TKN Telecommunication Networks Group

Technical University Berlin

Telecommunication Networks Group

Self-learning and adaptive scheme for Supporting periodic Multi-flows in Wireless Sensor Networks

Osama Khader[⋆], Andreas Willig[•], and Adam Wolisz[⋆]

⋆ *Telecommunication Networks Group*
Technische Universitat Berlin
Berlin, Germany
Email:khader@tu-berlin.de
Email:awo@ieee.org

• *Department of Computer Science and* Software Engineering
University of Canterbury
Christchurch, New Zealand
Email: andreas.willig@canterbury.ac.nz

Berlin, March 2013

TKN Technical Report TKN-13-002

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Abstract

In this research report 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 approach comprises there main components: estimation and identification of the flows, asynchronous channel hopping and local dynamic multiple sleep states scheduling. We also propose a light and efficient controller to eliminate the collision caused by multi-flow overlap. We present detailed design, implementation, and evaluation of our autonomous framework using real-life measurements, and realistic trace-based simulation. The results show that our asynchronous channel hopping solution improves the packet reception rate without the need of an expensive signalling and time synchronization overhead. We also show that with this scheme the average energy consumption yields a $\approx 50\%$ lower than the single channel solution. This work is to the best our knowledge, the first to explore channel hopping without maintaining a tight time synchronization protocol.

Contents

Chapter 1

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 [24], [25], [9]. 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 [5], and therefore it has been

adopted in recent standards for industrial wireless sensor networks, for example Wireless-HART and ISA100.11a, see [19], [13], [16] [3]. 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 overhead. 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 work 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 [18]), however in this research report we extend the work to multi-flow and multi-channel scenarios. Secondly – and this is the novel contribution of this work – 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 technical report) 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 research report 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 report is structured as follows. Chapter 2 presents an overview of the general autonomous framework of wake-up times scheduling approach and algorithms. Subsequently, in Chapter 3 we describe the experimental jitter distribution measurements, experiment set-up, and results. The asynchronous channel hopping mechanism is presented in Chapter 4. In this Chapter we also introduce the multi-flow overlapping mechanism. In Chapter 5 we analyze the transceivers energy-consumption when the sleeping capabilities are

more fully exploited. The methodology, performance metrics, and experimental setting are explained in Chapter 6. Chapter 7 presents the result of the autonomous framework. Related work is discussed in Chapter 8 and finally, Chapter 9 concludes the research report with some future works.

Chapter 2

Autonomous Framework Overview

In this chapter we propose to develop an autonomous framework which capable of satisfying and supporting communication reliability and energy consumption for period traffic applications. Our distributed and self-learning framework includes, flow estimation algorithm, asynchronous channel hopping, overlapping controller, and dynamic multi-flow wakeup scheduling (see figure 2.1). Please note that our autonomous framework is independent from the main MAC functionalities. In the following sections we describe in detail the components of the autonomous framework.

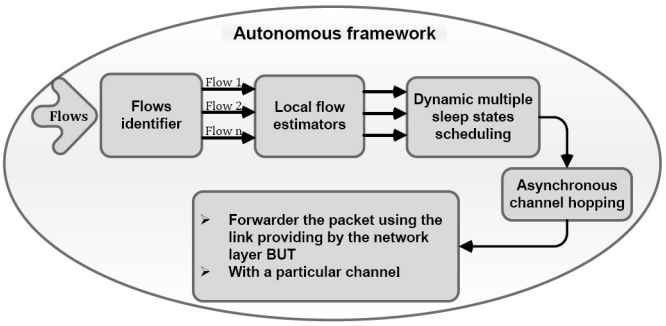

Figure 2.1: Autonomous components

2.1 Flow Estimation and Identification

The first building block of our scheduling approach is the design of appropriate estimators. Let us focus on one particular forwarder node. It receives multi-flow periodic traffic. Thus, the first estimator is the period estimator $\widehat{p_i}(n)$, which returns an estimate of the traffic period per-flow. However, the knowledge of traffic period alone is not really sufficient to schedule the node wakeup/sleep times. This is due to the imprecise time (clock drift and offset), and the activities of the previous sources or forwarders (random access, cross-traffic, etc.), there is inevitably some jitter; i.e. the actual arrival times of the packets deviate from their nominal arrival times determined by the packet generation period. It is therefore of paramount importance for the forwarder node to acquire knowledge not only of the nominal

traffic period, but also of the jitter quantiles. Therefore, the second important estimator is the quantile estimator which returns the estimate α-quantile of the jitter distribution. This knowledge shall be used to schedule the sleep and wake-up times of the forwarder so that not too many packets are lost because the forwarder sleeps at their arrival. Instead of assuming a particular distribution of the jitter we intend to measure it, and check whether we can model it or not. Another important estimator is the loss-rate estimator $\widehat{l}(n)$, which computes the local loss rate between the forwarder and its successor. It mainly operates on the sequence numbers, but since the sequence number space is in general finite and ambiguities might occur, the packet arrival timestamps are also taken into account. Please note that, these estimation are explained in Chapter 3. For identification of flows, we exploit the flows IDs generated by the source nodes. We then use these IDs for the identification process within the estimation algorithms.

2.2 Node States

The second major building block is the introduction of two separate node operations: the *learning phase* and the *operational phase*.

- In the *learning phase* each forwarder node does not sleep but unconditionally tries to capture enough packets in order to obtain reliable first estimates of the period and jitter for each flow. This state is entered after the node has been switched on or too many losses have been incurred during the operational state. The latter can occur for example when the cross-traffic situation, and therefore the actual jitter variance changes. All existing historical data is dropped upon entering the learning phase, and each individual node initially starts listen on a any of the available channel. It selects a random channel and then waits for a packet. Upon reception of the first packet the forwarder switches to the next channel, and waits for the next packet. We assume that each source node transmits its data packet using channel hopping (per-packet base). The channel switching mechanism is discussed in more detail in Chapter 4. Please also note that, in our solution each forwarder node can also start listening on a particular channel (default channel) in the learning phase. This default channel might decrease the joining time (to get the first sample) in case more that one node switch to the learning phase at the same time. The end of the learning phase is determined by a *stopping rule*, which in general can take the achieved accuracy of the period and jitter into account or can simply stop after recording a prescribed number of packets.
- In the *operational phase* the dynamic multiple sleep states scheduling is applied. The forwarder alternates between sleep phase and activity phase.
  - The forwarder maintains three main predicted times for each flow (see Figure 2.2):
    - $t_w(n)$ refers to the wakeup time of the n-th activity phase (i.e. it denotes the start of the activity phase)
    - $t_s(n)$ refers to the sleep time (denoting the maximum end time of the n-th activity phase), and
    - $t_a(n)$ is the nominal packet arrival time for the n-th activity phase.

