Constructive Interference in 802.15.4: A Tutorial

Tengfei Chang, Thomas Watteyne, Xavier Vilajosana, Pedro Henrique Gomes

Constructive Interference in 802.15.4: A Tutorial

Tengfei Chang, Thomas Watteyne, Xavier Vilajosana, Pedro Henrique Gomes

***Abstract*—Constructive Interference (CI) can happen when multiple wireless devices send the same frame at the same time. If the time offset between the transmissions is less than 500 ns, a receiver will successfully decode the frame with high probability. CI can be useful for achieving low-latency communication or low-overhead flooding in a multi-hop low-power wireless network. The contribution of this article is three-fold. First, we present the current state-of-the-art CI-based protocols. Second, we provide a detailed hands-on tutorial on how to implement CI-based protocols on TelosB motes, with well documented open-source code. Third, we discuss the issues and challenges of CI-based protocols, and list open issues and research directions. This article is targeted at the level of practicing engineers and advanced researchers and can serve both as a primer on CI technology and a reference to its implementation.**

I. Introduction

IEEE802.15.4 [1] is a standard which defines both the physical and link layers for low-rate wireless personal area networks (LR-WPAN). Numerous low-power wireless industrial technologies build upon it, including Zigbee [2], Z-Wave, WIA-PA, ISA100.11a [3] and WirelessHART [4]. The 2015 revision of IEEE802.15.4 includes the Time Slotted Channel Hopping (TSCH) link-layer mode, targeting deterministic access and industrial-grade reliability. 6TiSCH, a standardization activity at the IETF, integrates TSCH with IPv6 [5], [6], yielding Internet-enabled low-power wireless networks with industrial performance. This combination is seen as a key enabler for the Industrial Internet of Things. Even though many different technologies and protocols are – and will be – exploited to improve the performance of low-power wireless applications, everything indicates that IEEE802.15.4 will remain a major standard in this field.

Just like any wireless communication technology, IEEE802.15.4 is subject to external interference. This is all the more true in the unlicensed spectrum, such as the 2.400–2.485 GHz "Industrial, Scientific and Medical" (ISM) band. In face of this challenge, several techniques have been developed to cope with external interference. Examples include channel hopping, network coding and temporal diversity, all of which exploit some sort of diversity.

In an entirely counter-intuitive manner, however, interference can be beneficial for the network, which occurs when one exploits "constructive interference" (CI) [7]. The idea of CI is simple: if two or more devices send the exact same frame at the exact same time[1], there is no collision, and the receiver perfectly decodes the frame. Better, in some cases, the frame is received at a signal strength higher than if only one device were transmitting.

For a protocol designer, having CI in his/her toolbox changes everything. In the past, he/she had to make absolutely sure concurrent communications happened at different times (TDMA), different frequencies (FDMA) or using different coding schemes (CDMA). CI relaxes those requirements: if multiple devices have the same information to transmit, they can do so provided they send it at exactly the same time.

Constructive interference enables at least 2 very interesting network features.

First, **stateless relaying**, a simplification in networking. Traditionally, a routing protocol operates in the network to elect, for each node, a "next hop" neighbor. When the node has a packet to relay, it sends it to just that neighbor. With CI, a node can broadcast the packet to *all* of its neighbors. And, provided those neighbors repeat the packet at the same time, the information floods the network, eventually reaching the destination. Of course, many optimizations can be added to this uncontrolled flooding (see Section V), but avoiding the need for a rigid routing scheme opens up many possibilities, including "lazy routing" and mobility.

Second, **ultra-low latency**, a service to the application. Inline with stateless relaying, a device can relay packets without having to worry about solving contention to the medium. Without CI, solving contention can mean waiting for the right time slot, or adopting a back-off scheme, all of which takes time. With CI, a device can relay immediately. Assuming short packets, per-hop latencies of 100's of μs can be achieved.

There are, of course, challenges associated with the use of CI. First, from an *implementation* point of view, **maintaining synchronization** is hard, as devices need to transmit within 500 ns (see Section II), a very short time. Even with state-of-the-art TSCH protocol, network-wide sub-μs synchronization is not achievable at a reasonable energy cost. Synchronization hence needs to rely on a local and ephemeral time reference such as the reception timestamp of the last frame. We have crafted the hands-on portion of this article (Section VIII) to go into a deep discussion of how to efficiently implement CI. Section VIII should hence lift this challenge.

Second, from a *protocol design* point of view, achieving **energy efficiency** remains a challenge. If the radios of all

T. Chang and T. Watteyne are with Inria, EVA team, Paris, France.

X. Vilajosana is with the Universitat Oberta de Catalunya (UOC), Wireless Networks (WiNe) Lab, Barcelona, Spain.

P. H. Gomes is with Ericsson Research (Brazil); this work was done while he was a Ph.D. student at the University of Southern California (USC), CA, USA.

[1] To be precise, in IEEE802.15.4, the signals must be sent within 500 ns of one another, as detailed in Section II.

devices remain continuously on, using CI is a solved problem. But, to challenge the energy efficiency of today's state-of-the-art TSCH networks [8], sub-50 μA average current draw must be achieved, which necessarily translates into aggressive radio duty cycling. As discussed in Section IX, there is a real opportunity for combining the scheduled nature of TSCH networks with CI. This is largely not addressed in the literature at this point, leaving energy efficiency as an open problem, especially in large networks.

Third, and probably most importantly, **security** remains a major question mark. Not so much at the application layer, as the data communicated between the source and destination can be end-to-end protected. But while it is traversing the network, all relaying devices need to send the exact same frame, which eliminates the opportunity of using per-link keys or even authenticated tags in which the MAC address of the transmitter is used as part of the link-layer keying material. The simplest approach is to have a single network-wide key for link-layer protection. This is undesirable as it makes device repudiation hard. As discussed in Section IX, security is one of the strong remaining barriers to using CI in an industrial setting.

IEEE802.15.4 TSCH brings low power consumption and high-reliability to industrial applications [9], [10], [11]. 6TiSCH adds the Internet Protocol (IP) to TSCH for industrial IoT. With the RPL routing layer, the end-to-end latency for a TSCH network could be in the order of a few seconds. It meets the general requirements of industrial application for classes 4 and 5 defined in RFC5673 [12] (detailed in Section. IX). But for some applications involving control loops, more stringent requirements may be necessary (classes 2 and 3 in RFC5673). CI brings down the latency to the millisecond level. At the same time, it is possible to completely remove the routing layer and achieve end-to-end communication through flooding. Introducing CI into IEEE802.15.4 could improve its overall performance dramatically.

This article is crafted to be a primer on constructive interference in IEEE802.15.4 networks. It is tailored to the level of practicing engineers and advanced researchers and serves three roles. First, it **introduces CI**, taking the reader through a comprehensive presentation of the state-of-the-art literature on CI-based protocols, which are classified in three different categories. Second, it is a **hands-on tutorial**, explaining in a simple yet comprehensive manner how CI is implemented on a reference platform, the TelosB mote. All the source code used is provided as open-source alongside this article, allowing the reader to repeat the tutorial, and to use any part of the code, including in a commercial setting. Third, it offers a use-case driven **discussion** on the issues and challenges and research directions on the use of CI technique.

The organization of this article reflects these three goals, with three distinct sections focusing on a primer on CI (Sections IV, V and VI), a hands-on tutorial (Section VIII) and issues, challenges and future work (Section IX). While numerous internal references link these three sections together, each of them can be read independently and used as a reference.

Before going through the CI tutorial, it is important to introduce the key concepts that will help the reader understand the underlying phenomena that make CI work, and the reason behind some decision-making during the design of the protocols to be introduced in Sections IV, V and VI. These concepts are introduced in Section II.

II. Basic Concepts of Constructive Interference

Interference is a phenomenon that happens whenever signals overlap in frequency, time and space. If two (or more) different signals interfere with one another, the probability of correctly decoding either is reduced. Another common phenomenon in wireless communication is the *capture effect*. It occurs when one single signal is "captured" and correctly demodulated in spite of other interfering signal(s) being received at the same time. Capture happens if one signal is much stronger (higher receive power) than the others, or if it starts to be demodulated earlier than the others. When the capture effect dominates the reception, interfering signals are not perceived by the receiver. This phenomenon is present in low-power networks and can influence the behavior of protocols.

