

Chryso – A Multi-channel Approach to Mitigate External Interference

Venkatraman Iyer
Delft University of Technology
Email: v.g.iyer@tudelft.nl

Matthias Woehrle
Delft University of Technology
Email: m.woehrle@tudelft.nl

Koen Langendoen
Delft University of Technology
Email: k.g.langendoen@tudelft.nl

Abstract—When a wireless sensor network is deployed indoors, sensor nodes often need to compete for channel usage with other, more powerful devices such as 802.11 access points. To mitigate the effects of such external interference we propose Chryso, a protocol extension specifically designed for data collection applications that leverages the channel diversity of sensor node radios. In the face of external interference Chryso switches only the directly affected set of nodes onto a new channel. In this paper, we present the design of Chryso as well as its implementation in Contiki. Evaluation of Chryso on two different testbeds, and its comparison with a state-of-the-art channel hopping protocol stack show that Chryso is effective in maintaining a stable data flow at the sink, even under extreme interference.

I. INTRODUCTION

Wireless sensor networks (WSN) are networked embedded systems consisting of sensor nodes that collectively observe some physical phenomena. Individual sensor nodes communicate via radio and create an ad-hoc network that allows the nodes to collaboratively sense a deployment site. Sensor nodes are often deployed into an unknown environment. Nevertheless, a deployed WSN should operate autonomously, i. e., not require periodic maintenance. Thus, a WSN needs to cope with detrimental external effects such as large temperature gradients.

One specific concern for sensor networks is the radio communication. Since sensor node radios operate in the unlicensed ISM bands, they often face the problem of having to share the communication medium with other devices in the neighborhood. In particular 802.15.4 radios, which operate in the 2.4 GHz range, interfere with prominent communication technologies like Wi-Fi (802.11). This work presents a practical approach for exploiting the frequency diversity of 802.15.4 radios to mitigate the interference generated outside of the sensor network, so-called *external interference*. This paper focuses on the predominant class of data collection applications used for environmental and structural monitoring. We present the integration of Chryso¹, a multi-channel protocol extension, into a typical protocol stack of a convergecast routing protocol and a low-power MAC protocol as shown in Figure 1.

The design of Chryso considers several crucial observations: (1) External interference may only be local. Hence,

¹The protocol is named after the spider *Chryso venusta*, which has been observed to rapidly change its color when disturbed.

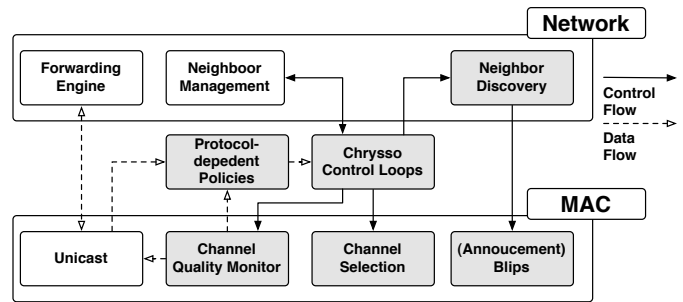


Fig. 1. Integrated protocol stack with Chryso-specific parts in grey

coordination of channel assignments to nodes should be localized. (2) External interference varies over time, so channel allocation cannot be performed offline. (3) For large networks it is not feasible, if not impossible, to determine a single channel that is not affected by any source of external interference. Accounting for these observations Chryso performs energy efficient, online monitoring of channel quality to detect external interference. Chryso handles interference locally and hence does not require global synchronization nor scheduling. When interfered, Chryso switches only the subset of affected nodes to a new channel effectively evading the interference source on the spot.

While we have described some initial ideas to cope with external interference with *Chamaeleon* [10], this work presents fundamental improvements and includes the following major contributions:

- It describes a novel mechanism for dealing with neighbor discovery in a multi-channel environment.
- It details on the implementation of Chryso, a multi-channel extension, for a typical data collection stack in Contiki.
- It shows experimental evaluations on real sensor nodes on two testbeds.

The outline of the paper is as follows: Section II describes the key ideas and high-level design of Chryso. Section III details on the implementation in Contiki. Experimental results of Chryso, obtained from two different testbeds, are presented in Section IV. Section V discusses related work on multi-channel protocols. Finally, we conclude in Section VI.

II. CHRYSO DESIGN

Before presenting the design of Chryso and its individual components, we introduce the general design goals. Additionally, we detail on Chryso's neighborhood discovery mechanism as it is a vital component of any multi-channel protocol.

A. General Design Goals

The design of Chryso is motivated by the following observations and ideas:

- Since external interference cannot be predicted, a continuous, yet energy-efficient monitoring of channel quality needs to be performed on each node. Hence, sensor nodes running Chryso continuously monitor link qualities. The detection of external interference depends on the specific protocol stack. It is implemented on top of Chryso with a set of protocol-specific policies. Example policies for our Contiki-based prototype implementation are detailed in Section III.
- External interference usually affects a group of nodes, rather than a single node. Therefore, interference detection requires a coordinated effort in a lean, energy-efficient manner. Chryso exploits the inherent routing tree of data collection protocols to coordinate a parent with its children as shown in Figure 2. Parents and children collaboratively change their communication channel, when detecting external interference. This collaboration of a parent with its children necessitates that the protocol stack provides a fairly stable routing tree.
- We utilize *frequency diversity* for robustness against external interference. This frequency diversity has its price: neighboring nodes may operate on different channels, hampering neighbor detection. Chryso includes an efficient neighborhood discovery mechanism for bootstrapping and joining of sensor nodes customized for multi-channel operation.

We specifically focus on the class of low-power listening protocols that use acknowledgements. Hence, asymmetric interference cannot be distinguished from symmetric interference as both result in retransmissions. Note that Chryso specifically concerns robustness against external interference. While some of its concepts also directly apply to *internal interference*, we leave the mitigation of internal interference, funneling effects and throughput optimization as future work. As such we focus in this work, on low data rate data collection applications.

