrival times in order to obtain the full distribution. From this distribution then the desired upper and lower $\alpha/2$ quantiles are derived (see our technical report [16]).

1) Experimental Setup: We carry out our experiments on the TWIST testbed (TKN Wireless Sensor Networks Testbed) [9]. It has approximately 102 Tmote sky nodes spread over three floors of our FT building at the TU Berlin campus. Each mote is integrated with the popular IEEE 802.15.4-compliant ChipCon CC2420 radio transceiver [2] operating in the 2.4 GHz ISM band and has a data rate of 256kbps. The transceiver supports 16 channels (from 11 to 26) in the 2.4 GHz band, with a center frequency separation of 5 MHz for adjacent channels. There are some obstacles in the TWIST testbed area that could impede RF communication and cause multi-path reflections. In addition, the building is occupied with some WiFi access points which may introduce external interference to the TWIST network. We use the TinyOS version 2.0 operating system [12], [6] and its default protocol stack, which uses a CSMA-type MAC protocol. We also use the well known CTP routing protocol [8] to construct routes between the sources and the sink. For the measurements each sensor samples the temperature sensor periodically. The generation period was varied, ranging from 1 to 30 to 60 seconds. Furthermore, for each of the 16 channels a separate set of measurements has been carried out. During one experiment, each source transmits 5000 packets to the sink node via a set of forwarders on its respective path. MAC-layer acknowledgements are enabled and a maximum of two MAC-layer retransmissions are carried out. Each forwarder records the timestamps of the received packets, source and destination addresses, flow ID, and other parameters. The packet size is set to 80 bytes (not including packet overheads). The number of sources is set to 10 and we allow for each source and forwarder to have up to five neighbours and communicate on all of the available channels (16 channels). The minimum, average and maximum number of hops from any source to the sink node was, 3, 5.6 and 8, respectively.

2) Discussion and Result for Jitter Measurements: For all the scenarios covered in our measurements we find that the per-flow jitter distribution is well modelled by a normal distribution. For brevity we only show the histograms and the quantile-quantile plots for channel 11 as shown in Figures 1 and 2, respectively. Please note that this finding was also confirmed in our previous paper [15], which was limited to a single source scenario with a common single channel. However, here we have extended our measurements to multi-flow and multi-channels scenarios. Please note that similar trends are observed also for scenarios with 30 and 60 seconds traffic generation period and for all other channels.

III. ASYNCHRONOUS CHANNEL HOPPING

Our asynchronous channel hopping approach is not synchronized to any external time reference. Instead, channel hopping is synchronized to the period with which a source node sends its data packets. Our approach distinguishes itself from existing channel hopping protocols, such as WirelessHART [4] and ISA100.11a [13], by scheduling the whole network activities in a distributed manner and without maintaining an explicit time synchronization protocol, thus reducing the signaling load and saving overall system complexity. We first explain our approach for a single flow and then extend to the case with multiple flows in a network.

There are two main approaches to channel hopping: (synchronized) blind channel hopping and adaptive channel hopping. Blind channel hopping (as used in WirelessHART) might use all 16 channels independent of their current quality and hops on a per-time-slot basis (which in WirelessHART amounts to a per-packet basis). In contrast, adaptive hopping aims to use a subset of the best channels (white-listing). Adaptive hopping is more complex to implement, as it first requires a mechanism to frequently scan and rank all the channels for their quality for each link. Second, each node has to keep
ABCH exploits the characteristics of a single periodic traffic flow and estimates the next channel to be used based on the sequence number of the packet. Each source node starts hopping blindly on a per-packet basis, using available channels. The source includes sequence numbers into its packets, and the next channel to use depends on the sequence number as follows:

$$\text{NextChannel} = (SQ + \text{chOffset}) \mod \text{chNum} \quad (1)$$

where $SQ$ is the next sequence number, $\text{chOffset}$ is the channel offset and $\text{chNum}$ is the number of channels being used. In the learning phase, each forwarder starts by listening for a packet on some random channel. Upon receiving the first packet on this channel, the forwarder retrieves the sequence number and determines the next channel according to Equation 1. As an example, if $\text{chOffset} = 1$, $\text{chNum} = 16$ and a forwarder received packet with $SQ = 8$, then the current channel index is 9 and the next one is 10 (Note that here the channels are numbered from 0 to 15 instead of 11 to 16, but translation is straightforward). Please note that each forwarder applies the ABCH mechanism after receiving the first packet – specifically, a forwarder also uses the determined channels for its own transmissions of the packet. In the next section we examine the synchronization of two neighbors in the presence of transmission errors.