When exactly the same signal is transmitted by nodes A and B to node C, what C receives is the sum of both signals. This sum can be constructive or destructive, depending on the time offset between the signals. Two sine waves shifted by 90 degrees completely cancel each other out; on the other hand, two in-phase sine waves add up. The same happens with more complex modulation schemes, such as O-QPSK in IEEE802.15.4. For CI to work, the different signals should be sent precisely at the same time. *What is the maximum allowable time offset between the signals for CI to work?*

The answer depends on modulation and bit rate. We focus on IEEE802.15.4 [1] at 2.4 GHz, which uses Offset Quadrature Phase-Shift Keying (O-QPSK) modulation and Direct Sequence Spread Spectrum (DSSS). Transmission can be divided into 3 phases: (i) bit-to-symbol conversion, (ii) symbol-to-chip conversion, (iii) modulation of the chip stream. The data stream is initially grouped into 4-bit *symbols*. Each symbol is then converted into a 32-bit pseudo-random noise (PN) sequence, specified in the IEEE802.15.4 standard. Each binary value in a PN sequence is called a *chip*. The stream of chips is then modulated onto a carrier using O-QPSK with half-sine pulse shaping. The transmission chip rate is 2 Mcps (Megachips per second), which results in a data rate of 250 kbps. The chip duration T_c is 500 ns. Fig. 1 shows that Q-phase chip is delayed by T_c with respect to the I-phase chips.

When receiving a signal, the same happens, in reverse order. The modulated carrier is first converted to chips, which are grouped in PN sequences. A decision-making process in the receiver then converts PN sequences to 4-bit symbols. The redundancy included in the PN sequences is such that the bits can be recovered even when some chips have been corrupted during transmission.

The effect of delayed replicas in MSK baseband signals on bit error rate is a well-understood problem. Even though MSK and O-QPSK signals are similar, the introduction of PN sequences makes the theoretical analysis harder. The work in [13] includes a simulator-based statistical analysis. The

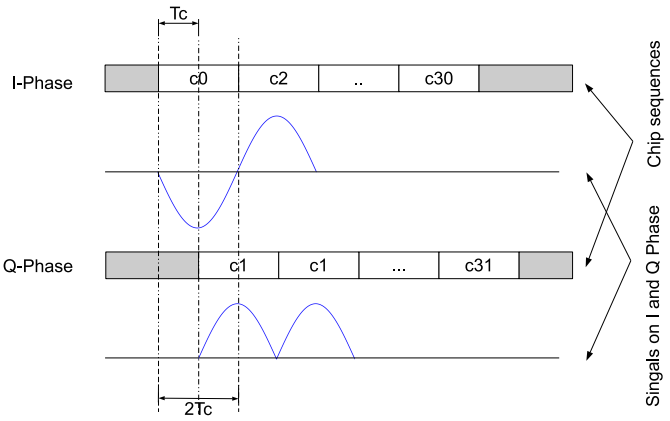


Fig. 1. The Q-phase chip is delayed by T_c with respect to the I-phase, where T_c is the inverse of the chip rate.

result is that, when two frames are offset by 250 ns, the receiver still correctly receives 98% of them. An offset larger than 500 ns reduces that to less than 20%. Similar results appear in [14], in which a mathematical analysis is used to show a frame reception rate of 90% for DSSS signals offset by 150 ns. These results are confirmed by additional experimental and theoretical analysis [15], [16]. The resulting rule-of-thumb is that, when using IEEE802.15.4 at 2.4 GHz, for CI to work with high probability, frames need to be offset by at most 500 ns.

III. Classifying CI Prior Art

The use of CI in (low power) wireless networks has been explored for almost a decade. The first work that applies CI to IEEE802.15.4-based nodes is from 2008, by Dutta et al. [17]. In this early work, hardware-generated acknowledgment frames are used to facilitate anycast communication. In 2010, the same concept is employed as the basis for asynchronous receiver-initiated MAC protocol **A-MAC** [18], which outperformed the state-of-the-art protocols at that time. A larger interest emerged more recently after the publication in 2012 of **Glossy** protocol by Ferrari et al. [13]. **Glossy** provides a thorough investigation on how to achieve optimized network flooding in IEEE802.15.4-based networks using CI. It leverages CI to enable simple network-wide synchronization and implements an efficient network flooding mechanism. Results in [13] on end-to-end delay and reliability triggered further optimizations.

We categorize the main prior-art considering three main aspects:

1) **modeling and optimization** (Section IV) includes seminal papers [18], [13], as well as recent developments [19], [15], [14]. These studies are concerned with understanding how constructive interference takes place, and focus on specific "atomic" issues, such as time drift, capture effect or packet length.
2) **protocol proposals** (Section V) which utilize CI to target a particular problem, such as data prediction [20], bulk transfer [21], neighbor counting and identifying [22], etc. through the proposal of new networking protocols. Most of the studies take a stable implementation such as **Glossy**, and build on top of it. In addition, the *International Conference on Embedded Wireless Systems and Networks (EWSN)* has held a dependability competition in its 2016, 2017 and 2018 editions, in which most competitors have based their implementation on CI.
3) **security challenges** (Section VI) focuses on security aspects and possible attacks on CI-based networks. This is an important and not (yet) solved issue that impacts the adoption of a CI-based solution in industrial settings.

Different types of tutorials and surveys on low-power wireless already exist. Some of them are focused on **use cases**, for example illustrating different mobility management protocols using 6LoWPAN technology [23], discussing the application of WSN in Urban Areas [24] and multiple streaming in WSN [25]. Others are focused on **protocol stack design**, for example, analyzing the performance of multiple resource allocation algorithms at different protocol layers [26] and multichannel routing algorithms [27]. A third type is focused on **specific techniques**, for example utilizing channel bonding to increase the bandwidth of wireless networks [28] and presenting a new fuzzy logic based node localization mechanism [29]. This article is part of the third type of work. We present a overview of CI and a hands-on tutorial.

The requirement of a maximum offset of 500 ns limits the design of the MAC layer in CI-based networks. Flooding is generally the employed method for constructive interference to disseminate the message throughout the network. These features distinguish our tutorial on CI from the general low-power wireless surveys, which usually cover different MAC and routing layer designs. Section IV focuses on the techniques at the physical layer closely related to the core of CI. Since CI-based protocols have a very simple layering approach, we group all upper-layer designs together as protocol proposals in Section V. Most of the protocols are either MAC layer or application-oriented designs. Security in CI is presented as a topic that needs further work (Section VI).

IV. Modeling and Optimizing CI

This section illustrates how CI and related approaches have evolved over time.

A. Optimizing CI

The first work that employs CI in IEEE802.15.4 is **Backcast** [17], presented in 2008. **Backcast** utilizes the hardware automatically acknowledge (ACK) reply features to enable anycast communication. The transmitter node sends a frame and all receivers that match the destination address – which can be unicast, multicast or broadcast – reply with identical ACKs that constructively interfere and are received by the transmitter. **Backcast** allows the sender to know that *at least* one node has correctly received the transmitted packet. **Backcast** is *not* focused on relaying data in a multi-hop manner using CI, but to allow acknowledgment for anycast communication.

Later on, the authors of **Backcast** proposed the receiver-initiated low-power protocol **A-MAC** [18]. The receiver periodically sends probe frames; a probe frame has the "acknowledgment requested" bit set in the IEEE802.15.4 header. Senders that have pending frames to the receiver set their radio to automatically generate ACKs. When a receiver sends a *probe* and hears back an ACK, it knows at least one sender has a frame pending for it. The receiver then leaves its radio on. A back-off mechanism is used to solve collisions when there are multiple transmitters. The receiver can optionally send additional probes to randomly spread the packet retransmissions.

A-MAC was implemented and evaluated on CC2420 radios [30]. The evaluation shows higher packet delivery ratio and lower energy consumption when compared to **LPL** [31] and **RI-MAC** [32], the state-of-the-art transmitter-initiated and receiver-initiated protocols at the time, respectively. An emulator-based evaluation in [18] shows that the packet delivery ratio decreases significantly if the time between two ACKs is greater than 500 ns, the chip period T_c of IEEE802.15.4 (see Section II).

Backcast nor **A-MAC** provide a thorough design analysis on CI. However, this changed with **Glossy** [13]. Among all work related to CI technique, **Glossy** had the highest influence and it is currently the basis of most of CI-based protocol implementations.

Glossy uses CI in the forwarding process of all frames across multi-hop routes. It is based on controlled flooding, where the network activity is decoupled from other application tasks running on the nodes. Simulation results focus on how the time offset between different IEEE802.15.4 frames affects the capacity of correctly receiving them. The results show that, when the timing offset between two current transmissions is less than 500 ns, there is a high probability of achieving CI. These results are in-line with Section II.