Figure 1 shows the design of Chryso: It comprises several components integrated into an existing protocol stack, which will be explained below.

B. Channel Selection

The fundamental idea of Chryso is to (logically) organize nodes of a data collection tree into parent-children groups as shown in Figure 2. Each parent-children group communicates on a selected channel. Individual parent-children groups may operate on different channels. Each node operates in two different roles: as a parent receiving packets from its children on its *inchannel* and as a child sending packets to its parent

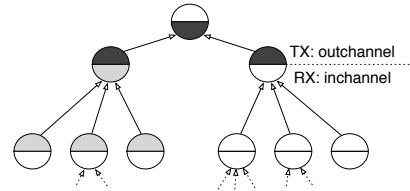


Fig. 2. Parent-children groups each using a single frequency for communication. Each node listens on its inchannel and sends on its outchannel.

on its *outchannel*. Note that the inchannel and the outchannel of a node may be the same. The *channel selection* component controls the channel switching between in- and outchannel as determined by the *Chryso control loops*. It maintains a pre-defined logical list of available channels, so that a parent and its children are consistent with their view on the next channel. It is advisable to space out (logically-adjacent) channels (frequency-wise) in a way that minimizes the possibility of consecutive channels being interfered by the same source. As an example, our implementation uses five 802.15.4 channels; 26, 14, 20, 11 and 22, in the specified order.

C. Channel Quality Monitoring

On the lowest level, Chryso adds a *channel quality monitor* that continuously monitors the links. For children, the monitor collects information about congestion backoffs and send successes. These are periodically piggybacked onto data packets to inform the parent as discussed below. The parent directly reads from the channel quality monitor the number of packets received. The information given by the monitor about the parent and its children is used by protocol-specific policies to determine if interference is present. The overhead of forwarding the information is negligible: for our implementation it is a mere 5 bytes.

D. Chryso Control Loops

The core of Chryso is the set of control loops that manages whether a parent-children pair should stay on the same channel or switch to a new channel. The *inner loop* is responsible for coordinated channel-switching between a parent and its children as soon as external interference is detected. When interference completely blocks any communication, a coordinated channel switch cannot be performed. For this case Chryso uses the so-called *outer loop*, a watchdog mechanism that initiates an autonomous channel switch.

1) *Inner loop*: Figure 3 shows the working of the inner loop on a single node. It differentiates among the behavior of a node in its role as a parent (on the left) and as a child (on the right).

A child node periodically collects data from the channel quality monitor and piggybacks that onto data packets (*send-status*). Additionally, a child node responds to channel switching requests from its parent (*receive-switch*) and subsequently it (logically) increments its outchannel (*out++*).

In periodic intervals of T_{inner} , the parent uses protocol-stack specific policies to determine whether or not the current measured channel quality indicates external interference (*channel-bad?* or *channel-good?*). In our implementation,

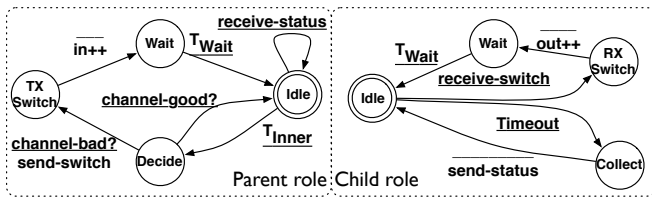


Fig. 3. State machine for parent (left) and child role (right) of the *inner loop*. Transitions are labeled with conditions including timers and action labels (above the line, or below respectively).

the parent node computes an average over the backoff values and checks whether it exceeds a predetermined threshold. If this is the case, the parent initiates a coordinated switch by setting a channel switching indicator on subsequent ACK messages (*send-switch*). Having notified all its children, the parent node switches to the next (logical) inchannel (*in++*).

2) *Outer loop*: The outer loop functions as a watchdog for cases of severe interference that disrupt the operation of the inner loop. Hence, the outer loop executes over a larger time interval than the inner loop. Here a node (in child and parent role) decides independently to switch channels based on the protocol-specific policies. These decisions involve no explicit coordination between nodes; thus, channel switches triggered by the outer loop are instantaneous. However, in general both the parent and its children detect such interference and all perform the autonomous switch. In turn, the parent-children group meets on the consecutive channel.

For detecting severe interference, a child node monitors the number of failed transmissions (messages that were never sent) and switches its outchannel if the failure ratio exceeds a threshold. Likewise, the parent switches to the next inchannel when the ratio of received packets drops below a pre-set value. As these ratios depend on the specific protocol stack and the application, the corresponding thresholds are defined by the protocol-specific policies.

3) *Wait time*: Chryso incorporates a *Wait* state for both parent and child nodes, immediately after a re-connection on the new channel. Since transmissions failed before the switch, nodes are expected to have queued several packets causing a momentary increase in message transmissions after switching to the next channel. This leads to increased contention and could be mistaken for further external interference. Therefore Chryso suspends channel switching decisions during such transient periods by implementing a timeout T_{wait} . Note that the wait time does not affect the reconnection latency; it only delays decisions about switching to yet another channel until statistics have stabilized again.

E. Neighborhood Discovery

Neighborhood discovery, i. e., finding a (new) parent, is vital in practical deployments. As an example, a routing switch may be necessary when the link quality to the current parent deteriorates and cannot be remedied by a channel switch. Because neighboring nodes operate on different channels, topology information available at a child node is only partial, and also subject to change, as neighbors may switch channels during network operation. Hence, Chryso employs a special

neighborhood discovery phase, the so-called *scan mode*, to find a new parent. Since discovering neighboring nodes in a multi-channel environment incurs additional overhead on processing and energy consumption, the scan mode is only triggered on demand.