### A. Handling Transmission Errors

Figure 3 shows the interaction between a pair of nodes for exchanging packets. We assume that the nodes have learned the flow period and are ready to communicate. Figure 3 illustrates three sequences, the first sequence shows a simple error-free transmission. In this sequence a sender transmits packet $p_1$ on channel 11 and waits for an ACK for a predefined timeout on the same channel. Upon reception of the packet the receiver sends an ACK back to the sender indicating the next expected sequence number to be received and performs a statistics update. If the receiver receives the ACK, it also performs a statistics update and removes that packet from its buffer, otherwise a copy of the transmitted packet is kept in the buffer.

The second sequence illustrates the interaction in case of a data packet loss. When the receiver wakes up on channel 12 to receive a packet, it waits for its wakeup window and remains awake until either a packet is received or until the upper $\alpha/2$ quantile has passed, as explained in Section II-A. In this example the receiver does not get a packet and assumes that the packet is lost and updates its statistics. However, it computes the next channel frequency as if it received packet $p_2$ (the lost packet). This is important as we will explain in the third sequence (ACK loss). Similar actions are taken at the sender side. Once the ACK time-out is triggered, the sender assumes that packet $p_2$ or its associated ACK is lost. However, it computes the next channel as if it received a successful ACK for packet $p_2$ and then updates its statistics. In the next wakeup-window it transmits packet $p_5$ on channel 13. Upon receiving $p_5$, the receiver node sends back an ACK indicating the next expected packet to receive. In this case the receiver returns the sequence number of packet $p_2$ and it stays awake\(^1\) to receive packet $p_2$ on the same channel (channel 13). The sender then retransmits packet $p_2$ on channel 13. Please note that, the recovery process of the lost packet $p_2$ immediately follows the previous successful transmission of $p_5$, leveraging the good conditions on the current channel.

The third sequence shows the packet exchanges in case of ACK packet loss. When the sender transmits $p_5$ on channel 15 and its timer expires before the packet is acknowledged, it assumes that either the data packet or the ACK packet is lost. In either case, it computes the next channel as if it got a successful ACK for $p_5$. It also updates its statistics and goes to sleep. In the next wakeup window, the sender transmits $p_5$ on channel 16 and waits for an ACK. If it receives a successfully ACK then it knows that the previous packet ($p_5$) was correctly received, because the receiver indicates in its ACK that the next expected sequence number is that of $p_5$, otherwise the ACK would have included the sequence number of the lost packet. Please note that our scheme prevents duplicate packets caused by ACK packet loss, thus more energy-efficient.

### B. Multi-flow Overlapping Mechanism

We finally consider the operation of our scheme in the multi-flow case, focusing on a forwarder through which two or more flows of possibly different period and from different sources pass. For such a forwarder it might happen that two upstream nodes want to send packets at nearly the same time but possibly on different frequencies -- we refer to this as a collision. To deal with this, we propose to exploit the traffic estimation values to detect and resolve a potential collision beforehand. Specifically, after receiving a packet a forwarder

---

\(^1\)Please note, that in this circumstance, the forwarder increases its wakeup-window temporary to accounts for the retransmission. Upon receiving the missing packet, the forwarder may go to sleep (depends on how much time left for the next activity).
checks all flows going through it whether there is a collision for the next packet. If so, it notifies the upstream node (by setting a special flag in the ACK packet) to randomly back-off longer in the next transmission cycle. Also the forwarder readjusts its corresponding wakeup window temporarily. We assume that there is no overlap in the first cycle of the transmissions. This is a realistic assumption since each node usually starts with a random offset.