The synchronization of nodes in **Glossy** is done implicitly through the flooding process. The source node (flood initiator) adds a 1-byte relay counter c to the frames. Counter c is initialized to 0 and is incremented at each hop. A node only retransmits packets in which counter c is higher than previous copies it has received. The counter has an upper limit to scope the flooding process.

Fig. 2 shows the state machine when a flooding process of **Glossy** is executed. There are mainly four states: **Off**, **Transmit**, **Receive** and **Wait**. Initially, all nodes turn their radio off and wait for the beginning of **Glossy** flooding. When **Glossy** starts, the initiator transmits a packet and changes its state to **Transmit**. The other nodes change their state to *Receive* at the same time. Once a node recognizes the start of a packet reception, its micro-controller starts to read the packet from the receive buffer and transitions to **Receive** state. If the frame reception fails, the node returns to **Wait** state and keeps listening. If the frame reception succeeds, the node modifies the received packet by incrementing the counter C by one and writes it into the transmit buffer. This action corresponds to a transition of **Transmit**. To increase the reliability of **Glossy**, a node transmits the packet N times. After N re-transmissions, the node turns its radio off and waits for the next **Glossy** execution period.

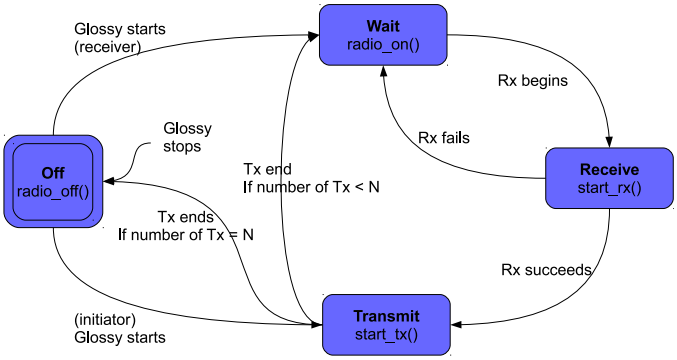


Fig. 2. The **Glossy** [13] state machine.

Glossy can be used as a loose network-wide synchronization mechanism. This relies on the assumption that it takes a deterministic time for a packet to be relayed by a node. That time is expressed in (1), in which T_{sw} is the delay introduced by the software, T_{cal} is the delay due to radio calibration, T_{pr}, T_f, T_l and T_m are the transmission durations for the preamble, SFD, length and MPDU, respectively, and T_d is the delay introduced by the receiver radio at the beginning of frame reception.

$$T_{hop} = T_{sw} + T_{cal} + T_{pr} + T_f + T_l + T_m + T_d \quad (1)$$

All durations are due to the radio operation, except T_{sw} which is introduced by the micro-controller. **Glossy** details how to make T_{sw} deterministic by limiting code execution, choosing a fixed packet size, and calibrating the unstable digitally controlled oscillator which clocks the micro-controller using a stable low-power crystal.

In **Glossy**, network flooding happens periodically. The nodes switch between a flooding period and a period when the micro-controller executes non-deterministic tasks. During that second period, the node switches off its radio to conserve energy. The ratio between the two periods trades off energy conservation and end-to-end communication delay. Nodes only need to be loosely synchronized, as an arbitrarily large guard time can be introduced at the beginning of the flooding period to account for the synchronization inaccuracy between nodes.

Glossy became a seminal work on the exploitation of CI in IEEE802.15.4 networks. Several other works have improved upon it. **Glossy** improvements add features such as frequency hopping, power and topology control, or exploit hardware features to solve issues and limitations of the original proposal.

SCIF [33] is an improvement of **Glossy** for large-scale networks. The challenge with scalability and CI is that the synchronization error accumulates with the number of hops. Some neighbor nodes far from the source node may be off by more than 500 ns because they synchronize across different multi-hop paths. Assuming M is the number of independent paths to the sink, and H the depth of the network in hops, if $M \to \infty$ and $H \to \infty$, the probability of communication at the edge of the network is zero.

SCIF uses **Glossy**-like flooding, with two twists. First, the multi-hop synchronization paths interleave with each other

whenever possible. Second, independent propagation paths with a large number of hops are avoided whenever possible. These two rules, which together form the Spine-based Constructive Interference Flooding (**SCIF**) algorithm, reduces the number of nodes involved in the flooding and, consequently, the time drift.

In practice, **SCIF** consists of two phases: *spine construction* and *flooding*. During the *spine construction* phase, the position of the nodes and their (theoretical) communicating range are used to estimate how a packet would flood the network. If a node is part of two independent branches of the flooding tree, it is not part of the "spine" of the network. During the *flooding* phase, only nodes which belong to the spine relay packets. Simulation results based on real data traces show that, for a network with 4,000 nodes, **SCIF** yields an end-to-end reliability of 94%. In the same setting, regular **Glossy** yields only 30% end-to-end reliability.

TriggerCast [19] selects the links that participate in CI based on detailed observations on the capture effect on CI. The authors observe that when the difference of reception power between two frames is above 3 dB, the *capture effect* dominates and no further improvement is provided by CI. They also observe that, when nodes participating in CI are at very different distances from the receiver, the difference in propagation delay can significantly reduce the packet delivery ratio. Distance differences higher than 40 m may reduce the packet delivery ratio (PDR) by 20%.

Through mathematical derivation based on the signal to noise ratio (SNR) and signal power, the authors propose the following three rules for CI to work:

1) concurrent frames should have a time offset below 500 ns.
2) the time offset between the i-th frame and the frame with the strongest reception power should be less than a threshold which is a function of their relative received power.
3) the ratio between the minimum and maximum SNR of concurrent signals should be larger than a threshold which is a function of the time offset between all concurrent signals.

TriggerCast selects only the links that satisfy those conditions. **TriggerCast** compensates clock skew and radio processing delay, to increase the probability of CI to work.

The results from a real experiment using TelosB motes show that **TriggerCast** achieves 95% reliability when the time offset between frames is below 250 ns, a situation in which **Glossy** only achieves 85% reliability. Further experiments carried out with different levels of PDR links (<5%, 5%-95%, >95%) show that **TriggerCast** increases the RSSI and packet delivery ratio of links with different qualities, and make very weak links become stronger.

B. Modeling CI

Wilhelm et al. [14] extend the mathematical analysis of **TriggerCast** by building a model to predict the outcome of concurrent transmissions. The model takes into account the power ratio (SINR), the timing offset between concurrent frames, the channel coding, the packet content, and the carrier phase. The model considers two different types of chip-to-symbol decoding mechanisms for DSSS signals. Hard Decision Decoding (HDD) considers the highest bit-wise cross-correlation of the chip sequences. Soft Decision Decoding (SDD) adds weights to the bits, improving the quality of demodulation.

Through modeling and simulation, Wilhelm et al. [14] conclude that, if the signal power is greater than the noise and the sum of interference signal power, the capture effect ensures correct reception of concurrent frames at the receiver side. For the case when the concurrent frames are identical, it can be received correctly even when power ratio is negative. Further, under this setting, using SDD yields a PDR radio above 90%, if the time offset is below 150 ns, and with HDD a PDR of 60-80% can be achieved with a maximum drift of 100 ns.

Wilhelm et al. also compare when there is a single interferer at a high power and several interferers at low power. In the latter case, the total power is identical to the former case, and interferers have independent uniformly distributed time offsets. The result is that the multiple-interferer case has a higher impact when the frame sent are different, but no impact when the frames are identical.

Yuan and Hollick [15] further model CI, focusing on one-hop networks and argue that, even in simple cases, CI is hard to be achieved and does not depend only on the timing offset between frames. They use **Glossy** and CC2420 radio chips. Their study is based on an experimental scenario with N motes and one root; motes and root are 1 m apart. The system runs on a 1 s time slot, which implements 2 communication patterns. At time offset 0 ms, the root node polls all motes in a round-robin fashion. This allows the root to continuously monitor the received signal strength from each mote. At time offsets 250 ms, 500 ms, and 750 ms in each 1 s time slot, the root transmits a frame and all motes retransmit it. This results in potentially N concurrent transmissions and allows the root to measure the success rate of CI.

Yuan and Hollick use this experimental setup to model the one-hop delay of **Glossy**. They look at the three main timing components: (i) the duration of frame transmission; (ii) the delay between the end of transmission and the end of reception, introduced by the radio, and (iii) the delay from the end of the packet to the start of the acknowledgment, introduced by the software. Transmission or reception of frames is timestamped using a logic analyzer connected to the Start of Frame Delimiter (SFD) pins of the CC2420 radios.