The scan mode relies on beacons containing routing information. Routing beacons are sent on the *inchannel*. A node performing neighborhood discovery scans through the list of available channels to search for a new parent. A channel scan typically involves the node listening for beacons from all potential parents in the neighborhood. Having collected a set of parent candidates, the node chooses the best one and re-starts channel monitoring. In Section III-B4, we detail on our efficient implementation of the scan mode using a special kind of routing beacons, so-called announcement blips.

III. IMPLEMENTATION

To study the feasibility of our approach, we implemented Chryso in Contiki [4]. In the following we detail on generic implementation aspects, including the tuning of the timing parameters of Chryso, and describe the protocol-specific policies for Contiki.

A. Protocol Stack

Contiki uses by default a low-power listening (LPL) MAC, similar to X-MAC [3], using strobed preambles and duty-cycles the radio with a wakeup interval T_w . There are two major differences to X-MAC in the implementation on Contiki: (1) The given protocol does not use long broadcast messages with length T_w , but uses short strobe packets, so-called announcement blips, that are sent periodically with a random offset so that neighbors receive it with a given probability. (2) The MAC protocol also includes an optimization that uses time synchronization between a sender node and its receivers. In the following we use this optimization that closely resembles a synchronized LPL, in particular WiseMAC [7], wherein a sender node starts transmission just before the the intended receiver wakes up.

We use the *Collect* routing protocol. The routing tree is formed and maintained by announcement blips [5] that are broadcast by every node. Each blip contains information about the node's address and its routing gradient to the sink. Based on announcements, each node builds a neighbor table that is used by the *Forwarding Engine* (cf. Figure 1) to select a parent. By default, the *Collect* protocol uses a naive policy for forwarding packets; for every packet, a node selects its neighbor with the best routing metric to the sink. Chryso works best on a stable routing tree. To this end, we improve routing stability by leveraging a thresholding policy based on expected transmission count (ETX) for performing a routing switch as suggested in CTP [8]. Nodes switch their routing parent only if the newly discovered ETX is below the currently perceived best ETX by a factor of 1.5.

For testing Chryso, we use a sense-and-send application that generates data every T_{data} on every node except the sink node. The relevant parameters of the implementation are shown in Table I.

TABLE I
PARAMETERS OF APPLICATION AND PROTOCOL STACK

Stack	Parameter	Notation	Value
MAC	Wakeup Interval	T_w	250ms
	Retransmissions	N_{MAC}^{retx}	2
Collect (CT)	Retransmissions	N_{CT}^{retx}	4
Application	Sampling interval	T_{data}	32s

B. Chryso Implementation

In the following we detail on the implementation of the Chryso control loops and their coupling with the channel quality monitoring and neighbor discovery. Our implementation relies on a set of timing values that are related to protocol-specific parameters. Typically, the per-packet generation interval T_{data} and the wakeup interval T_w are important parameters that determine respectively the network traffic and available bandwidth. Our focus is on low data-rate applications, in which $T_{data} \gg T_w$. While in general Chryso's timing parameters are dependent on T_w , in the considered low data rate applications the timing values in Chryso are largely derived from T_{data} . Hence, they can be determined at compile time. Most importantly, the temporal notion of the control loop evaluation is captured in the two main parameters T_{inner} and T_{outer} .

T_{inner} is the inner loop interval over which the parent evaluates the channel quality (cf. Figure 3). We prefer congestion backoffs as a measure of interference, chiefly because they abstract away from the underlying details of signal strength measurements as conducted in [16], and are freely available on most MAC implementations. For convergecast networks, nodes transmit packets over an interval that is equal to (for leaf nodes) or less than (for forwarding nodes) the data sampling interval. Therefore, the backoffs observed by several child nodes over a window of the data sampling interval T_{data} average to a value that is typically consistent with individual measurements. Hence, we set $T_{inner} := T_{data}$.

T_{outer} is the interval after which the outer loop decides on an autonomous channel switch. The outer loop operates over a much larger interval. This is because autonomous channel switches should only be performed when the inner loop fails to trigger a channel switch for an extended duration. We obtain good results with T_{outer} set to six times T_{inner} . With increasing values of T_{data} , the channel monitoring interval gets proportionally spaced out in time, which will increase the *absolute* latency in channel switching and reconnection. However, it must be noted that the traffic flowing to the sink reduces by a fixed proportion, such that the resulting performance would not be affected either by scaling up (or down) T_{data} .

1) *Inner loop*: A parent node collects backoffs of its children measured on an average-per-packet basis. It computes the harmonic mean² over the child nodes' backoffs during an interval T_{inner} . If the computed backoff exceeds a threshold $r_{back,max}$, the parent triggers a co-ordinated channel switch by

²The harmonic mean is typically used to express average of *rates*, and has been known to mitigate the impact of large outliers.

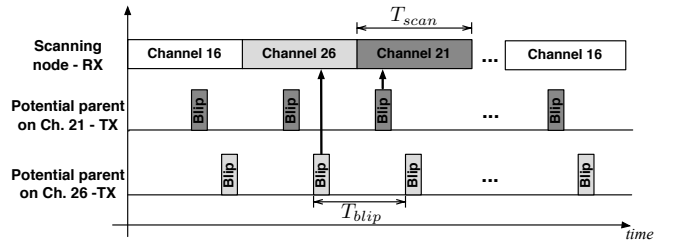


Fig. 4. Channel scanning procedure: A scanning node sweeps through its logical list of channels (here 16, 26, 21, ...), listening for blip transmissions of neighboring nodes

indicating so on subsequent ACK messages. The parent stays on the current inchannel for T_{data} before moving on to the next inchannel in order to notify all of its children and allow the children to follow the switching decision. Correspondingly, after receiving a switch indication, a child node waits for T_{data} before switching its outchannel.