IV. METHODOLOGY AND EXPERIMENTAL SETTING

Selecting the right methodology for evaluating the above scheme is not trivial. On one hand, theoretical channel models usually do not capture complex phenomena such as multi-path fading, or the impact of a dynamic environment. On the other hand, real-world experiments do not provide the ability to evaluate different schemes or algorithms under the exact same condition, as the RF environment is time-varying. We decided to combine these two methods and to use connectivity traces gathered from a real world deployment as an input to the simulation. We first discuss the traces, then describe the simulation setup.

A. Connectivity Traces

We evaluate our approach using both a single channel and multiple channels using connectivity traces gathered in a real world deployment. The connectivity traces have been collected by the DUST networks group [3]. The measurements were conducted in a printing factory in Berkeley, California, and data was collected over 26 days. The building has a rectangular footprint, measuring 250 feet x 225 feet. There are many obstacles in the work area that could impede RF communication and cause multi-path reflections. Overall, 45 sensor nodes were deployed with a relatively uniform distribution. Each node is equipped with a ChipCon CC2420 radio chip [2]. For this experiment, the data consisted of periodic reports of the quality of the communication path, where a path represents all transmission between a pair of nodes. The nodes had up to five neighbours and communicated on all of the 16 channels supported by IEEE802.15.4. The trace contains all path-channel reports. For more details about the setup of the experiment please refer to [3].

B. Simulation Setup

In order to realize a simulation model to study the performance of autonomous framework over a wireless multihop network, we have chosen connectivity traces, and the well-known OMNeT++ [25] simulation environment together with the Castalia WSNs framework [18]. OMNeT++ is an open-source discrete-event simulator, Castalia is an OMNeT++ based framework. We have modified the radio and channel models of the simulator to support not only the gathered traces but also the 16 channels of the IEEE 802.15.4 standard. We have also modelled the time required for channel switching. Table I summarizes the main power consumption parameters of a CC2420 transceiver and of a MSP430 microcontroller, assuming a 3.3V supply voltage. The main parameters of the autonomous framework is also listed in the same table (see our technical report [16] for more details of the parameters).

For the channel error model we use (unless otherwise specified) the traces introduced above. Specifically, for each link and each channel we change every 15 (simulated) minutes the packet delivery ratio by reading the next value for the packet delivery ratio for this link and channel from the trace files.

C. Network Topology and Traffic

We have generated 150 random topologies and for each setting of simulation parameters we correspondingly perform 150 replications. For each random topology we have placed 45 nodes in an area of size $225 \times 225$ feet, using a uniform distribution for node positions. The sink node is placed in the upper right corner of the nodes. Out of the 45 nodes we randomly pick five nodes as source nodes. Each of these sources periodically generates packets with a payload of 80 bytes (not including PHY and MAC overhead ). Unless otherwise specified, all the sources transmit with the same period, however, the starting phase is set randomly. The generation period was varied, ranging from 1 to 30 to 60 seconds. During each simulation run, each source transmits packets based on its periodicity and then forwards these packets to the sink node via some forwards. MAC-layer acknowledgements are enabled and the size of the ACK packet is 12 bytes. If the packet is lost then the sender tries to transmit the packet for a maximum of two retries.

D. Major Performance Measure

The simulation time is fixed to 168 hours (one week) and the two main performance measure are the total energy spent by the radio transceiver of a node over this period, and the end-to-end packet delivery ratio (PDR), i.e. the fraction of all packets sent by the sources that reach the destination. At one or two occasions we also use the end-to-end packet loss rate, which is just the complement of the packet delivery ratio, as a performance measure.

The simulation records the amount of time spent in various states (transmit, receive, listen, sleep and turnover) and calculates from this the total energy consumption of a node over a span of 168 simulated hours. We also take into consideration the energy consumed by the nodes microcontroller. We split the microcontroller energy consumption in two main states, active state and sleep state. The microcontroller is active at the same time as the radio. At the end of each run, the simulation computes the total energy consumed for all nodes in the network using the amount of energy consumed by the radio and microcontroller in each state.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit power (0dBm)</td>
<td>57.42 mW</td>
</tr>
<tr>
<td>Receive power</td>
<td>62.04 mW</td>
</tr>
<tr>
<td>Listen power</td>
<td>62.04 mW</td>
</tr>
<tr>
<td>Sleep-mode-1 power</td>
<td>1.406 mW</td>
</tr>
<tr>
<td>CPU active power</td>
<td>6 mW</td>
</tr>
<tr>
<td>CPU sleep power</td>
<td>0.148 mW</td>
</tr>
<tr>
<td>Channel switching time</td>
<td>192 µs</td>
</tr>
<tr>
<td>Length of learning phase</td>
<td>5 packets</td>
</tr>
<tr>
<td>Allowable packet loss rate α</td>
<td>2</td>
</tr>
<tr>
<td>Loss threshold</td>
<td>3 packets</td>
</tr>
</tbody>
</table>
V. RESULTS