Results indicate that the duration of packet transmission varies slightly from one frame to another. However, this variation is larger when we compare different nodes. The transmission time of 10-byte packets vary a maximum of about 0.04 μs but do not change between different nodes. On the other hand, the transmission time of 126-byte packets shows much larger variability (closer to 0.10 μs) among different nodes. The time offset between two concurrent frames can be modeled with (2), in which l is the frame length in bytes, and k_1 and k_2 are the drifts of the two radio crystals in ppm (parts per million).

$$D_T = 32(l+12)(\frac{1}{1+k_1\times 10^{-6}} - \frac{1}{1+k_2\times 10^{-6}}) \quad (2)$$

The latency introduced by the radio and the software are measured and modeled as normal random variables: the mean radio latency is 3.79 μs with a variance of 0.0019; the mean software latency is 23.28 μs with a variance of 0.0080. Radio and software latency can be considered independent. Eq. 3 expresses the time offset between concurrent frames sent by two motes, where Y is a normal random variable with $\mu_Y = 0$ μs and $\sigma_Y = 0.0198$.

$$D_\Delta = 32(l+12)(\frac{1}{1+k_1\times 10^{-6}} - \frac{1}{1+k_2\times 10^{-6}}) + Y \quad (3)$$

As for the capture effect, Yuan and Hollick observe that it takes place when the time difference between two concurrent transmissions is less than the duration of the preamble.

Combining the results from constructive interference and capture effect, Yuan and Hollick propose an algorithm that predicts the success rate of concurrent transmissions. The algorithm takes the set of nodes, the transmission start time and signal strength of each node, noise floor and the SER (Symbol Error Rate) vs. SINR model obtained from measurements as the inputs, and returns whether the CI will be successful. In the 2-mote case, the model has 90% accuracy when predicting CI success, and close to 100% when predicting CI failure. In the 6-mote case, the prediction accuracy drops to 70%.

Rao et al. [16] also provide a list of conditions necessary for the success of CI based on real-world experiments and introduce **DIPA** (Destructive Interference-based Power Adaptation) protocol. It considers the fact that power imbalance among concurrent transmitters aids packet reception [19] and proposes an algorithm that dynamically adapts power transmission in order to improve the performance of CI. The proposed heuristic consists of a feedback byte appended to the frames that indicates the success or failure of last transmissions of concurrent packets. The nodes decrease transmit power if the last n consecutive transmissions were successful; they increase the power if a negative feedback is received, and randomly choose a transmit power if negative feedback persists for more than k transmissions. The main *caveat* of this solution is that the CRC has to be calculated in software so that only the feedback byte is changed in the frame and a destructive interference of such byte indicates a negative feedback. **DIPA** is able to yield up to 25% lower bit error rate (BER), and a reduction of around 50% in energy consumption when compared to **Glossy**.

C. Summary

The works listed in this section focus on better understanding why CI works and predicting when/whether it does/does not work. The following are the factors that were found to play important roles to make CI work:

- *Time offset between concurrent IEEE802.15.4 frames must be below 500 ns.* This is empirically and theoretically verified in all works in this section.

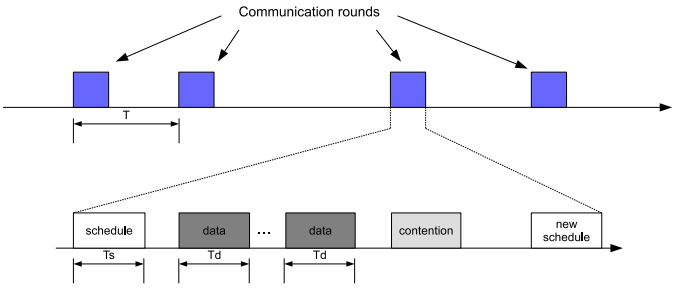


Fig. 3. Slot schedule in **LWB** [34] communication rounds.

- *Having more concurrent nodes or more hops in the network is **bad** for CI.* This is because time synchronization is harder to be achieved with more nodes [33].
- *Having a large power imbalance is **good** for CI.* The difference between the minimum and maximum SNR between concurrent frames has to be larger than a threshold to contribute to CI, which is derived from the phase shift of all received signals [19];
- *Large frames are **bad** for CI.* This is because the bit-error-rate tends to stay constant during a frame: the more bits in the frame, the larger the probability that one gets corrupted.

V. Protocol proposals

This section provides a tutorial on the design of application protocols that exploit CI.

Low-Power Wireless Bus (**LWB**) [34] is a protocol introduced by **Glossy**s authors that provides fast data dissemination in multi-hop low-power networks[2]. It supports many-to-many, one-to-many or many-to-one communication patterns. In **LWB**, data dissemination is based on floods that are globally scheduled. All nodes participate in the transmission and data relay; the multi-hop network operates like a bus to which all nodes are connected.

LWB separates the communication into rounds as shown in Fig. 3. Nodes keep their radio off between two rounds to save energy. Each round consists of a sequence of time slots which can be used for different purposes during the data dissemination process. Within each time slot, a single node initiates a data transmission, all others retransmit the packet in a **Glossy**-like flood. The first slot of each round is a *schedule slot*; it contains a message indicating the duration of current round (T_l) and the mapping of the following *data slots* to nodes that have previously requested a transmission opportunity. A *contention slot* follows the *schedule slot*; nodes can use these slots to request opportunities to send data. These requests are used by the central scheduler to compute a new schedule. A sequence of *data slots* follows the *contention slot*. In each *data slot*, a single node starts flooding a packet through the network.

CRYSTAL [20] uses **Glossy** to improve the energy consumption of networks where data prediction is used. In data prediction applications, the data generated by the nodes is

[2] The authors make **Glossy** and **LWB** implementations available at https://github.com/ETHZ-TEC/LWB.

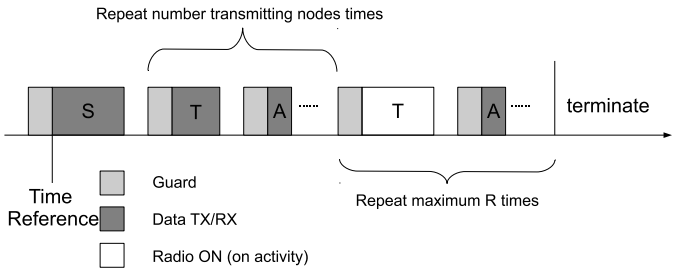


Fig. 4. An example **CRYSTAL** [20] active period.

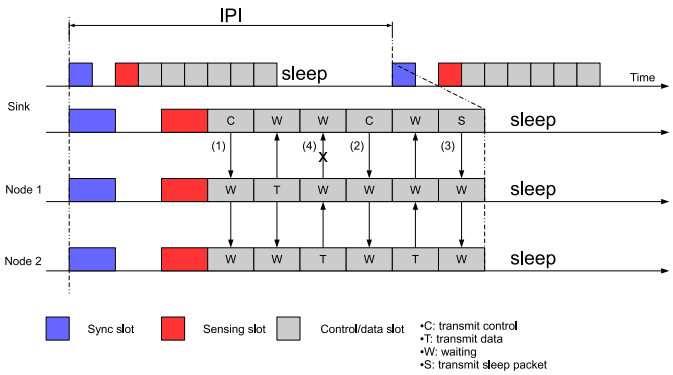


Fig. 5. Slot schedule in one inner packet interval (IPI) in **Choco** [37].

such that a model can predict future values based on past values. Nodes are aware of that model, and only send data if this data differs from what the model has predicted. In very specific applications, this can dramatically reduce the number of packets transmitted by the nodes.

CRYSTAL uses a sequence of synchronized slots similar to **LWB** but with a different communication pattern. As shown in Fig. 4, each active period starts with slots of type S used for synchronizing the whole network for the following communication and long sleep interval. It is followed by a sequence of pairs of slots T and A for node transmission and sink acknowledgment, respectively. The number of pairs (slots T and A) depends on the number of nodes with data to transmit and the desired reliability. At each slot T, a node with data to transmit acts as a flooding initiator. If the destination node receives the data, it floods back an ACK in the subsequent slot A. This process repeats until all nodes with data to transmit send their data to the destination. It is expected that all pairs of slots T and A have an initiator transmitting. Whenever a number of consecutive pairs contain no data but only negative acknowledgments, the active period ends. **CRYSTAL** introduces R consecutive silent pairs of slots T and A to determine whether to sleep under two conditions: (i) the sink sleeps after R consecutive slots T without data. So if a data packet is not successfully delivered in a slot T, the node has $R-1$ additional attempts to transmit it before the sink stops listening. (ii) the nodes sleep after N consecutive negative acknowledgments, since it knows the sink is going to sleep as well. **CRYSTAL** also contains a solution for handling the case when the acknowledgment is missing. **CRYSTAL** is shown to reduce the radio duty cycle by a factor of more than 7 while keeping close to 100% end-to-end reliability when compared to the Collection Tree Protocol **CTP** [35] using Derivation Based Prediction **DBP** [36], the state-of-the-art model for data prediction.