2) *Outer loop*: The outer loop is evaluated at the end of every interval T_{outer} , by both child and parent nodes. A child node monitors the number of successful and failed transmission attempts over T_{outer} . A transmission is only considered successful when a node manages to send the data packet after having received a strobe acknowledgement from its receiver. If the number of failed transmissions exceeds a certain fraction $r_{tx,max}$ of the total transmission attempts, then the child node switches its outchannel instantly and autonomously. Likewise, a parent node keeps a record of the number of packets received over T_{outer} . If the fraction of received over expected packets falls below a threshold $r_{rx,min}$, the parent switches its inchannel instantly and autonomously.

3) *Switching mechanism*: On channel switches, a parent node resets its collected channel statistics. For a child node, both channel switches and routing switches necessitate a reset of its collected channel quality statistics. Additionally, the child node also clears its routing entries.

Immediately following a channel switch, a child node starts a watchdog timer. The watchdog initiates the scan mode (cf. Section III-B4) if the node does not find a parent available on the new channel. The watchdog is set to T_{outer} , thereby enabling the child node to detect a re-connection problem.

Once a connection is established, a wait interval T_{wait} is used before new channel quality evaluations are performed, as nodes experience transient effects of channel switching. In our implementation, we set T_{wait} to T_{outer} , i.e., the nodes skip one evaluation of both the inner and outer control loops.

4) *Scan mode*: The *scan mode* is invoked at network bootstrap or whenever a sensor node loses connectivity to its parent and has no (other) entry in its neighbor table, e.g. following a channel switch by the outer loop. It replaces the neighbor discovery and management of the original stack.

During scan mode, a sensor node sweeps across the list of channels. The previously used outchannel is blacklisted for that period. A node listens for announcement blips for a duration of T_{scan} per channel, as shown in Figure 4. Having received one blip, the node continues to sweep across the

channels one additional time, before deciding on a routing parent. This allows the scanning node to (potentially) take a better routing decision. We set T_{scan} to the same order of magnitude as T_{blip} , so that a scanning node is assured of a blip on a channel, if there is a node on it that can receive packets.

5) *Bootstrap*: A node in scan mode does not send announcement blips. This is used for *network bootstrap*: All nodes are in scan mode and merely listen for blips. Only the sink sends blips to announce a routing gradient of 0. Sensor nodes recursively connect to the sink using the regular scan mode procedure.

In order to determine the bootstrap latency using our novel multi-channel scan mode, we performed experiments with and without Chryso. These tests were carried out on a four hop network comprising 26 Tmote sky nodes, including the sink (cf. Testbed-Zurich in Section IV-A). Without Chryso the median latency of the time to bootstrap for the sensor nodes is 130s and the 99%-percentile is 717s. With Chryso the median latency is 164s and the 99%-percentile is 547s. The comparatively larger median latency of Chryso is attributed to the *additional* scan time that a node spends before connecting to the network. Nevertheless, Chryso produces fewer outliers on bootstrapping latency, thereby showing the effectiveness of our scan mode for multi-channel neighbor discovery.

6) *Blips*: We prefer blips for neighbor discovery over long preamble transmissions, the original broadcast mechanism of X-MAC and other LPL protocols. This is because long preamble transmissions cause considerable channel contention due to their excessive strobing. In practice this results not only in an increased connection latency, but also in a sub-optimal routing tree. Moreover, blips allow for fast neighbor discovery in a multi-channel environment. Finally, we note that receive and send energy of typical 802.15.4 radios is comparable, e. g. for the the Tmote and the Iris sensor nodes. As a result, we advocate to replace LPL broadcasts with blips for multi-channel protocols on 802.15.4 radios.

Blips are only sent on the inchannel of a node. In addition to announcing the routing metrics using blips, Chryso also piggybacks routing metrics onto software acknowledgement messages. This alleviates a fundamental problem of multi-channel protocols: *channel deafness*, i. e., not hearing a packet on a different channel. The piggybacking feature involves minimal overhead on message payload, i. e., 2 bytes per ACK message.

C. Protocol-specific Policies

To identify external interference, protocol-specific policies need to be determined, i. e., few threshold parameters need to be adapted. While the threshold values vary from one protocol stack implementation to another, the process of determining these values remains the same. In the following, we describe the reasoning for determining the thresholds using our specific implementation, and support it with data from controlled tests on sensor nodes. In particular, to verify our reasoning, we performed a series of controlled jamming experiments with

TABLE II

OUTER LOOP EXPLORATION FOR $r_{tx,max}$. FN AND FP IN PERCENT. LATENCY (LAT.) IN SECONDS. NODES WERE SUBJECTED TO EXTREME JAMMING TO DETERMINE LATENCY AND FN. FPs WERE DETERMINED WITHOUT ANY JAMMER PRESENT

$r_{tx,max}$	0.6			0.7			0.75			0.8		
Metric	FN	FP	Lat.	FN	FP	Lat.	FN	FP	Lat.	FN	FP	Lat.
Result	0	2	163s	0	1	187s	0	0	187s	0	0	379s

Tmote sky nodes. We use a setup of 1 parent and 16 child nodes. The jamming impact is varied from mild to extreme interference (as described in Section IV-B). We have different performance metrics for evaluating the policies:

- *False negatives* (FN), i. e., nodes failing to switch although they are interfered, should be low.
- *False positives* (FP), i. e., nodes switching channels although they are not interfered, should be low.
- The *latency* of a switching decision. The longer the latency, the larger the impact of the interference on the data collection process.

False positives are measured using a non-interfered set of nodes using channel 26, a relatively clean 802.15.4 channel.

1) *Inner loop policy*: The inner loop must detect channel interference within a short time, and co-ordinate a channel switch between a parent and its children. The inner loop should neither enforce unnecessary channel switches nor fail to react to external interference. The sensitivity of the inner loop is controlled by a backoff threshold $r_{back,max}$ that represents the maximum average backoff value per packet. $r_{back,max}$ mostly depends upon the MAC protocol implementation. With X-MAC, nodes back off, i. e., abort a transmission attempt, as soon as they receive a strobe or detect any interfering signal in their neighborhood. Therefore, a backoff threshold of 1 suffices to indicate whether the current channel is interfered. Our experiments show that in all cases neither false positives nor false negatives occurred. This demonstrates that Chryso's inner-loop policy is robust, and effectively detects external interference.