In order to study the performance of our autonomous framework, we compare the asynchronous blind channel hopping with 16 channels against a system in which only one channel is used. We first investigate the packet delivery ratio and the total energy-consumption. Then we study the impact of the multi-flow overlap on energy-consumption and packet delivery ratio. We also investigate the impact of the length of the learning phase on the performance of the autonomous framework.

A. Packet Delivery Ratio

Figure 4, shows the average packet delivery ratio when using single channel vs. using all 16 channels in case of 1sec data rate. The results are averaged over all runs. This graph confirms that our framework is able to reap the benefits of channel hopping, the single channel scenario has a lower packet reception rate that varies across the channels. This is due to the fact that there is usually no single channel which is persistently reliable most of the time. On the other hand, the ABCH mechanism increases the reception rate because if the current channel is bad the next retransmission will be done on a different channel, thus increasing the probability of successful transmission. Similar trends are observed also for scenarios with 30s and 60 seconds traffic generation period.

B. Energy Consumption

Figure 5 shows the average per-node energy-consumption for both the ABCH mechanism and the single channel solutions (for all channels), where the average is only taken among the nodes being on the path of any source flow. We can observe from the figure that the energy consumption of the single channel solution is much higher than with all 16 channels available. This is due to the higher number of retransmissions carried out on lossy channels. It is worth noting that our framework save the energy consumption by about 50%. This is due to the fact that our ABCH mechanism is performed on a per-packet basis. For instance, if packet p is lost in ch_i, then the retransmission will be performed in the next channel ch_{i+1}.

C. Impact of the Multi-flow Overlap

We study the performance of the multi-flow overlapping mechanism in terms of both energy-consumption and end-to-end packet loss rate under multi-flow traffic. For this we use the same setting as explained in Section IV-B, but without channel errors. This ensures that packet loss are due to flow collisions at forwarders and not due the channel errors. We have varied the number of paths sharing one forwarder from one to five. Specifically, within a single run, each source picks a random period ranging from 1 sec to 60 sec. The long simulated time of one week / 168 hours guarantees the occurrence of collisions. In Figure 6a we show the impact of the number of flows on the packet loss rate with and without applying the overlapping mechanism. The confidence intervals are very tight, the 95% confidence intervals for the packet loss rate is within ±0.06% and ±0.12% with and without applying the overlapping mechanism, respectively. For the energy consumption the 95% confidence intervals are within ±0.003 Joules. The figure shows that without applying the overlapping mechanism the packet loss rate increases steeply as the number of flows increases. However, when applying our overlapping mechanism the packet loss rate increases much slower. In Figure 6b we show the relationship between number of flows and the energy consumption for the same simulations. This figure shows that the energy consumption increases with the number of flows, presumably due to retransmissions after collisions. Furthermore, it can be seen that the overlapping mechanism has a modest additional cost over the case without the overlapping mechanism, coming from additional times that the forwarder has to be awake.

D. Length of Learning Phase

Our autonomous framework depends on obtaining good estimates of the period and the relevant quantiles (wich for the assumed normal distribution boils down to finding the average and variance of the interarrival time). The quality of these estimates can be expected to depend on the length of the learning phase. To get more insight into this, we vary the length of the learning phase (expressed as number of packets to be observed) and observe both the energy consumption and packet loss rate in an otherwise error-free channel. Figures 7a and 7b show the impact of the length of the learning phase on
both measures. For this result, the 95% confidence intervals are within ±0.011% for the loss rate, and ±0.002 joul for the energy consumption. It is interesting to find that the packet loss rate or the energy consumption is more or less constant regardless of the length of the learning period. So the length of the learning phase does not really affect the performance. This is because the system continues to improve the estimators based on all arrivals and reacts in an adaptive manner.