For the n-th activity phase, the forwarder schedules wakeup for the time $t_w(n)$. The forwarder remains awake until either a packet is received and forwarded, or until sleep time $t_s(n)$ is triggered. At the end of the activity phase the forwarder updates its estimates of the period $\widehat{p_i}(n)$, jitter $\widehat{q}(k)$ and loss rate $\widehat{l}(n)$. From the updated jitter the quantiles $\widehat{q}(k)$ of the jitter distribution are updated. Based on this, the times for the $n+1$-st activity period are calculated as follows :

$$\begin{aligned} t_a(n+1) &= t_a(n) + \widehat{p}(n) \\ t_w(n+1) &= t_a(n+1) - \widehat{q}(n) \\ t_s(n+1) &= t_a(n+1) + \widehat{q}(n) \end{aligned}$$

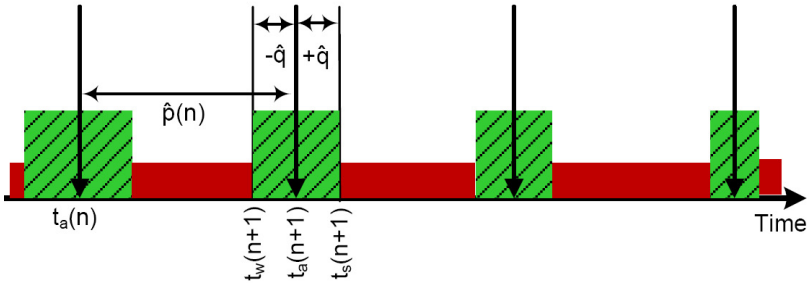

Figure 2.2: The figure shows the three predicted timers $t_a(n), t_w(n),$ and $t_s(n)$ for the wakeup and sleep scheduling algorithm

As the second major action at the end of the activity phase, the forwarder decides about a possible state transition back into the learning phase. We refer to the rule used for this as the *transition rule*. Please note that, in this approach the forwarder continuously updates its period, packet loss and jitter estimates in the operational state.

Chapter 3

Experimental Jitter Distribution Measurements

In this Chapter we study and analyze the characteristics of the jitter distribution under general and realistic scenarios. Rather than making strong assumptions about the nature of the jitter distribution, we attempt to measure it and check if we can fit the jitter distribution to a theoretical one.

3.1 Experimental Setup

We carry out our experiments on the TWIST testbed (TKN Wireless Sensor Networks Testbed) [11]. It has approximately 102 Tmote sky nodes spread over three floors of our FT building at the TU Berlin campus, resulting in more than 1500 m2 of instrumented office space. Each mote integrated with the popular IEEE 802.15.4-compliant ChipCon CC2420 radio transceiver [4] which operates in the 2.4 GHz ISM band and has a data rate of 256kbps. In its 2.4 GHz, it has 16 channels (from 11 to 26), 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 [14], [8] and its default protocol stacks. We also used the well known TinyOS CTP routing protocol [10] to construct routes connectivity 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. During each experiment-run, each source transmits 5000 packets in each channel to the sink via a set of forwarders path. MAC-layer acknowledgement is enabled and if the packet is lost the node tries to retransmit the packet for a maximum of two retires. Each forwarder records the timestamps of the received packets, source and destination addresses, flow ID, packet sequence, RSSI, LQI, and the frequency channel. In our experiments data 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 5 neighbours and communicate on all of the available channels (16 channels). Th minimum, average and maximum number hops from any source to the sink node was, 3, 6 and 8, respectively.

3.2 Discussion and Result for Jitter Measurements

We conducted several experimental runs, measuring the jitter under various scenarios, including different channels, random topology, multiple flows, cross traffics, varying the number of hops and the packet generation period. As a matter of fact, in general the jitter distribution depends on a multitude of factors: the MAC protocol, the local load situation, and the position of a particular forwarder in the network topology.

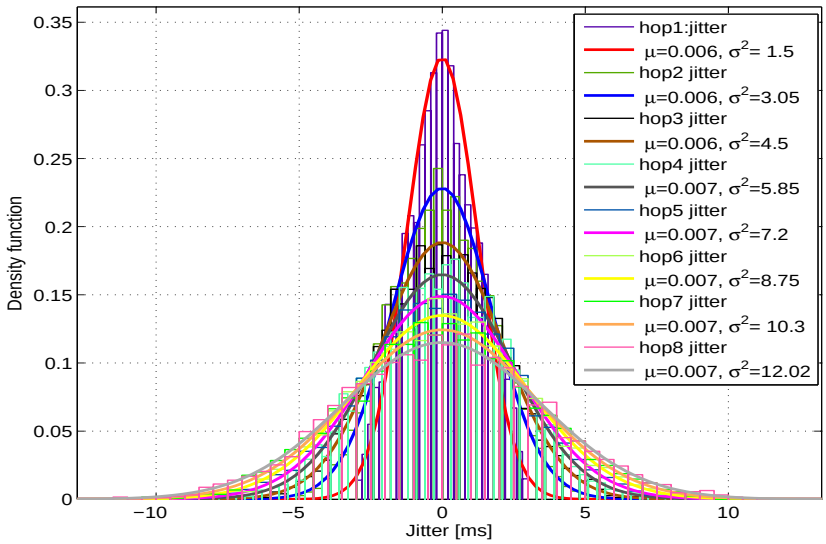

Figure 3.1: Multi-hop jitter PDF for channel 11

For all the scenarios we find that the per-flow jitter distribution is well modelled by a normal distribution. For brevity we only show the PDF and the quantile-quantile plots for channel 11 as shown in Figures 3.1 and 3.2, respectively. Please note that this finding was also conformed in our previous paper [18], which is limited to a singe source scenario with a common single channel solution.

However, here we extend the work to a more general scenario with multiple channels solution. Please also note that similar trends are observed also for scenarios with 30 and 60 seconds traffic generation period and for different channels.

3.3 Estimation of Quasi-Periodic Traffic

Direct estimation of quantiles is non-trivial and relatively memory-intensive [20, Sec. 9.5] as compared to the estimation of simple averages. In order to handle this issue we apply parametric approach, the class of distribution functions for the jitter distribution is known a priori (for example from measurements) and the task reduces to the problem of estimating the actual parameters of the distribution and the subsequent computation of adjusting proper window for sleeping activities. It turns out that even for the parametric case considered in

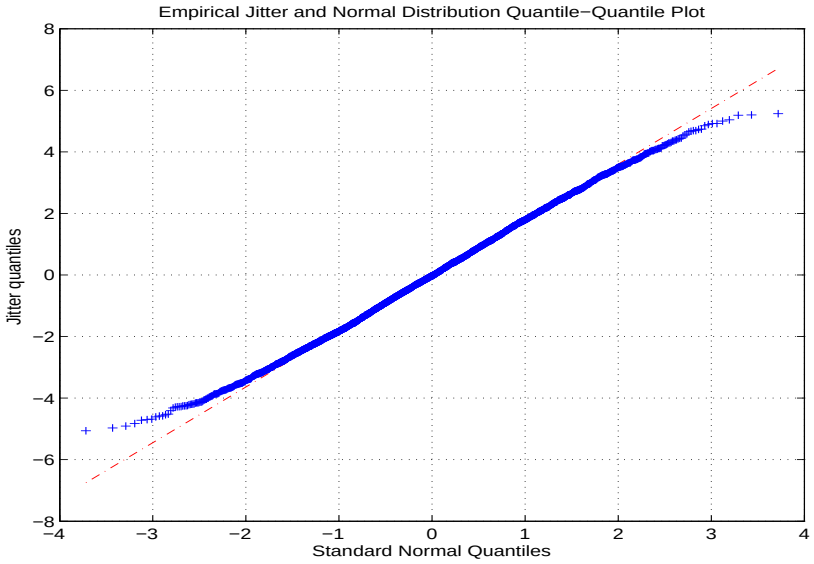

Figure 3.2: Quantile of empirical jitter against quantile of normal distribution for channels 11 using TWIST testbed

this research report nothing more than a variance estimate for the jitter is required (Based on our measurements, see Section 3.2);