2) *Outer loop policy*: The outer loop serves as a fallback mechanism for nodes to switch channels in scenarios where the inner loop fails to trigger. Such cases typically occur when a child node misses the channel switching indication from the parent. They may also occur, although infrequently, when only a local set of child nodes experience interference, such that the parent notices nothing wrong with the channel. As with the inner loop, the thresholds for the outer loop need to be determined to minimize the occurrences of false positives and false negatives. Since the outer loop decisions are taken autonomously by both parent and children, the channel switch conditions are implemented independently for both roles.

A child node switches its outchannel if the following condition holds:

$$\frac{N_{TX_failed}}{N_{TX_failed} + N_{TX_succeeded}} > r_{tx,max} \quad (1)$$

where N_{TX_failed} and $N_{TX_succeeded}$ are the number of failed and successful transmissions, respectively. As the outer

TABLE III
MEMORY CONSUMPTION (BYTES) FOR CHRYSO AND THE ORIGINAL
SINGLE-CHANNEL (SC) PROTOCOL STACK.

	Chryso			SC	
	Control loops	MAC	Routing	MAC	Routing
Program	3342	2956	2462	2254	1810
Data	148	122	398	118	358

loop is mainly targeted for handling severe cases of interference, we conducted short experiments with *Extreme Jamming* to evaluate different values for $r_{tx,max}$. The results shown in Table II suggest that $r_{tx,max} = 0.75$ results in a short channel switch latency without causing either false positives or false negatives. Hence, we adopt $r_{tx,max} = 0.75$ in the following.

A parent node keeps track of the number of messages received N_{RX} and switches its inchannel if the following condition holds:

$$N_{RX} < r_{rx,min} \cdot \frac{T_{outer}}{T_{data}} \quad (2)$$

The ratio $\frac{T_{outer}}{T_{data}}$ gives the number of packets generated by a single leaf node. In the case of extreme interference on the parent's side the number of incoming packets would immediately drop close to zero. When only the children are affected, the incoming rate also goes to zero as an effect of them switching autonomously to another channel. In both cases, we set $r_{rx,min}$ to 0.5, so that the parent assumes the current channel noisy when it receives less than half of the packets that a single child would minimally generate. Such a policy is simple, yet effective in practice for detecting the impact of external interference on packet reception.

D. Memory Footprint

Sensor nodes are typically resource constrained, for example, Tmote sky motes are equipped with a program flash memory of 48 KB and a data memory of 10 KB. Therefore applications need to be optimized for memory consumption. Table III shows the memory footprint of Chryso, broken down to component level (control loops, MAC, and routing), as well as the size of their counterparts in Contiki's original, single-channel protocol stack. In total, Chryso consumes around 4.7 KB of additional program memory and 192 extra bytes of RAM.

IV. EVALUATION

In the following we show results from several experiments across two testbeds and compare Chryso against (i) the protocol stack with the channel control loops and scan mode disabled; that is, a single-channel solution, and (ii) a channel-hopping protocol.

A. Experimental Setup

For our test we use Tmote sky nodes that feature the CC2420 transceiver, a 802.15.4 radio. The tests were conducted on two real-world testbeds: Testbed-Delft at TU Delft comprises 17 nodes and is deployed over several rooms on an office floor. We set the transmission power levels such that all nodes are within communication range. Testbed-Zurich at ETH Zurich features 25 sensor nodes that are also deployed

on an office floor and usually create a four-hop network. Every node (except the sink) generates packets every T_{data} . For controlled jamming, we use the technique described in [1]: The jammer transmits an unmodulated signal on a channel, thereby interfering any concurrent communication within range.

We evaluate Chryso using two performance metrics: data yield and energy consumption. Since we are interested in ensuring robustness to interference, we mainly focus on the data yield at the sink: the percentage of uniquely generated packets that are received at the sink. As data collection WSNs typically rely on battery-operated sensor nodes, energy use is a prime concern. In our evaluation we focus on the radio duty cycle as the radio is the major consumer of energy on typical sensor node platforms. We use Contiki's energest module [6] and present the radio duty cycle averaged over all sensor nodes. We present the results in four subsections. First, we test Chryso using a controlled jammer to verify the functionality of the control loops. Second, we induce Wi-Fi traffic in the proximity of the network, and observe how Chryso reacts to a sudden increase in channel activity. Both tests are performed on Testbed-Delft. These first tests show how Chryso is able to cope with interference compared to a single-channel solution. Third, we assess the stability and scalability of our channel switching protocol on a multi-hop network on Testbed-Zurich with and without controlled jamming. Finally, we compare Chryso with a frequency-hopping multi-channel protocol [2] on Testbed-Delft.

B. Controlled Jamming

First, we compare the data yield and duty cycle of Chryso against that of a single-channel solution using controlled jamming. Since we are interested in evaluating the operations of Chryso's control loops, we perform these tests by manually choosing 802.15.4 channel 26 as the channel on which nodes begin communication.³ All experiments run for at least an hour on Testbed-Delft in the presence of a jammer, in our setup a jamming Tmote sky node. We turn on the jammer after the nodes had sufficient time to form a routing tree to the sink ($\approx 15min$). We vary the so-called *jamming impact*, defined as the percentage of the time that the jammer node is active over a given period. This period, a so-called *epoch*, is fixed in our experiments to two minutes. We chose the *epoch* considering that while typical Wi-Fi bursts last much shorter, downloading a large file takes over a few minutes. Note that we additionally performed these tests with much shorter jamming *epochs*, and found similar results. We distinguish four different categories of jamming: *mild*, *moderate*, *extreme* and *full*. For mild jamming, 17% of the epoch the jammer is turned on, for moderate jamming the ratio of jamming is 50% of the epoch, for extreme jamming it is 83%, and for full jamming 100%.