Choco [37] focuses on data collection and – similarly to **CRYSTAL** – employs a specific slot type for synchronizing all nodes. Fig. 5 shows one inner packet interval slot schedule defined in **Choco**. The interval starts with a *sync slot* for synchronization. The following slot is a sensing slot and allows a node to read data from its sensor. This is followed by multiple *control/data slots*, which can be used to transmit a control packet (C), transmit data (T), wait (W), or transmit a sleep packet (S). The control packet contains the schedule for the following slots, indicating which node sends in which slot. During the first *control/data slot* shown in Fig. 5 (1), the sink transmits a control packet to nodes 1 and 2 using glossy-like flooding. The following two slots are for nodes 1 and 2 to transmit data packets. During a *wait slot*, a node listens for a packet if it does not own that slot. When node 2 fails to transmit a packet to the sink (indicated as (4) in Fig. 5), in the following control slot (2), it schedules another *transmit slot* for node 2 to re-transmit the packet. During the third *control/data slot*, the sink transmits a sleep packet to nodes 1 and 2 asking them to sleep, since there is no more data to send.

"Packet in Pipe" (**PIP**) [21] is designed for transferring a large amount of data as fast as possible. Its design goal is to fully utilize the wireless medium and collect data at the sink node. In a linear topology, this is achieved by alternating transmissions at odd and even hops from the destination node. To avoid interference, communication happens at different frequencies. In a network with a tree routing structure, the size of each subtree can be chosen so that the destination node receives exactly one packet in every time slot, at most. Such structure results in optimal throughput, but solving this in all cases is known to be NP-hard (the capacitated minimum spanning tree problem). Besides, the interference graph may require more channels than the number of channels available. Every time transmitting a frame fails (no link-layer acknowledgment is received), it needs to be re-transmitted, and the pipeline "falls behind". CI helps to avoid this situation as it improves the probability of receiving both data and ACK.

Splash [38] applies constructive interference on the pipeline networking problem of **PIP**. Instead of transmitting on a single path, **Splash** floods data packets through the entire network. This avoids having to maintain a routing structure and avoid interference.

Fig. 6 shows the fours cycles of Splash, as indicated by the sub-figures (a) to (d). During the first cycle (a), the initiator starts to transmit the first packet $P1$. During the second cycle (b), the first-hop nodes receive $P1$ and forward it to the second-hop nodes using CI. During the third cycle (c), the second-hop nodes receive $P1$ and forward it to the third-hop nodes. At the same time, the initiator starts to transmit the second packet $P2$ as the first-hop nodes are free to receive. During the fourth cycle (d), similarly to cycle (b), the first-hop nodes forward the packet $P2$ to the second-hop nodes. At the same time, the third-hop nodes forward them to the fourth-hop nodes, and so on.

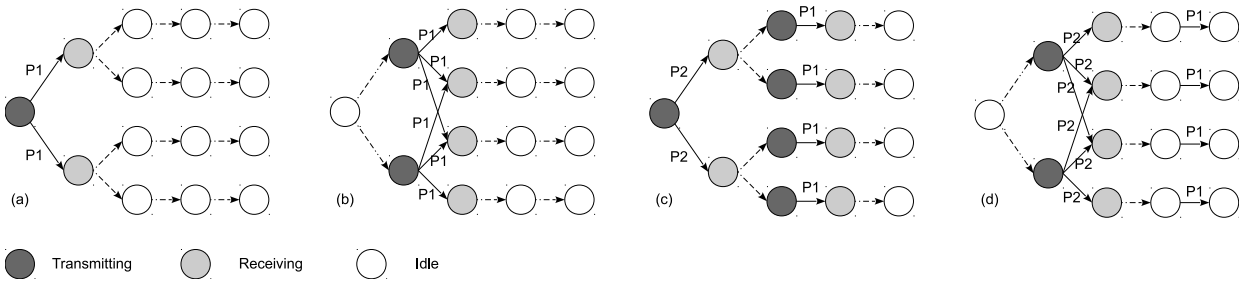


Fig. 6. **Splash** [38] splits traffic between hops.

$\mathbf{P}^3$ [39] is an improvement over **Splash** by the same authors. It splits the intermediate nodes (not including source and destination) into two different groups. The two groups are scheduled to transmit and receive at consecutive slots. The probability of data stalling in the middle of the pipe is thereby reduced, and $\mathbf{P}^3$ is able to double the throughput of **Splash**.

Fig. 7 shows how $\mathbf{P}^3$ works in four cycles for an 8 node network. The cycles are indicated as (a) to (d). The nodes in the network are grouped as *odd packet handler* (dashed white circles) and *even packet handler* (solid white circles). The packet handlers are indicated by solid gray circles. During the first cycle (a), the source node sends the first packet $P1$ to its first-hop neighbors 1, 2, 3 and 4. Since this is the odd number packet, only nodes 1 and 2 receive it, and nodes 3 and 4 abort the reception. During the second cycle (b), the source node sends the second packet $P2$ to nodes 3 and 4. At the same time, nodes 1 and 2 forward $P1$ to nodes 5 and 6 using CI. Node 7 also hears $P1$ transmitted by node 2, but aborts reception since node 7 is an even packet handler. During the third cycle (c), the source node sends the third packet $P3$ to nodes 1 and 2 (which are now done transmitting $P1$). At the same time, nodes 5 and 6 forward $P1$ received from nodes 1 and 2 to the destination node. Even packet handlers 3 and 4 forward $P2$ to nodes 7 and 8. During the fourth cycle (d), similarly to (c), the source node sends $P4$ to nodes 3 and 4. Nodes 7 and 8 forward $P2$ to the destination node at the same time. Meanwhile, nodes 1 and 2 forward $P3$ to nodes 5 and 6. It can be seen that $\mathbf{P}^3$ doubles the throughput compared to **Splash** protocol.

Different from the previous CI-based protocol designs focusing on conveying data to the destination, Dingming et al. [22] utilize CI for neighbor discovery and identification. Counting and identify neighbors are two fundamental operations for most of low-power wireless networks. The authors propose two fast and accurate mechanisms for this two purposes: **Power based Counting, (Poc)** and **Power based Identification, (Poid)**.

The main technique behind the two mechanisms is power assignment. In such a network, each node is assigned a different transmit power to respond to a specific frame used for counting purpose. In the beginning, the node broadcasts a *predicate* frame. All neighbors that receive the *predicate* frame respond using an identical ACK. Because of CI, the central node receives the ACK frame as a superposed signal. Through pre-modeled response power for each neighbor, the node can count the number of neighbors and identify them by looking at a mapping table between power and neighbor setting. The advantage of the mechanism is that it utilizes CI to make the process faster with only one transmission.

Reliable one-to-one communication is needed when a sensor node forwards a state change to a destination. This problem was exactly the challenge proposed in a dependability competition of the International Conference on Embedded Wireless Systems and Networks (EWSN), in 2016 [3], 2017 [4] and 2018 [5]. A short summary of the main works that tackled this problem employing CI-based protocols is shown in Section V-A.

A. EWSN Dependability Competition

In the dependability competition of 2016, 2017 and 2018, TelosB motes are randomly placed in an indoor environment. One of the motes is the source: an external circuit controls a bright light mounted on top of the TelosB's light sensor. Another mote is the destination: an external circuit timestamps when one of the TelosB's GPIO pins transitions low to high or vice-versa [6]. All motes are programmed by the participating teams. The goal of the competition is to have the pin on the destination mote reflect the state of the LED on the source node. That is, each time the LED switches on/off, the source node should send the corresponding signal to the destination node so it switches its pin to high/low state. Source and destination are far enough apart that they are out of radio range, and other TelosB nodes need to be used as relays. All competitors go through the same 35-min test, during which reliability (how many transitions occur), latency and overall energy consumption of the network are measured. During the test, an separate set of TelosB nodes are used to generated interference and all WiFi networks are turned off.