Since in our measurements, we want to compute the current estimate from the previous estimate and the current measurement, we use recursive estimators for the traffic period and the jitter variance. The mean period for any independent flow f can be computed as:

$$\widehat{X}_{f,k+1} = \widehat{X}_{f,k} + \frac{1}{k+1}\left(X_{f,k+1} - \widehat{X}_{f,k}\right) \tag{3.1}$$

where $\widehat{X}_k$ refers to the estimated mean packet inter-arrival time after the k-th packet and X_k is the k-th actual interarrival time – interarrival times are only obtained for two successively received packets. For any independent flow f, a recursive method can be found to compute for the jitter variance as:

$$\widehat{\sigma}^2_{f,k+1} = \sigma^2_{f,k} + \frac{1}{k+1}\left[\frac{k}{k+1}(X_{f,k+1} - \widehat{X}_{f,k})^2 - \sigma^2_{f,k}\right] \tag{3.2}$$

where σ^2_k denotes the estimated jitter variance after observing the k-th interarrival time per flow f.

3.3.1 Mean Estimator

Given k measurements of packet inter-arriving time X_i the sample mean is

$$\widehat{X}_k = \frac{1}{k}\sum_{i=1}^{k} X_i \tag{3.3}$$

Suppose that $\widehat{X}_k$ has been computed based on measurements Now one more measurement X_{k+1} is made. The new sample mean is computed as :

$$\widehat{X}_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} X_i \tag{3.4}$$

$\widehat{X}_{k+1}$ can be computed in terms of $\widehat{X}_k$ and X_{k+1} by proceeding as follows.

$$\begin{aligned}\widehat{X}_{k+1} &= \frac{k}{k+1}\left(\frac{1}{k}\sum_{i=1}^{k} X_i\right) + \frac{1}{k+1}X_{k+1} \\ &= \frac{k}{k+1}\widehat{X}_k + \frac{1}{k+1}X_{k+1}\end{aligned}$$

So that the estimator for the mean for any independent flow f is:

$$\widehat{X}_{f,k+1} = \widehat{X}_{f,k} + \frac{1}{k+1}\left(X_{f,k+1} - \widehat{X}_{f,k}\right) \tag{3.5}$$

The random variable (packet inter-arrival time) is denoted by $x(k)$ and its estimated value of the mean is denoted by $\widehat{x}(k)$.

3.3.2 Variance Estimator

For any independent flow f, a recursive method can be found to computer an estimator for the variance:

$$\widehat{\sigma}^2_{f,k+1} = \sigma^2_{f,k} + \frac{1}{k+1}\left[\frac{k}{k+1}(X_{f,k+1} - \widehat{X}_{f,k})^2 - \sigma^2_{f,k}\right] \tag{3.6}$$

Where is $\sigma^2_{f,k}$ represents the kth estimate of variance per flow f.

Chapter 4

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 [6] and ISA100.11a [16], 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 statistics of channel qualities for each link. Third, each pair of nodes needs to achieve consistent rankings of the individual channels, otherwise they will may end up communicating using different channels. This requires additional signaling. For these reasons we base our asynchronous blind channel hopping (ABCH) on the blind hopping approach.

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:

$$NextChannel = (SQ + chOffset) \mod chNum \tag{4.1}$$

where SQ is the next sequence number, $chOffset$ is the channel offset and $chNum$ 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 4.1. As an example, if $chOffset = 1$, $chNum = 16$ 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.

4.1 Handling Transmission Errors

Figure 4.1 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 4.1 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 time-out 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 transmitter 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 2.2. 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_3 on channel 13. Upon receiving p_3, 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_3, 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_6 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_7, 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

[1]Please note, that in this circumstance, the forwarder increase 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).

loss, thus more energy-efficient.

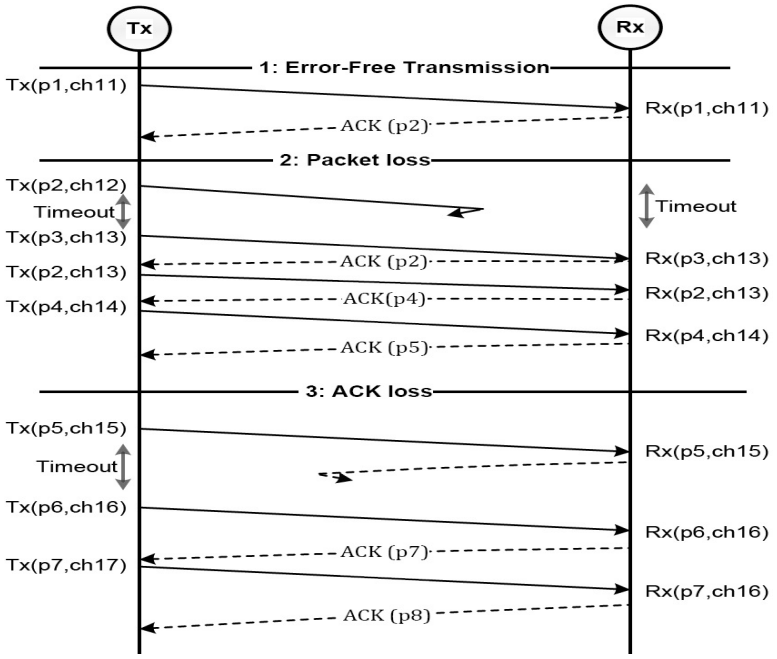

Figure 4.1: Autonomous channel hopping transition diagram

4.2 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 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.

Chapter 5

Local Dynamic Sleep State Scheduling

In this Chapter we analyze how an improved usage of the transceiver sleep states can substantially reduce the overall energy-consumption, thereby increasing the autonomous system energy-efficiency.