Figure 5 shows the averaged results of the controlled tests, with 5 tests for each jamming impact value. We see that when jamming increases, the data yield for the single-channel stack drops as low as 70%. In contrast, Chryso sustains a data

³Channel 26 is relatively cleaner than the rest and allows for experiments with controlled interference.

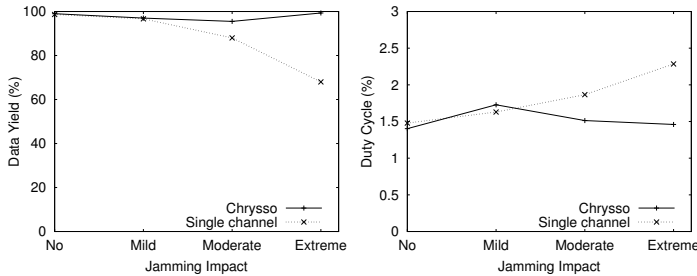


Fig. 5. Results from controlled jamming on data yield (left) and duty cycle (right)

yield of greater than 97%, with merely a momentary loss of packets that results from the channel switching co-ordination. We recorded that in 80% of the cases, the channel switch is initiated via the inner loop. For the rest, only a subset of child nodes were interfered by the jammer. In these cases, the scan mode was invoked after an outer loop switch on the affected child nodes. Note that while Chryosso’s yield is higher, its duty cycle is similar to or (for moderate and extreme jamming) even lower than for the single-channel solution. This is because communication on a noisy channel is very expensive mainly due to re-transmissions by the sender.

We performed detailed tests on the performance of the outer loop control for the case of *full* jamming on Testbed-Delft, with 8 child nodes and a sink, over 20 runs. We observed that both child nodes and the parent switched channels via the outer loop in all the runs, and reconnected on the next channel. The latency incurred in reconnecting parent nodes and their children on a new channel is influenced by the relative phase of the jammer startup within the outer loop interval. Intuitively, if the jammer begins interfering right before the outer loop evaluation, then it is unlikely that the affected nodes will switch channels at the current evaluation. For the worst case, in which both parent and child nodes switch channels at the next firing of the outer loop, the resulting latency to reconnect is bounded by $2 \times T_{outer}$. This places an upper bound on performance degradation in the case of *full* jamming, which we verified in our experimental runs.

C. Wi-Fi Interference

In the second set of tests, we compare the data yield and duty cycle of Chryosso against a single-channel solution under real Wi-Fi interference. Testbed-Delft is located within the proximity of 3 Wi-Fi access points on the same floor. These access points occupy the Wi-Fi frequencies 1, 6 and 11, and exhibit intermittent bursts of activity throughout the day and hence provide a realistic, uncontrolled setting. These tests were done during day time when there was *moderate* amount of activity in the above mentioned Wi-Fi bands. Additionally, we wanted to test how Chryosso would perform in comparison to the single-channel solution when there is a *high* activity on the Wi-Fi network. To this end, we initiated two file downloads of 693 MB each, while the data collection on the 802.15.4 channel was in progress. Depending upon the number of user requests over the Wi-Fi, such a download would last in the order of minutes. We choose an 802.15.4 channel that is close

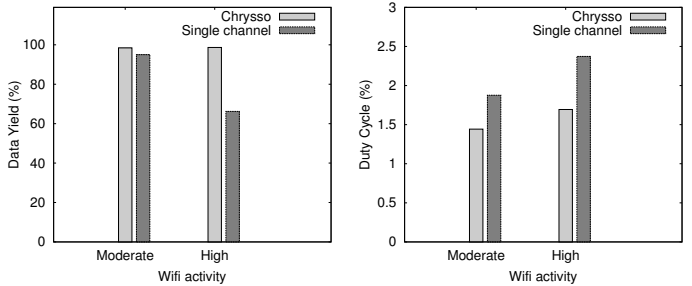


Fig. 6. Results from moderate and high Wi-Fi interference on data yield (left) and duty cycle (right)

to the Wi-Fi hotspot to experiment with a realistic jamming impact. Note that this represents a worst-case scenario in that the external interference is on the same channel as the sensor nodes operate.

Figure 6 shows averaged results from three experimental runs, each lasting for over an hour for both a moderate and a high amount of Wi-Fi activity. For moderate Wi-Fi activity, Chryosso achieves a marginally better data yield than the single-channel protocol, but consumes less energy. In this case, nodes running Chryosso switched their outchannels at most once, showing a stable operation of the control loops.

Even under high Wi-Fi activity, Chryosso sustains a data yield above 98%. Most of the nodes running Chryosso switched their outchannels only once, via the inner loop. This indicates not only a stable operation of the control loops, but also the ability of Chryosso to avoid interference effects on the spot. In contrast, the single-channel solution suffers from a poor data yield (66%) and increased energy consumption under high Wi-Fi activity. For both moderate and high activity, our logs reveal that the energy overhead of the single channel stack is the result of the re-transmissions caused by the Wi-Fi interference.

D. Multi-hop Setup

Additionally, we tested Chryosso on Testbed-Zurich concerning a stable operation over a multi-hop network. These experiments were conducted during the night, so we attribute any performance degradation to unstable links rather than Wi-Fi activity. Nevertheless, it must be noted that Testbed-Zurich has challenging conditions due to contention caused by its dense deployment and multiple Wi-Fi access points in the building. Table IV shows results from 3 runs, each lasting 10 hours. While the single-channel protocol stack achieves 86% data yield under normal operation, Chryosso performs better with a data yield over 92%. Chryosso also outperforms the single channel solution on the issue of routing stability; an average of 3 routing switches per node were observed, in contrast to 12 routing switches per node when the network operates on a single channel. This highlights the ability of Chryosso to maintain a stable routing tree in the majority of cases, effectively only performing routing switches in the face of really unreliable communication.