In the dependability competition of 2018, the setup changed a little bit. Instead of sensing the light status, the source detects the GPIO pins voltage level and sends those information to a destination or a set of destinations. Multiple sources are also presented inside the network. Those settings create the scenarios of point to multiple point (P2MP) and multiple point to multiple point (MP2MP). The performance indicators are

[3]http://ewsn2016.tugraz.at/cms/index.php%3Fid=5.html

[4]http://www.ewsn2017.org/dependability-competition.html

[5]https://ewsn2018.networks.imdea.org/call-for-competitors.html

[6]In the 2018 edition of the competition the source and destination nodes are not fixed for the whole experiment, but always there is only one source and one destination in the network

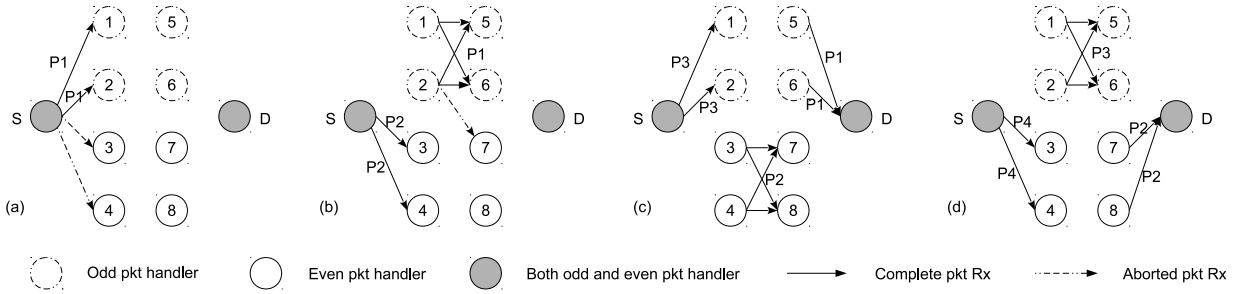


Fig. 7. $\mathbf{P}^3$ [39] improves **Splash** by splitting traffic between hops and groups of neighbors.

still the same as previous competitions: reliability, latency and energy consumption of the network.

The used of CI technique fits the competition very well. The top 3 competitors in all 3 editions [40], [41], [42], [43], [44], [45], [46], [47], [48] used a CI-based solution.

P. Sommer and Y.-A. Pignolet [40] added channel hopping to **Glossy**. Based on the counter c, each node determines the channel to be used from a static sequence, resulting in channel hopping across the 16 available frequencies. The number of employed channels impacts the reliability and energy consumption, as explained in Section VIII.

Chaos [49] builds an all-to-all information dissemination protocol on top of **Glossy**. It implements a sequence of network-wide computations and data aggregation to optimize dissemination. In [41], [42] the authors introduce **Robust Chaos**, which extends the **Chaos** framework with channel hopping and channel blacklisting to improve its performance in environments with high levels of interference. **Robust Chaos** adaptively selects a subset of channels to hop on at different locations in the network, depending on the local pattern of external interference sources.

RedFixHop [43], [44] improves the delay by taking advantage of the hardware-generated ACK frames. In this approach, data packets are generated by nodes when the packet relay counter c is even. For odd values of c, hardware-generated ACKs are employed, which completely bypasses the microcontroller and improves the delay and synchronization. This is the same techniques described in detail in Section VIII, the hands-on tutorial part of this article.

R. Lim et al. [45] add a mechanism to solve the problem of flood stalling when a subset of channels in the hopping sequence is blocked. Relay nodes re-transmit a packet also based on a timeout when no relayed packets are overheard. Yet, since the timeout used for that is much longer than the duration of a time slot, the probability of drifting by more than 500 ns is higher. The authors overcome this by exploiting the *capture effect* using a randomized transmit power among nodes.

BigBangBus [46] does not rely simply on CI, but also on the capture effect. The authors argue that pure CI at 2.4 GHz is hard to achieve because of the frequency deviation problem and the random phase of signals. In this solution, one unique schedule is agreed by the nodes once in life time (Big Bang) and then it is executed. The schedule does not have any idle gap or guard time and the source is always transmitting each event, while relays repeat the heard packet a fixed number of time. Varying the repetition pattern and how multiple channels is used lets you trade off energy, delay and reliability.

Trobinger et al. [47] modified the CRYSTAL [20] protocol by adding frequency hopping over all 16 channels and used CRYSTAL control messages (S and A) to piggy-back the sensor information.

Mao et al. [50], [48] designed OF∂COIN to flood the message which partially uses constructive interference. OF∂COIN defines a special frame where parts of it are transmitted by all nodes. Besides, the topology is studied during the first period of the competition to assign a rank to each node. These ranks are be carried inside the frame in the non-CI part. The capture effect helps this part to be received correctly. For flooding the message through the network, OF∂COIN utilizes three channels to transmit message, and one channel is selected as the lock channel. The node starts transmitting on the other two channels and then on the lock channel. The transmission restarts on the two channels and back to the lock channel for listening. Nodes turn off their radio if it received a valid message. A node retransmits if an invalid message or nothing is received.

The solutions proposed for the competition add the following improvements to **Glossy**: (i) **channel hopping**, (ii) **hardware-generated ACK frames** and (iii) **power control**. The resulting solutions are highly optimized for the competition, however, and whether the solution is useful in the general case is questionable.

Tab. I compares the protocols mentioned in this section. **LWB** is a MAC layer protocol scheduling each flood using CI in a reserved slot for each node. **CRYSTAL** is a MAC layer protocol that also uses slots, but includes an ACK packet for reliability. **Choco** mixes MAC and application layers by introducing a sensing slot in its schedule for reading sensor data. Also, its schedule is dynamically created at each control slot rather than fixed as in **LWB** and **CRYSTAL**. **Crystal Clear** is an updated version of CRYSTAL that utilizing channel hopping in its approach. The EWSN competition proposal [40] and **Chaos** are the transformations of Glossy. The former uses channel hopping to increase reliability and the later uses cahnnel hopping and blacklisting technique to further increase reliability. **RedFixHop** and **BigBangBus** are both focusing on low latency. **RedFixHop** achieves low latency by using hardware-ACK feature. **BigBangBus** achieves low latency by increasing frequency of data sending. **Splash** is

an application-oriented protocol that applies CI at different hops at the same time to increase the throughput. P_3 is an enhancement over **Splash** that applies CI at different hops, and groups the neighbors to further increase the throughput. Finally, **Poc&Poid** is a protocol designed for fast neighbors counting and identification.

VI. Security Challenges

Security is clearly the Achilles heel of CI. One perceived technical reason is that the overhead associated with security makes synchronization harder. This point is, however, largely debunked in Section VIII. The real security problem with CI is that all transmitting nodes need to use the same key for encrypting the frames, making per-link keys impossible to use. While end-to-end encryption is possible (i.e. relay nodes cannot decrypt the payload they are relaying), the requirements of having network-wide keys clearly favor Denial-of-Service (DoS) attacks.

The fact that none of the solutions presented so far include any type of security is rather astonishing. This is all the more worrying as **Glossy**-like approaches are being discussed for critical industrial applications (see discussion in Section IX). And while, to the best of our knowledge, no real security solution for CI has been designed, some work exists which analyzes the vulnerability of **Glossy**-like solution to different types of attacks.

K. Hewage et al. [51] provide an experimental study on the possible attacks that CI-based network is vulnerable to. The attacks can be classified into three types: (i) Delaying Packet Relay attack (DPR), where the concurrent transmissions are delayed by some attacker node, (ii) Relaying Packet Earlier attack (RPE), where attacker causes early concurrent transmissions, and (iii) Modifying packet attack (MP), where the content of the packet to be transmitted is modified. The experiments are carried out in a 26-node testbed deployed across an office building floor, running **Glossy**. An attacker node is placed in-between source and destination. Without the attack, the network yields 99.99% end-to-end reliability. The experimental results show how a single DPR and RPE attacker lowers the end-to-end reliability to 99.7% and 99.87%, respectively. The authors discuss the fact that the capture effect can counteract the effect of the attack. An MP attack has a much more severe impact as it replaces the relay counter field (used for synchronization) by a random value. Nodes cannot synchronize, and packets get lost.

In the same vein, Z. He et al. [52] proposed a simple DoS attack called **Arpeggio**. It exploits the fact that IEEE802.15.4 radios stay on listening mode for the number of bytes specified in the PHY header of the frame. By frequently sending (bogus) frames consisting only of a length byte with the maximum value (127), the attackers can capture the wireless medium.