Modern radios have built-in support for several sleeping states of operation with each state consuming a different amount of power. The radio also requires some time to switch into and out of different sleep states. For example, the CC2420 has three sleeping states: the idle-sleep-state, the power-down-state, and voltage-regulator-off-state, hereafter referred to as sleep-mode-1, sleep-mode-2, and sleep-mode-3, respectively. These sleep modes and their possible transitions are illustrated in Figure 5.1. In sleep-mode-1, both the voltage regulator and the crystal oscillator are enabled. The energy-saving in sleep-mode-1 state is obtained by disabling the radio frequency synthesizer which controls the channel selection and up/down RF conversion. Sleep-mode-1 has the fastest transition time of around 0.192 ms and consumes 1.4 mW of power, which is the highest among the sleep modes. In sleep-mode-2, the voltage regulator is enabled and the crystal oscillator is disabled. This mode consumes 0.07 mW of power. In sleep-mode-3, both the voltage regulator and the crystal oscillator are disabled. This mode has the slowest transition time and lowest power-consumption ($6.6 \cdot 10^{-5}$ mW). In general this mode switches off the radio chip completely, including radio RAM. As a result any packet waiting in the receiving or transmitting buffer is lost. Despite the fact that most (if not all) energy-efficient WSNs MAC protocols use the popular CC2420 Radio chip or similar Radio chip that support multiple sleep modes, they only use one single-fixed sleep mode. Moreover, they usually control and use the lightest sleep mode (sleep-mode-1) which calls in CC2420 datasheet "idle" mode. For example B-MAC [22] and X-MAC [2] uses sleep-mode-1. According to X-MAC paper: (when X-MAC sleeps the radio, in fact it puts the radio into idle mode, as sleep mode turns off the oscillator and requires a longer time to transition back to receive mode. [2]). To the best of our knowledge there is no a MAC protocol in WSNs that utilizes the multiple radio sleep states and thus provide a dynamic assignment of multiple sleep modes. According to our previous study we showed that about 40% of the total energy-consumption is due to the sleeping activity thus, we believe that our approach elegantly copes with the issues raised above, and it will become commonplace for

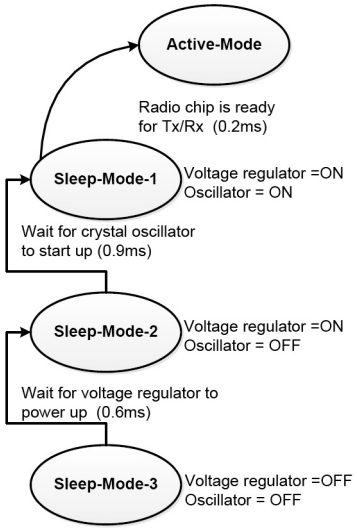

Figure 5.1: Sleep transition states for CC2420 Radio.

the energy-saving mechanisms in WSNs MAC protocols under variant periodic traffic rates.

In the remainder of this Chapter we propose a generic simple approach that may runs on top of any MAC layer protocols and in which each node can apply individually, based on its current traffic rate.

5.1 Dynamic Multiple Sleep States Scheduling (DM3S)

In what follows, we propose a practical and effective dynamic multiple sleep states scheduling scheme, abbreviated as DM3S. It exploits the multiple sleep states of the CC2420 radio and utilizes them based on the estimation of the next packet arrival. This approach is independent of the underlying link scheduling algorithm, but a node uses its given schedule to determine the right sleep states. For ease of presentation, we called the scheduled activity window (Tx or Rx interval time) as a time slot. Generally speaking, a nodes activities are constrained to certain slots (whether these are exclusive or shared does not matter for the following presentation), whereas in all other slots they can sleep. We call the slots that a node might be involved in its **active slots**. There will generally be some active slots in which a node will have to wake up unconditionally, for example those slots in which the node is scheduled to receive, or those transmit slots where a packet is transmitted the first time. On the other hand, retransmission slots are only used when a transmission in a previous transmit slot has failed (i.e. the sender has not received an acknowledgement). A key observation is that at the end of a transmit slot the sender will know if it has to utilize a retransmission slot or not. More generally, based on its schedule and the transmission outcomes in the current active slot, at the end of the current slot a node can determine how much time will elapse before its next active slot starts.

The second key ingredient is borrowed from a technique used in dynamic power management to control the device's operational states, see [15] and [1]. Specifically, since the number

of transceiver states and their switching time is known a-priori, it is possible to construct a function $\phi(\cdot)$, which takes a non-negative time duration τ as a parameter and which returns a sleeping schedule that: (i) ensures that after τ seconds the node transceiver is ready to transmit or receive, (ii) sends the transceiver through a "monotone" sequence of sleep states (the deepest state at the beginning and the lightest state at the end), and that (iii) ensures that the chosen sequence of states (and the times being spent in each visited state) has the smallest energy-consumption over the time horizon of τ seconds. For the CC2420 transceiver this function $\phi(\cdot)$ is straightforward to construct. Specifically, we need to determine three threshold values: (i) a duration τ_1 that is minimally needed to make sleep-state-1 more energy-efficient than to stay awake; (ii) a duration $\tau_2 > \tau_1$ that is minimally needed to make an initial choice of sleep-state-2, followed by a transition through sleep-state-1 and subsequent wakeup more energy-efficient than to start initially with sleep-state-1; and (iii) a duration $\tau_3 > \tau_2$ that is minimally needed to make an initial choice of sleep-state-3, followed by a transition through sleep-state-2, sleep-state-1 and subsequent wakeup more energy-efficient than to initially start with sleep-state-2. When at the end of an active slot it takes a time τ before the next active slot starts, it is a simple matter of comparing τ to the three thresholds τ_1, τ_2 and τ_3 to figure out which sleep state (if any) should be entered next. The run-time overhead caused by this computation is only moderate.

Chapter 6

Methodology and Experimental Setting

The study of selecting the right methodology for evaluating the above schemes is non-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-life experiment does not provide the ability to evaluate different schemes or algorithms under the exact same condition as the RF environment is time-varying in nature. In this work we decided to combine these two methods and use a connectivity traces gathered from a real world deployment as an input to the simulation. We believe that combining the two methods is essential for evaluating such complex environment.

6.1 Connectivity Traces

We evaluate our approach under both single channel and multiple channel using a connectivity traces gathered in a real world deployment. The connectivity traces are collected by DUST networks group [5]. The network deployed was 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. As shown in Figure 6.1 (as example), 45 sensor nodes were deployed in an indoor printing facility in a relatively uniform distribution. Each node occupies with ChipCon CC2420 radio chip [4]. For this experiment, the data consisted of periodic reports on the quality of the communication path, where a path represents all transmission between a pair of nodes. Each node allows to have up to 8 neighbours and communicates on all of the 16 channels of the IEEE802.15.4. The trace contains all path-channels reports. Each report contains the following statistics for a path-channel: path ID, channel ID, number of transmit attempts, number of transmit fails, number of transmits without ACK reception, and the mean RSSI and LQI. The link quality was evaluated based on the packet reception ratio, the ratio between the number of successfully received packets and the number of transmission attempt. For more details about the setup of the experiment please refer to [5].