To see how Chryosso performs under the effect of external interference, we also selected a node near the sink as a jammer. We jammed the network (full jamming) for the duration of the experiment, after a bootstrap phase as described before.

TABLE IV

EVALUATION ON TESTBED-ZURICH. DATA YIELD AVERAGED OVER THREE EXPERIMENTS. ROUTING SWITCHES AVERAGED PER NODE, AND OVER THREE EXPERIMENTS.

<i>Jamming Type</i>	<i>Metric</i>	<i>Single channel</i>	<i>Chryso</i>
No Jamming	Data Yield	86.3%	92.7%
	Routing switches	11.9	3.1
Full Jamming	Data Yield	65%	91%

We performed 2 runs for each 10 hours. Table IV shows that the data yield for the single-channel stack drops below 70%, while Chryso maintains a data yield greater than 90%. It must be noted that we ran the single channel implementation on channel 26, a relatively cleaner frequency. This shows that Chryso not only offers robustness against external interference, but also provides frequency diversity that guarantees a better performance than a single channel solution.

To assess the fidelity of the coordinated channel switching in a realistic network setup, we analyzed the data from the experimental runs without jamming for incidences where a parent initiated a channel switch via the inner loop. For each incidence, we studied the number of child channel switches that resulted out of coordinated (inner loop) switching, autonomous switching (outer loop) and channel scans. We also looked at possible routing switches for a child node. Table V shows that 85.7% (349 switches out of 407 in total) of the channel switches for the child nodes are inner loop switches, coordinated by the parent node. In 8.6% (35 out of 407) of the cases, child nodes failed to receive the channel switch indication from their parent, and performed an outer loop switch. In contrast, only 4.7% of the channel switches resulted out of channel scans. This underlines the reason for Chryso being energy efficient: Channel switching is typically coordinated and thus has minimal energy overhead.

E. Comparison with a Frequency-Hopping Protocol

Finally, we performed a comparison with a frequency hopping protocol, MuChMAC [2]. We use the protocol implementation from the original authors with eight 802.15.4 channels and 100ms slots. Unfortunately MuChMAC experienced time synchronization, bootstrap, and neighbor management issues on Testbed-Delft. Note that these are typical issues for this class of protocols. In order to allow for a fair comparison, we used a simplified static topology of eight nodes, where MuChMAC performed flawlessly. We evaluated both MuChMAC and Chryso during daytime with Wi-Fi activity predominantly on Wi-Fi channels 1, 6 and 11 for the course of 12 hours.

While Chryso achieves a data yield of 98% by successfully evading interfered channels, MuChMAC only achieved 84% data yield. This is to be expected: a frequency hopping protocol is still affected when it hops to a channel that is interfered. In fact, when one out of 8 channels is blocked, we would expect under perfect conditions MuChMAC's data yield to be slightly higher at 87.5%. Nevertheless, this indicates that while frequency hopping ensures some form of robustness to external interference, it struggles with persistent channel interference. Hence, some form of blacklisting may be needed.

TABLE V

INNER LOOP PERFORMANCE FROM TESTBED-ZURICH. THE VALUES DENOTE THE PERCENTAGE (%) OF THE TOTAL NUMBER OF CHANNEL SWITCHES OBSERVED.

Inner loop	Outer loop	Channel scans	Routing change
85.7	8.6	4.7	1.0

V. RELATED WORK

Multi-channel protocols are used for mitigating two different sources of interference: internal and external interference. In this work, we focus on related work addressing external interference. Protocols that mitigate internal interference are typically not designed to mitigate external interference, since: (1) They rely on offline approaches to channel allocation [9], [21] and hence do not help with external interference that is not predictable and varies over time. (2) They apply channel assignments to independent routing paths or trees [14], [19] and hence are not flexible enough to cope with the localized nature of external interference.

Previous work has proposed channel-hopping protocols [2], [12], [17], [20]; however, these protocols necessitate time synchronization between nodes. Chryso does not require time synchronization and focuses on asynchronous, low-power listening MAC protocols. Nevertheless, we have shown a comparison to a channel-hopping protocol particularly designed for low-power sensor nodes [2] in Section IV.

Our work is similar to the multi-channel protocol by Le et al. [13] that is specifically designed for data *aggregation* in WSNs. This is, however, subtly different from our focus on data *collection*, where packets are not processed in-network, but transmitted to the sink in their entirety. In particular, Le et al. propose aggregation-specific policies that cannot be applied to data collection. In contrast to Chryso, the work does not focus on robustness to external interference, but rather on throughput maximization. As such, it focuses on switching complete subtrees of an aggregation tree. This is a considerable overhead for deep trees and not necessary for localized external interference. Voigt et al. [18] developed a multi-channel feature that effects an adaptive channel switch when nodes find the current channel noisy. While the basic control mechanism is similar to Chryso, the channel switching policy is very simplistic: a parent-children group switch channels when the parent has not received a data packet. Furthermore, it does not consider the case of localized interference, which may require a routing solution. The work by Kang et al. [11] is specifically targeted towards resolving external interference effects in ZigBee WPANs by the use of multiple radio channels. However, the authors assume a static routing topology, and do not account for dynamics at the network layer that could complicate channel allocation strategies. Every node within a cluster operates at the same channel, in contrast to our design that allows frequency diversity within a cluster of nodes. Furthermore, the multichannel protocol proposed in [11] has additional communication overhead such as *beacons* and *test frames* to detect external interference, which constitutes a significant overhead for duty-cycling radios.

Interestingly, Liang et al. [15] independently suggested a multi-channel protocol architecture that resembles the design of Chryso. Although our work focuses on data collection WSNs, the *Channel Allocation* and *Channel Synchronization* components proposed by Liang et al. closely match Chryso's *control loops* and *channel selection* in their functionality. While their proposed architecture was only conceptual at the time of publication, we actually have outlined the mechanisms for channel monitoring and control and also provide policies and how to derive them. In this regard, their work supports our approach to mitigate external interference and our research efforts for a more general use of Chryso.