Clearly, the work on security discussed above is far from complete. While implementing interesting "tricks", both [51] and [52] cannot be considered "security" proposals. What is missing from CI literature is a true security solution, in which mechanisms are used to ensure data confidentiality, integrity, and authentication. This solution should be based on well-known security solutions used in standards.

VII. Summary of the CI techniques

Table II summarizes the work illustrated in previous sections. We group the different proposals, and for each proposal, we provide a brief summary.

VIII. A Hands-on Tutorial on Constructive Interference

This section is built as a standalone hands-on tutorial of a complete CI implementation, called **Flashflood**. We are strong believers that seeing is believing, that one learns best by doing, and the devil is in the details. We, therefore, opt for a *hands-on* tutorial, which is in our mind the best way to understand *exactly* how CI works. This is all the more applicable to CI since it is a quite complex technique to implement, which requires a good understanding of low-level concepts. This hands-on tutorial is targeted at the level of practicing engineers and advanced researchers who have some experience with embedded programming. Ideally, you have a setup similar to the one described in Section VIII-B and you replicate the experiment as you read through this hands-on tutorial. But if you do not, you can read this tutorial alongside the source code, and extract (most of) the same information, without running the code.

Flashflood implements the different techniques introduced in Section IV, and serves as a basis for the discussion in Section IX. As an online companion to this article, all the source code of **Flashflood** is published under an open-source BSD license[7,8] To make the tutorial as useful as possible, and to demonstrate CI does not require cutting-edge hardware support, **Flashflood** is implemented on the TelosB mote, a very popular platform in the academic and startup communities.

We take the setup from the EWSN Dependability Competition (Section V-A) as a target application. In a **Flashflood** network, the information that specifies who is the source and the sink nodes is hard-coded (see how in Section VIII-B). When the bulb shines a bright light onto the light sensor of the source mote, it initiates a network-wide flood which indicates this new state. You can "see" the flood happening as other motes switch their LED on when the flood traverses them. With an oscilloscope or a logic analyzer, you can also see the pin at the destination mote go high. With a probe connected to the source mote, you can precisely measure the end-to-end latency. Similarly, when the light is switched off, the LEDs of all motes switch off, and the pin at the destination mote goes low.

We recommend you read through this tutorial in-order, as it is organized in a didactic manner. Section VIII-A describes what **Flashflood** firmware does. It describes the behavior of the devices precisely, without entering (or "getting lost") in minute implementation details. Section VIII-B describes how you can run the source code on your TelosB mote. It gives the high-level steps, but refers to the `README.md` instructions of

[7] https://github.com/twatteyne/flashflood/

[8] **TEMPORARY NOTE TO REVIEWERS**: we understand IEEE Surveys & Tutorials encourages multi-media additions to the articles it publishes. We are happy to provide the source code under a form different from a link to a public repository.

TABLE I
PROTOCOLS COMPARISON

Protocol	Layer	Description
LWB	MAC	Slot communication. Transmission for each node is pre-scheduled.
CRYSTAL	MAC	Slot communication. Transmission for each node is pre-scheduled. MAC layer acknowledgement required.
Choco	MAC/APP	Slot communication. Reading sensor is scheduled in a slot. Transmission for each nodes are dynamically scheduled.
Crystal Clear	MAC	An updated version of CRYSTAL using channel hopping.
[40]	MAC	Transformation of Glossy. Use channel hopping to increase reliability.
Chaos	MAC	Transformation of Glossy. Use channel hopping and blacklisting technique to further increase reliability
RedFixHop	MAC	Low latency design utilizing hardware-ACK feature.
BigBangBus	MAC	Low latency design through transmitting source data frequently on multiple channels, in a cost of energy.
Splash	APP	Multiple concurrent CI floods at different hops.
P^3	APP	Multiple concurrent CI floods at different hops. Neighbors are grouped.
Poc&Poid	MAC/PHY	Specific for Fast Neighbor Counting and Identifying.
[45]	MAC	Using capture offect to overcome non-constructive interference.
OF∂COIN	MAC	Partial CI-based frames flood through networks on multiple channels.

the repository for details such as what to install or specific commands to enter. Section VIII-C goes through the implementation "gotchas" and tricks that are key to the performance of **Flashflood**, including the auto-ACK feature of the radio, the calibration of the clock sources, etc. Section VIII-D presents the measured performance of **Flashflood** (reliability, latency, lifetime, throughput) which you should be able to measure as well.

A. Overview of *Flashflood*

Fig. 8 shows the setup we use to describe how **Flashflood** operates. It contains many important details, the explanation of which we refine throughout this tutorial. The setup consists of a multi-hop topology with source and destination motes separated by 4 hops, and with 2 relay motes at each hop. We call "hop 1 motes" all the motes which are exactly one hop from the source, i.e. motes 2 and 3 in Fig. 8. We replicate this on a bench-top setup when running the code (Section VIII-B) and when measuring the performance of **Flashflood** (Section VIII-D). Section VIII-B3 details the subtle differences in addressing between the bench-top setup (in which multi-hop communication is forced) and a final real-world deployment; for now, we focus on the bench-top setup.

Each mote is identified by an address `my_addr`, which is used as the IEEE802.15.4 short address of the device. Several motes have the same address, which is key for using CI. We take advantage of the auto-ACK feature of the radio (see Section VIII-C4), but for this to work, all frames exchanged must be formatted according to the IEEE802.15.4 standard. The source mote (mote 1) generates a 1-bit piece of data indicating the state of the light (off or on). This data floods through the network and is carried by an alternation of DATA frames (in hops 1, 3, 5, etc.) and ACK frames (in hops 2, 4, 6, etc.). In the IEEE802.15.4 frame, the data is carried in the 1-byte DSN (Data Sequence Number) field. This field is normally used to match DATA and ACK frames; in **Flashflood** this field is overloaded to carry the actual data. The key is that, per the IEEE802.15.4 standard, the ACK frame contains the same DSN as the DATA frame it acknowledges, thereby propagating the data (light state).

Referring to Fig. 8, when mote 1 generates a DATA packet (with the state of the light encoded in the DSN field), it is sent to destination address `0x0002`. Both motes 2 and 3 are configured with that short address, so *both* motes generate an ACK frame. This is done in hardware, so both ACK frames are sent at exactly the same time, resulting in CI. Both motes 4 and 5 receive the ACK, and, in software, generate a new DATA packet. Making sure that DATA frames leave the radio of both motes 4 and 5 at the same time is the tricky part; Section VIII-C discussed in detail how this is done. The DATA frames sent by motes 4 and 5 are logically equivalent to that sent by mote 1. The result is that the data floods the network hop by hop, each hop resulting in either a DATA or an ACK frame.

Of course, timing is everything in **Flashflood** as in any CI-based solution. Fig. 9 shows a chronogram of the activity of the different motes, annotated with different durations. The explanation and value of these durations are shown in Table III. Specifically, Fig. 9 shows the activity of the SFD ("Start of Frame Delimiter") pin of the motes, which allows us to visualize when a mote transmits/receives a frame.

As shown in Fig. 9, **Flashflood** operates in cycles, with one cycle every T_{cycle}, which can be tuned. In every cycle, the source mote sends a packet which contains the state of the light (on/off):

- **source mote**. It takes T_{data} μs for that frame to be sent by the source node and received by the hop 1 motes.
- **hop 1 motes**. The radio chip of the hop 1 motes automatically generate and send an ACK frame, a process that takes D_{hw} (the time for the hardware to generate the ACK), plus T_{pr} (the time to send the physical preamble of the ACK), plus T_{ACK} (the time to send the ACK, excluding its physical preamble). Since the generation of the ACK is done entirely in hardware, all hop 1 motes start transmitting the ACK at most tens of ns