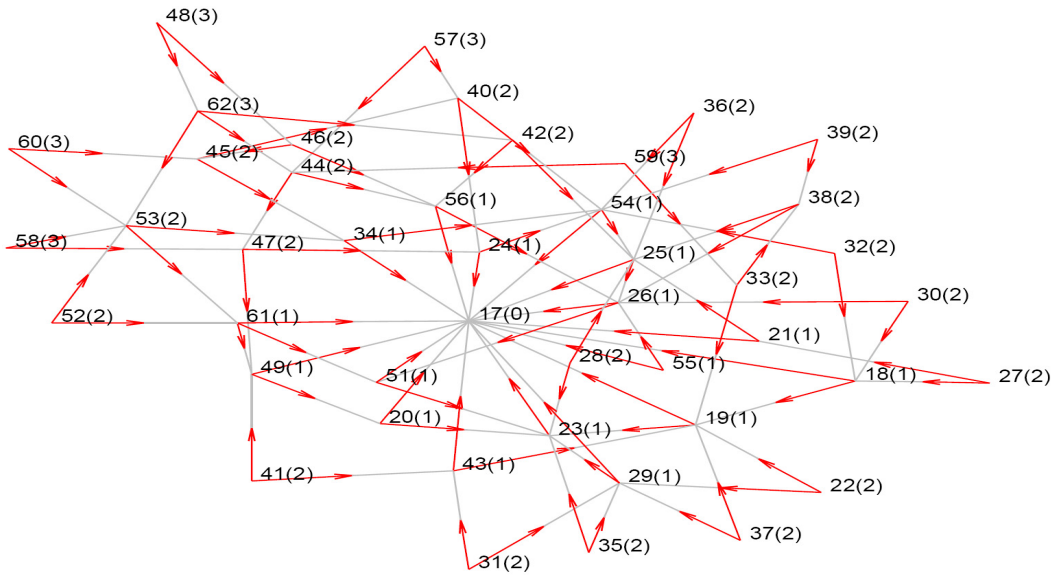

Figure 6.1: An example of multi-hop network topology

6.2 Simulation Setup

In order to realize a simulation model to study the performance of autonomous framework over a wireless multi-hop network, we have chosen connectivity traces, and the well-known OMNeT++ [28] simulation environment together with the Castalia WSNs framework [21]. 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 6.2 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 listed in Table 6.1.

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.

6.3 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 $\times$ 225feet, 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

Table 6.1: Main Autonomous Framework Parameters

Parameter	Value
Data packet size	128 bytes
Ack packet size	12 bytes
Number of retry	2 times
Length of learning phase	Varying
Number of flows	Varying
Data packet rate	Varying
Number of neighbor	5 nodes
Allowable packet loss rate α	Varying
Loss threshold	3 packets
Channel switching time	192 μs

Table 6.2: Main CC2420 Power Consumption Parameters

Main power consumption parameters of CC2420 radio			
Notation	Parameters	I(mA)	Power(mW)
P_{Tx}	Transmit power (0dBm)	17.4	57.42
P_{Rx}	Receive power	18.8	62.04
P_L	Listen power	18.8	62.04
P_{S-m1}	Sleep-mode-1 power	0.426	1.406
CPU_A	CPU active power	1.8	6
CPU_S	CPU sleep power	0.045	0.148

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 forwarders. 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.

6.4 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.

Chapter 7

Result

In order to study the performance of our autonomous framework, we compare the same set-up as described in Section 6.2 using the asynchronous channel hopping and single channel solutions. We first investigate the packet delivery ratio and the total energy-consumption. Then we study the impact of the multi-flow overlap to the energy-consumption and packet loss rate. We also investigate the impact of the length of the learning phase on the performance of the autonomous framework.

7.1 Packet Delivery Ratio

Figure 7.1, 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.

7.2 Energy Consumption

Figure 7.2 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 may reduce 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}.