Finally, we note that most multi-channel solutions have been evaluated on a static routing topology. This fails to account for long-term link quality variations that necessitate routing changes and hence require a form of neighborhood discovery. An exception is Liang et al. [14], wherein nodes could connect to different routing trees on (possibly) different channels. However, their work concentrates on multiple data collection trees (using multiple sinks), each on an individual frequency, while Chryso uses multiple channels within one data collection tree. We based our neighborhood discovery scheme upon this work.

VI. CONCLUSIONS

This paper addresses the issue of external interference hampering radio communications, such as Wi-Fi traffic affecting low-power 802.15.4 radios. To this end, we presented Chryso, a multi-channel protocol extension, that leverages the channel diversity of typical sensor node radios. Chryso mitigates the effects of external interference by switching (only) the affected nodes to a new set of channels. It is specifically tailored to data collection applications, and allocates channels for individual parent-children groups with the parent coordinating a channel switch upon detecting interference. For efficiency nodes normally operate on two channels only, one for incoming and one for outgoing traffic. We also present a novel scanning procedure probing all channels that can be used either at bootstrap or when losing contact with the parent.

As a proof of concept we integrated Chryso in the Contiki protocol stack and extensively tested this implementation on two different testbeds. The results show the effectiveness of Chryso in the face of Wi-Fi interference and (controlled) jamming. A comparison with MuChMAC, a state-of-the-art channel hopping protocol, confirms that Chryso outperforms such solutions for mitigating persistent external interference as it avoids the affected channels completely.

VII. ACKNOWLEDGMENTS

Venkatraman Iyer is supported by the ALwEN project, a Point-One project funded by SenterNovem. We would also like to thank the authors of MuChMAC for providing their source code and technical advice on the experiments.

REFERENCES

- [1] C. Boano, K. Römer, Z. He, T. Voigt, M. Zúñiga, and A. Willig, "Generation of controllable radio interference for protocol testing in wireless sensor networks," in *SenSys '09*, 2009, pp. 301–302.
- [2] J. Borms, K. Steenhaut, and B. Lemmens, "Low-overhead dynamic multi-channel MAC for wireless sensor networks," in *EWSN '10*. Springer, 2010, pp. 81–96.
- [3] M. Buettner, G. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *SenSys '06*, 2006, pp. 307–320.
- [4] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *29th IEEE Int. Conference on Local Computer Networks (LCN'04)*, vol. 462, 2004, pp. 455–462.
- [5] A. Dunkels, L. Mottola, N. Tsiftes, F. Osterlind, J. Eriksson, and N. Finne, "The Announcement Layer: Beacon Coordination for the SensorNet Stack," in *EWSN '11*. Springer, 2011, pp. 211–226.
- [6] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *SenSys '07*, 2007, pp. 28–32.
- [7] A. El-Hoiydi and J. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," in *ISCC 2004*, 2004, pp. 244–251.
- [8] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *SenSys '09*, 2009, pp. 1–14.
- [9] O. Incel, L. van Hoesel, P. Jansen, and P. Havinga, "MC-LMAC: A multi-channel MAC protocol for wireless sensor networks," *Ad Hoc Networks*, pp. 73–94, 2011.
- [10] V. Iyer, M. Woehrle, and K. Langendoen, "Chamaeleon - exploiting multiple channels to mitigate interference," in *7th Int. Workshop on Networked Sensing Systems (INSS 2010)*, 2010, pp. 65–68.
- [11] M. Kang, J. Chong, H. Hyun, S. Kim, B. Jung, and D. Sung, "Adaptive interference-aware multi-channel clustering algorithm in a Zigbee network in the presence of WLAN interference," in *2nd Int. Symposium on Wireless Pervasive Computing (ISWPC'07)*, 2007.
- [12] Y. Kim, H. Shin, and H. Cha, "Y-MAC: An energy-efficient multi-channel MAC protocol for dense wireless sensor networks," in *IPSN '08*, 2008, pp. 53–63.
- [13] H. K. Le, D. Henriksson, and T. Abdelzaher, "A control theory approach to throughput optimization in multi-channel collection sensor networks," in *IPSN '07*, 2007, pp. 31–40.
- [14] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao, "RACNet: a high-fidelity data center sensing network," in *SenSys '09*, New York, NY, USA, 2009, pp. 15–28.
- [15] C. Liang and A. Terzis, "Rethinking multi-channel protocols in wireless sensor networks," in *HotEmnets '10*, 2010.
- [16] R. Musaloiu-E and A. Terzis, "Minimising the effect of wifi interference in 802.15.4 wireless sensor networks," *International Journal of Sensor Networks*, vol. 3, no. 1, pp. 43–54, 2008.
- [17] H. So, W. Walrand, and J. Mo, "McMAC: A parallel rendezvous multi-channel MAC protocol," in *WCNC 2007*, 2007, pp. 334–339.
- [18] T. Voigt, F. Osterlind, and A. Dunkels, "Improving sensor network robustness with multi-channel convergecast," in *2nd ERCIM Workshop on e-Mobility*, 2008, pp. 15–24.
- [19] X. Wang, X. Wang, X. Fu, G. Xing, and N. Jha, "Flow-based real-time communication in multi-channel wireless sensor networks," *Wireless Sensor Networks*, pp. 33–52, 2009.
- [20] T. Watteyne, A. Mehta, and K. Pister, "Reliability through frequency diversity: why channel hopping makes sense," in *6th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, 2009, pp. 116–123.
- [21] Y. Wu, M. Keally, G. Zhou, and W. Mao, "Traffic-aware channel assignment in wireless sensor networks," *Wireless Algorithms, Systems, and Applications*, pp. 479–488, 2009.