### E. Length of Wakeup Window

In this section we evaluate the influence of the length of wakeup window on the performance of the system. As customary when dealing with normal distributions, we express the wakeup window as multiples of one standard deviation, \( \sigma \). Figures 8 and 9 show the impact of the wakeup window length (as multiples of \( \sigma \)) on the loss rate and energy consumption, respectively. For these graphs, the 95% confidence intervals are within ±0.17% loss rate and ±0.0035 Joules for the energy consumption. The packet loss rate behaves as one would expect: smaller values of \( \sigma \) lead to higher packet loss rates (please note that the default value of \( \alpha \) is 2). The behaviour for the energy consumption is less straightforward: Figure 9 shows that the energy consumption for \( \text{sigma} = 1 \) is much higher than for larger values of \( \sigma \). To explain this, we recall from Section II-A that a forwarder goes back from the operational state into the (much more energy-consuming) learning state after having observed too many packet losses.

With \( \sigma = 1 \) the probability that this transition rule is triggered (after retransmissions failed) is substantially higher than for the larger values of \( \sigma \). The differences in energy consumption for the larger values of \( \sigma \) are smaller, but for \( \sigma = 3 \) it is noticeably larger than for \( \sigma = 2 \).

### VI. RELATED WORK

In this section we discuss related work in the area of power-saving and communication reliability schemes operated at the MAC layer for both single and multi-channel solutions. The main factor contributing to the energy dissipation of sensor nodes often is idle listening, i.e. nodes listening on the channel in expectation of incoming packets. Other factors contributing to energy-consumption are: collisions, overhearing, control packets such as clock synchronization and management packets, and other protocol overheads. Energy-aware sensor network protocols may be broadly classified into three main categories: asynchronous contention-based [19],[5], synchronous contention-based [26],[23], and schedule-based [24],[20]. All these solutions including our previous publication which introduced a novel scheme of extending the sleep times of wireless sensor nodes on-line and in decentralized way [15] are restricted to work in a single channel solution. This work is most closely related to energy-efficient multi-channel solutions.
in WSNs. A recently standardized solution for a multi-channel system is WirelessHART [10]. WirelessHART is a TDMA-based system which uses a centralized scheduling mechanism. WirelessHART and all the proposed multi-channel solutions in WSNs [13],[1],[10] require a tight time synchronization and extensive signaling overhead to communicate and to be able to keep channel switching times consistent even when there is no need for communication in the near future [14]. Moreover, because these protocols require communication schedules to be computed and distributed (using a centralized manager) in advance, it is relatively expensive (design, time, and energy) to adapt the network to new topologies or load situations. Our work distinguishes itself from the aforementioned schemes, by supporting both energy-efficiency and communication reliability without the associated costs. It schedules the whole network activities in a distributed manner and without maintaining an explicit time synchronization protocol, thus reducing the signaling load and saving overall system complexity. This paper is to the best our knowledge, the first to explore channel hopping without maintaining a tight time synchronization protocol.

VII. Conclusions and Future Research

In this work we presented a scheme which supports periodic traffic flows and frequency hopping without requiring an expensive protocol infrastructure providing synchronization features (time synchronization, hopping synchronization) by relying entirely on the periodicity of the traffic itself for synchronization purposes. Our approach is extremely light in terms of signaling, only ACK packets need to carry few bits of information. We believe that our approach is an attractive alternative to WirelessHART and similar systems in lightly loaded networks with periodic traffic. The main contribution of this paper is the asynchronous channel hopping mechanism, which we have integrated with the distributed wakeup scheduling scheme. Furthermore, we propose a light and efficient controller to eliminate the collision caused by multi-flow overlap at forwards. We have evaluated the proposed scheme in a range of scenarios using real-world measurements and trace-based simulations. We have shown that the autonomous framework indeed reaps the benefits of frequency hopping and can also improve energy consumption over systems working on just a single channel. Furthermore, our results show that the proposed scheme works at a very good level of reliability, and in addition has only little implementation complexity. In the future, we plan to further investigate different channel hopping patterns such as adaptive channel hopping in which a dynamic estimate of the channels is maintained.

REFERENCES