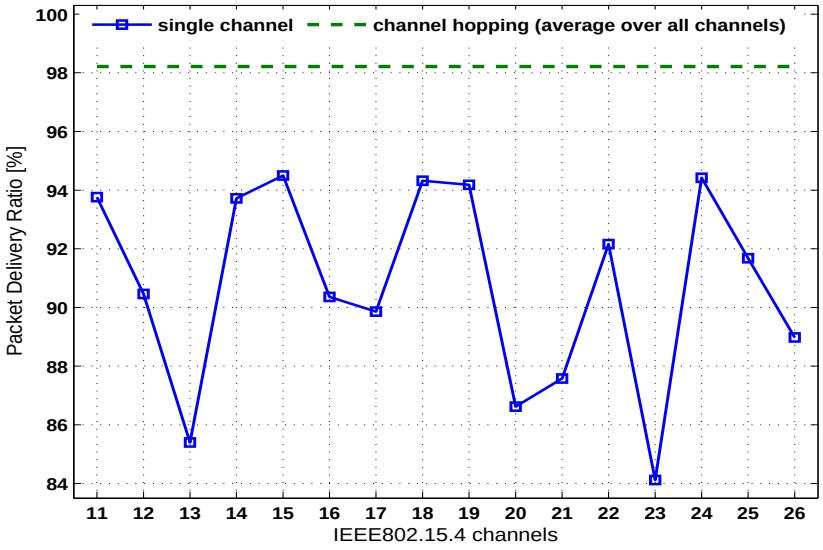

Figure 7.1: Average PRR: Single channel vs blind channel hopping

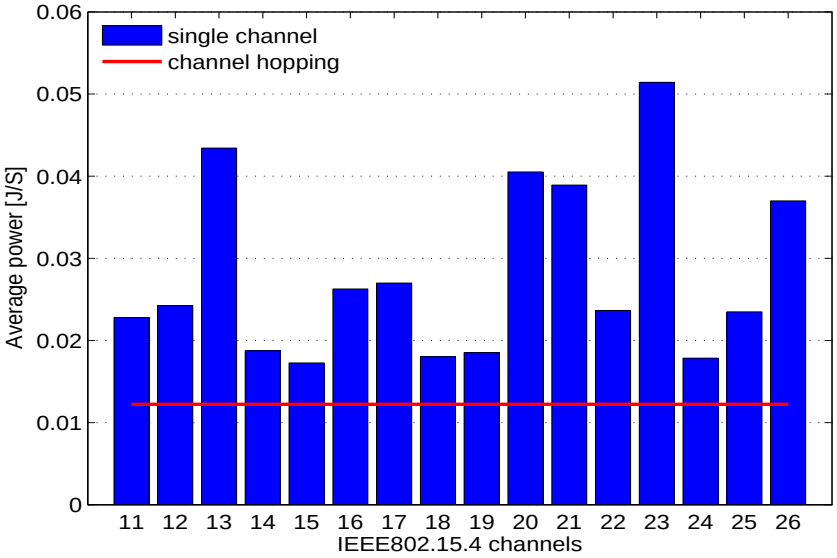

Figure 7.2: Average energy: Single channel vs blind channel hopping

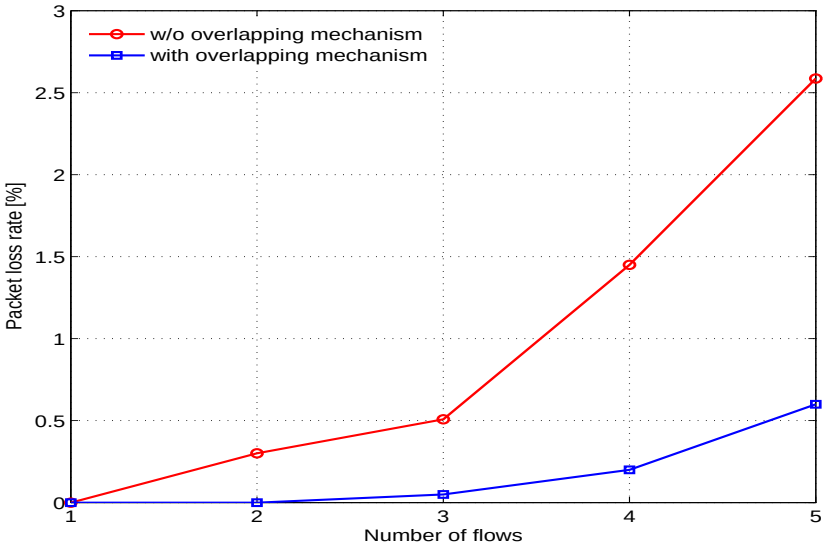

Figure 7.3: Impact of multi-flow overlap in packet loss

7.3 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 6.2, 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 7.3 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 $\pm 0.06\%$ and $\pm 0.12\%$ 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 7.4 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.

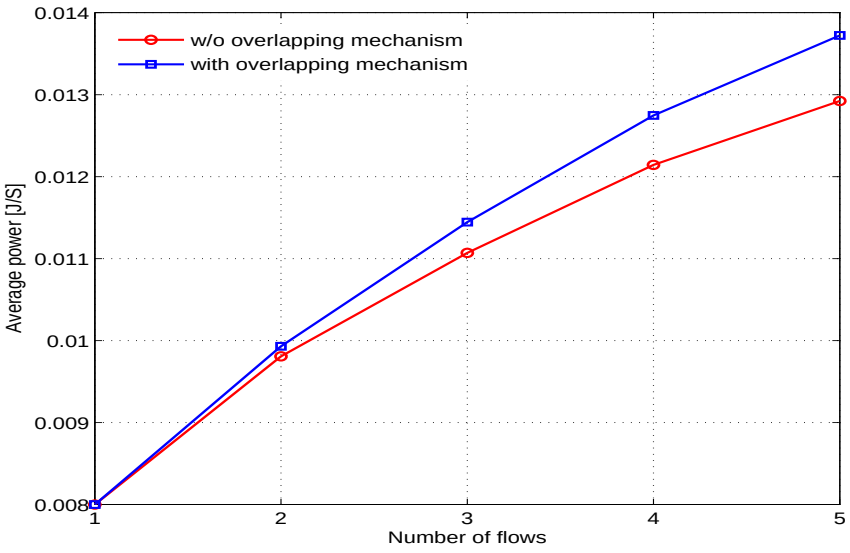

Figure 7.4: Impact of multi-flow overlap in energy consumption

7.4 Length of Learning Phase

Our autonomous framework depends on obtaining good estimates of the period and the relevant quantiles (which 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 7.5 and 7.6 show the impact of the length of the learning phase on both measures. For this result, the 95% confidence intervals are within $\pm 0.011\%$ for the loss rate, and ± 0.002 Joules 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 a adaptive manner.

7.5 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, σ. Figures 7.7 and 7.8 show the impact of the wakeup window length (as multiples of σ) on the loss rate and energy consumption, respectively. For these graphs, the 95% confidence intervals are within $\pm 0.17\%$ loss rate and ± 0.0035 Joules for the energy consumption. The packet loss rate behaves as one would expect:

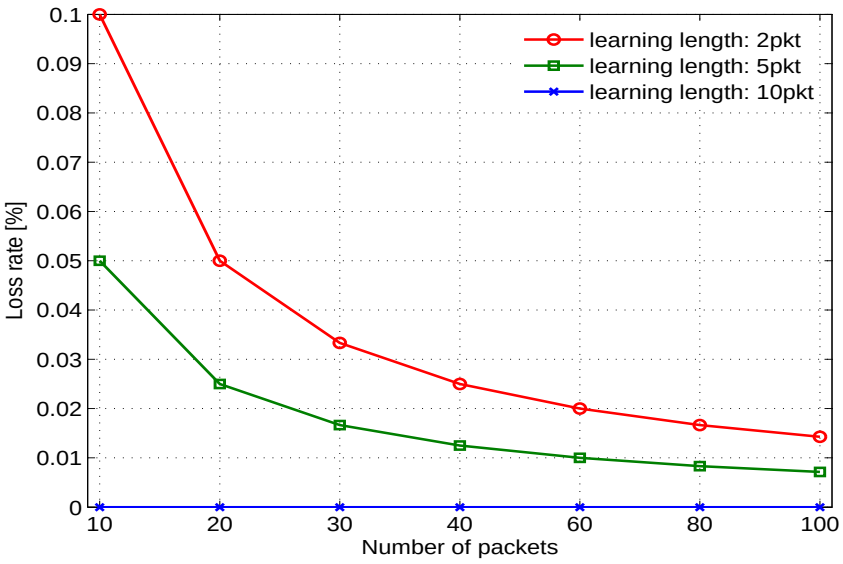

Figure 7.5: Length of learning phase vs packet loss rate

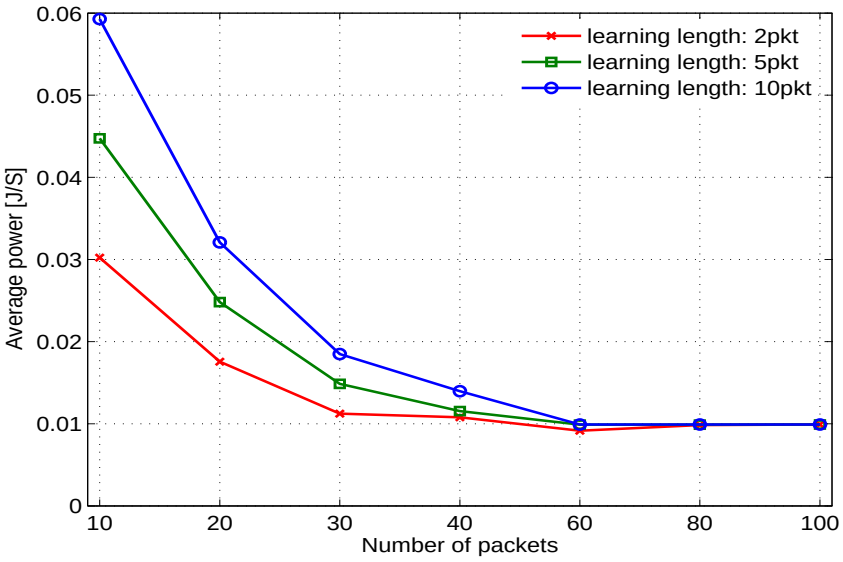

Figure 7.6: Length of learning phase vs average energy consumption

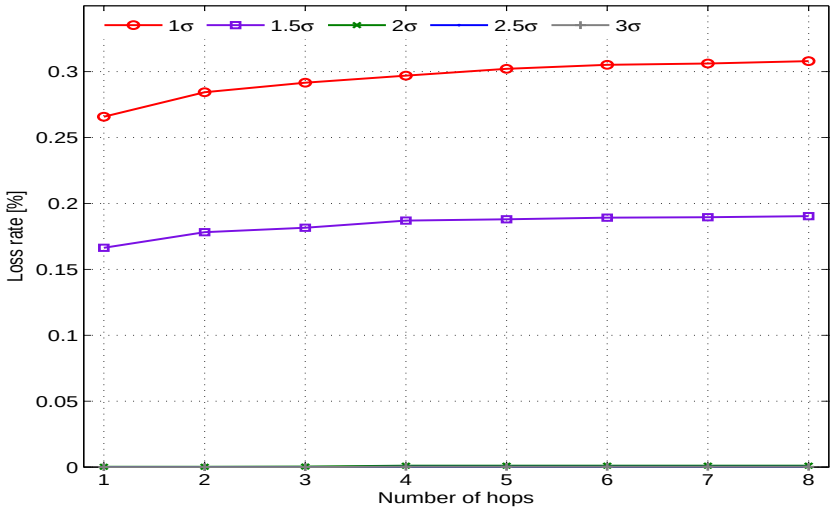

Figure 7.7: Length of wakeup window vs packet loss rate

smaller values of σ lead to higher packet loss rates (please note that the default value of α is 2). The behaviour for the energy consumption is less straightforward: Figure 7.8 shows that the energy consumption for $sigma = 1$ is much higher than for larger values of σ. To explain this, we recall from Chapter 2 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 σ. The differences in energy consumption for the larger values of σ are smaller, but for $\sigma = 3$ it is noticeably larger than for $\sigma = 2$.

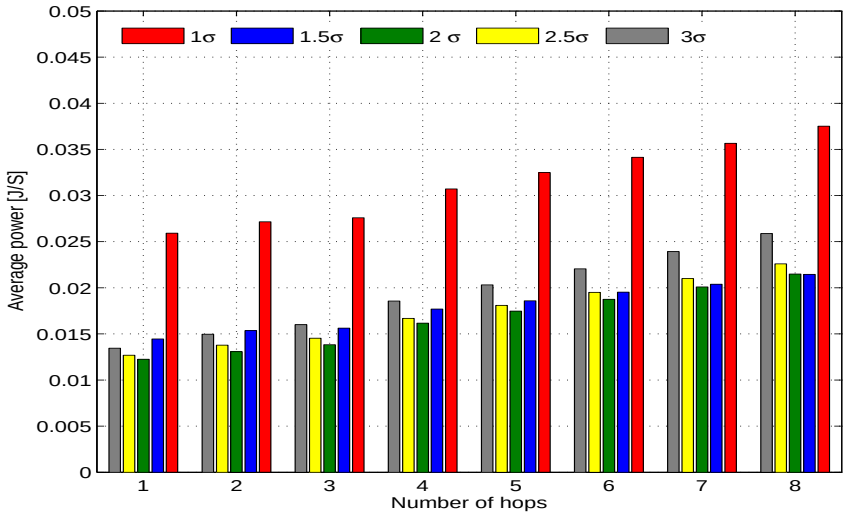

Figure 7.8: Length of wakeup window vs average energy consumption

Chapter 8

Related Work

In this Chapter we discuss related work in the of power-saving and communication reliability schemes operated at the MAC layer for both single and multi-channel solutions. One of the main concerns of low-power MAC protocols is to switch the radio into sleep mode as much as possible, otherwise energy would be wasted. For low traffic scenarios the main factor contributing to the energy dissipation is idle listening (nodes listening on the channel in expectation of incoming packets). Other factors contributing to MAC energy-consumption are: collisions, overhearing, control packets such as clock synchronization and management packets, and other protocol overheads. Energy-aware sensor network MAC protocols may be broadly classified into three main categories: asynchronous contention-based, synchronous contention-based, and schedule-based. An important example of asynchronous contention-based protocols are low-power listening protocols such as B-MAC [22], WiseMAC [7]. These protocols are asynchronous, i.e. there is no need for nodes to coordinate their wakeup cycles and therefore there is no need for clock synchronization. Each node periodically wakes up and checks the channel activity for short time without receiving any data. If the channel is idle it goes to sleep, otherwise it stays awake to receive the packet. To rendezvous with receivers, senders send a long preamble before the actual message (longer than the checking interval).

In synchronous contention-based access protocols such as S-MAC [29] and T-MAC [26], nodes sleep and wake up in a synchronized fashion, and use a contention-based access protocol to transmit data in the awake periods. Time is organized into cycles of equal size. Each cycle is divided into two time intervals. In the first time interval nodes can exchange synchronization information. In the second interval nodes may send or receive, using a CSMA approach with RTS-CTS signaling. A general problem shared by all such synchronized protocols is that communication is grouped at the beginning of each slot, raising the chances on collisions, hence limiting their dynamic range to low traffic rates only.

In schedule-based access protocols (TDMA protocols), time is sub-divided into time slots, and time slots are exclusively allocated to one specific pair of nodes, a transmitting and a receiving node. In these protocols slot assignment algorithms and tight clock synchronization are of great concern. LMAC [27] uses a simple random slot assignment algorithm that ensures that nodes at 2-hop distance do not use the same slot number. It assumes a global time synchronization. Similar to LMAC, TRAMA [23] uses a distributed election scheme to determine particular time slots, however it uses more complicated policies that take traffic

load into account, and which require relatively large amounts of memory for maintaining scheduling information among neighbors. All these protocols are restricted to work in a single channel solutions.

Our work is most closely related to multi-channel MAC protocols. Several researchers have explored the possibility of using multiple channels to overcome the limitation of single channel MAC protocols [19],[3], and [30].

Our approach is different than the synchronized protocols such as WirelessHART [12], ISA [16]. These protocols need to be tightly synchronized in order to be able to switch between channels and communicate. They are also required to have a centralized coordinator. Some of the drawbacks of such protocols are: (1) need an expensive hardware, (2) an extensive signalling overheads thus more energy-consumption. For instance and according to the WirelessHART standard [13], nodes need to resynchronize every 30s, even if there is no need to send packets in the near future [17]. Moreover, because nodes set-up schedules to communicate between each other in advance, adaptivity of network topology is usually not handled in such protocols.

A state-of-the-art solution for multi-channel system is WirelessHART [12]. WirelessHART is a TDMA-based system which uses a centralized scheduling mechanism. WirelessHART and all the propose multi-channel MACs protocols in WSNs require a tight time synchronization and extensive signalling overhead to communicate and be able to keep the switching of channels coherent even when there is no need for communication in the near future. Some of the drawbacks of such protocols are: (1) need an expensive hardware, (2) an extensive signalling overheads thus more energy-consumption. For instance and according to the WirelessHART standard [13], nodes need to resynchronize every 30s, even if there is no need to send packets in the near future. Moreover, because nodes set-up schedules to communicate between each other in advance, adaptivity of network topology is usually not handled in such protocols. In order to address these challenges, in this work we focus on supporting for both communication reliability and energy-efficient through the development of a distributed framework that integrates: asynchronous channel hopping, estimation and adaptation of multi-flows traffics, and local dynamic multiple sleep states scheduling.

Chapter 9

Conclusions and Future Research

In many applications in wireless sensor network source nodes generate and send periodic traffics to the sink node through a number of forwarder nodes. In such multi-hop networks forwarders have forwarding duties but should on the other hand support for both communication reliability and energy-efficiency. To this end, we explore a novel an asynchronous channel hopping for multi-flow periodic traffics in WSNs. Each sensor node acquires knowledge of the traffic period and its jitter in a distributed way. We then use this knowledge to let a node control its wakeup and sleep window in an efficient manner. Each individual node hops between different channels without maintaining and explicit time synchronization protocol or a centralized components. The main contributions of this research report are: (a) we propose an asynchronous channel hopping for WSNs that improves the communication reliability and energy-consumption, thus is robust to implement in distributed network. (b) We design and implement a local estimators for multi-flow traffic in which a node locally decides when to sleep and when to wakeup. (c) we propose a dynamic multiple sleep states scheduling that substantially reduce the overall energy-consumption. (d) we also propose a light and efficient overlapping resolution mechanism that reduce the packet loss due to the overlapping. We evaluate our proposed schemes using real-world experiments and realistic trace-based simulation. The results show that in multi-flow period traffic, it is possible to estimate the characteristics of traffics and adapt the node activities accordantly. Our decentralized and self-learning approach works without a prior knowledge of the traffics periodicity. We also show that asynchronous channel hopping improves the packet reception rate without an expensive signalling and and explicit time synchronization overhead. Thus, more energy-saving. This research report is, to the best our knowledge, the first to address and propose channel hopping without maintaining a tight time synchronization.

We plan to conduct a comparison study of centralized approach and our decentralized approach. Moreover we also plan to further investigate different channel hopping pattern such as white-listing channel hopping in which a dynamic estimate of the channel evaluated on-line.

Bibliography

[1] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. In *Proc. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2000,*, volume vol.8, NO. 3, pages pp. 299–316, NJ, USA, June 2000.

[2] Michael Buettner, Gary Yee, Eric Anderson, and Richard Han. X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks. In *Proceeding ACM SenSys '06 Proceedings of the 4th international conference on Embedded networked sensor systems, 2006*, pages 307–320, New York, USA, November 2006.

[3] Xun Chen, Peng Han, Qiu-Sheng He, Shi-Liang Tu, and Zhang-Long Chen. A multi-channel mac protocol for wireless sensor networks. In *Proc. of Computer and Information Technology, 2006. CIT '06. The Sixth IEEE International Conference*, pages 224 – 230, Fudan University, China, September 2006.

[4] Chipcon. *CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Available: http://www.chipcon.com.*, 2004.

[5] Lance Doherty, William Lindsay, and Jonathan Simon. Channel-specific wireless sensor network path data,. In *IEEE 16th Internatioanl Conference on Computer Communications and Networks (ICCCN), 2007*, pages 89 – 94, Turtle Bay Resort, Honolulu, Hawaii, USA, August 13-16 2007.

[6] Dust Networks. *Wirelesshart technical data sheet. White paper, Dust Networks, September 2007.*

[7] A. El-Hoiydi, J.-D. Decotignie, C. Enz, and E. Le Roux. Poster abstract: Wisemac, an ultra low power mac protocol for the wisenet wireless sensor network. In *Proc. ACM SenSys 03*, Los Angeles, California, November 2003. Poster Abstract.

[8] D. Gay, P. Levis, R. V. Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proc. ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)*, San Diego, California, USA., June 2003.

[9] Steven D. Glaser. Some real-world applications of wireless sensor nodes. In *Proc. (SPIE) Symposium on Smart Structures and Materials/ NDE 2004*, San Diego, California, March 2004.

[10] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009)*, pages 1 – 13, Berkeley, CA, USA,, November 2009.

[11] Vlado Handziski, Andreas Kpke, Andreas Willig, and Adam Wolisz. Twist: A scalable and reconfigurable testbed for wireless indoor experiments with sensor network. In *Proc. of the 2nd Intl. Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality, (RealMAN 2006)*, pages 63–70, Florence, Italy, May 2006.

[12] HART Communication Foundation. *HART Communication Protocol Specification, HCF SPEC 13 Revision 7.1, 05 June, 2008.*

[13] HART Communication Foundation. *TDMA Data Link Layer Specification, HCF SPEC 075 Revision 1.1, 17 May, 2008.*

[14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proc. ACM of the 9th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS),*, pages 83–104, 2000.

[15] Chi-Hong Hwang and Allen C.-H. WU. A predictive system shutdown method for energy saving of event-driven computaion. In *Proc. ACM Transactions on Design Automation of Electronic Systems 2000*, volume vol.5, pages pp. 226–241, New York, USA, April 2000.

[16] ISA. *ISA 100: wireless system for automation. Available: http:// isa.zigbee.org.*

[17] Osama Khader and Andreas Willig. An Energy Consumption Analysis of the WirelessHART TDMA Protocol. *Comput. Commun. (2013).* http://dx.doi.org/10.1016/j.comcom.2012.12.008.

[18] Osama Khader, Andreas Willig, and Adam Wolisz. Distributed Wakeup Scheduling Scheme for Supporting Periodic Traffic in WSNs. In *Proc. European Wireless (EW 2009)*, pages 287–292, Aalborg, Denmark, May 2009.

[19] Youngmin Kim, Hyojeong Shin, and Hojung Cha. Y-mac: An energy-efficient multi-channel mac protocol for dense wireless sensor networks. In *Proc. of IPSN '08, Proceedings of the 7th international conference on Information processing in sensor networks*, pages 53 – 63, Washington, DC, USA, April 2008.

[20] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, third edition, 2000.

[21] NICTA. *The Castalia simulator for Wireless Sensor Networks. Available: http://castalia.npc.nicta.com.au.*

[22] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proc. 2nd international conference on Embedded networked sensor systems (ACM SenSys)*, Baltimore, MD, November 2004.

[23] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. In *Proc. ACM SenSys 03*, Los Angeles, California, November 2003.

[24] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Proc. ASM SenSys '04*, pages 214–226, 2004.

[25] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *Proc. ACM SenSys '05*, pages 51–63, 2005.

[26] T. van and D. K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proc. ACM SenSys '03'*, Los Angeles, California, USA., November. 2003.

[27] L.F.W. van Hoesel and P.J.M. Havinga. A lightweight medium access protocol for wireless sensor networks. In *Proc. INSS, 2004*, 2004.

[28] A. Varga. *OMNeT++ Discrete Event Simulation System. Available: http://www.omnetpp.org.*

[29] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proc. INFOCOM 2002*, New York, June 2002. IEEE.

[30] Gang Zhou, Chengdu Huang, Ting Yan, Tian He, and John A. Stankovic. Mmsn: Multi-frequency media access control for wireless sensor networks. In *Proc. of INFOCOM 2006. 25th IEEE International Conference on Computer Communications.*, pages 1 – 13, Barcelona, Catalunya, Spain, April 2006.