TABLE II
CLASSIFICATION OF RESEARCH ON CONSTRUCTIVE INTERFERENCE

Category	Name	Description	Additions to Glossy
Modeling & Optimizing	**A-MAC** [18]	Use hardware-generated ACKs to enable CI on receiver-initiated low-power networks.	—
	Glossy [13]	Flooding protocol based on CI using IEEE802.15.4 O-QPSK signals. 500 ns maximum de-synchronization.	—
	SCIF [33]	Mathematical analysis of CI. Limiting the number of concurrent paths and nodes involved increases scalability.	Limited paths and nodes used
	TriggerCast [19]	Mathematical analysis of CI. Establishes three necessary conditions for CI and organizes the network as a tree satisfying those conditions.	—
	[14]	Investigates the 4 key factors that enable CI: power ratio, signal timing, channel coding, packet contents.	—
	[15]	Models one-hop Glossy networks. Shows the influence of packet size and capture effect on CI.	Limited packet size
	DIPA [16]	Analyzes and compares most assumptions from previous works. Proposes a power control algorithm to improve CI.	Power control
Protocol Proposals	**LWB**	Many-to-many and one-to-many protocol with Glossy-based time slots.	Synchronization & Data time slots
	CRYSTAL [20]	Reduces the energy waste of prediction application with Glossy time slots.	—
	Choco [37]	Uses a control time slot to schedule the medium to nodes with data to be collected.	Control time slot
	PIP [21]	Creates a schedule of Glossy-based time slots for fast data transfer.	Scheduling
	Splash [38] & **P^3** [39]	Fast data transfer based on flooding. Different channels are used to increase throughput.	Multiple channels
	Poc&Poid [22]	Fast neighbor counting and identify through CI. The power of superposed signal is used for identifying the neighbors.	—
	EWSN Competition: [40]	Glossy with time slots and frequency hopping.	Frequency Hopping
	EWSN Competition: Robust Chaos [41], [42]	Glossy with channel hopping and adaptive blacklisting.	Frequency Hopping
	EWSN Competition: RedFix-Hop [43], [44]	Glossy with hardware-generated ACK frames.	Hardware-generated ACK
	EWSN Competition: [45]	Glossy with re-transmissions based on timeouts.	Timeouts for data re-transmission
	EWSN Competition: BigBang-Bus [46]	Tight timeslots with no guard time and use of capture effect.	—
	EWSN Competition: [47]	CRYSTAL [20] protocol with channel hopping and sensor information piggy-backing.	—
	EWSN Competition: [50], [48]	Flooding protocol with partial constructive interference frame on multiple channel.	—
Security Challenges	[51]	Explores three types of DoS attacks: delaying packet relay, relaying packet earlier and modifying packet attack. The later shows more effectiveness.	—
	Arpeggio [52]	The attacker sends short frames with a fake length field to capture the medium, destroying CI.	—

TABLE III
FLASHFLOOD TIMING.

Time	Value	Description
T_{pr}	160 μs	physical preamble & SFD TX duration
T_{data}	320 μs	Data frame TX duration
T_{ACK}	192 μs	ACK frame TX duration
D_{sw}	214 μs	software delay before Data frame TX
D_{hw}	192 μs	radio delay before ACK frame TX

from one another[9]. This is well below the maximum de-synchronization of 500 ns for CI to work. Since the hardware copies the contents of the DSN field from the received DATA frame into the transmitted ACK frame, the data originally from the source mote reaches the hop 2 motes.

- **hop 2 motes**. While the processing is equivalent to that of the hop 1 motes (the data is relayed), the mechanism is very different as there is no radio hardware feature to forward data. Instead, when the radio of hop 2 motes hears the ACK frame from the hop 1 motes, the microcontroller wakes up, reads out the received ACK frame, creates a new DATA frame (copying the contents of the DSN field), and transmits that DATA frame. The key is that, for CI to work, all hop 2 motes need to start sending their DATA frames within 500 ns of one another, which is a challenge given the available timers. How this is

[9] This is a value we have measured, and which corresponds to the CC2420 buffer setup times [30].

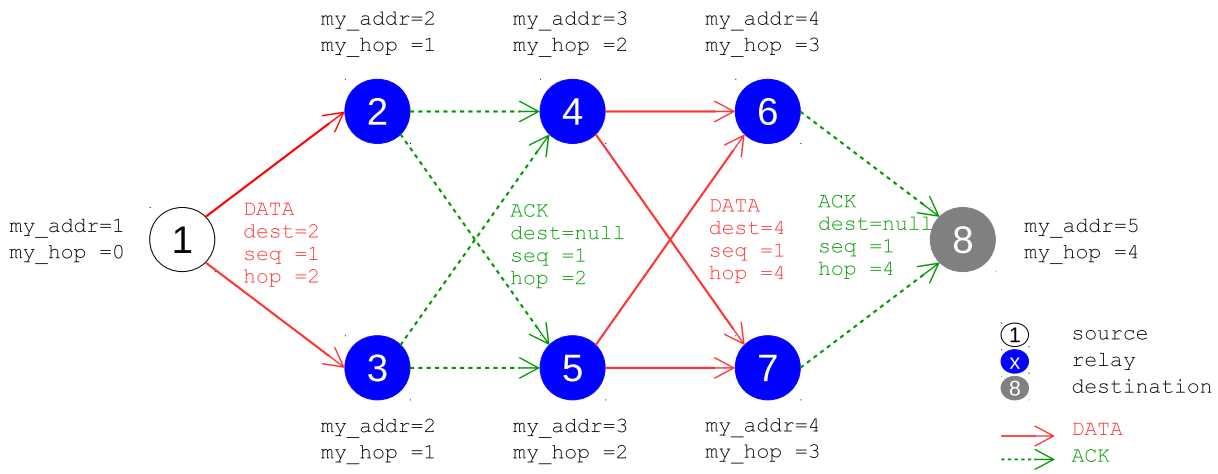


Fig. 8. Topology of the network used in the experiments.

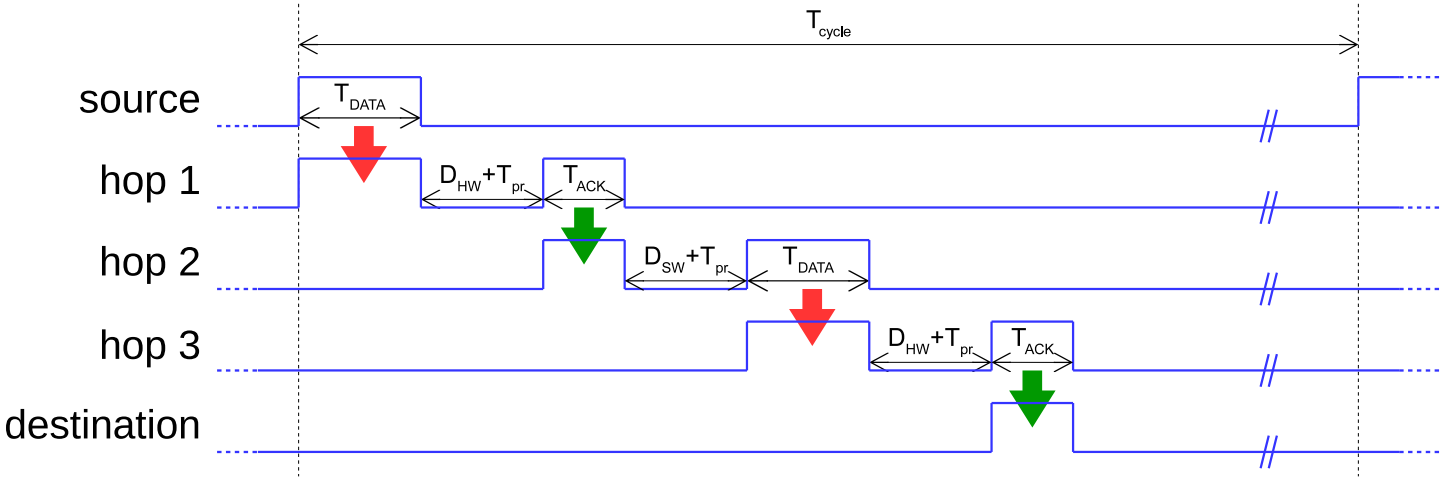


Fig. 9. Activity of the SFD ("Start of Frame Delimiter") pin of the motes as a packet floods the network. This pin which goes high at the beginning of a frame, and low at the end, both when a frame is transmitted or received by the radio.

achieved is detailed in Section VIII-C5.

- **odd hop motes (hops 3, 5, 7, etc.)**. Their behavior is equivalent to that of the hop 1 motes.
- **even hop motes (hops 2, 4, 6, etc.)**. Their behavior is equivalent to that of the hop 2 motes.

B. Running *Flashflood*

We strongly encourage the reader to download and run the source code, as you will be able to see the behavior described in this article for yourself. Alternatively, you can have the code open and read it as you go through this tutorial. The source code consists of a single 1114-line C file named `flashflood.c`. The toolchain (compiler, linker, debugger) we have used to develop the code is IAR[10], a leading Integrated Development Environment for embedded programming. We provide a number of IAR project files, which differ in the debug features that are enabled. Not to overload this article, we refer the interested reader to the source code (and specifically the `README.md` file at the root of the repository) for installation instructions as well as instructions on which project to launch.

1) Bench-top Setup: Fig. 10 is a picture of the setup we have used to develop and benchmark **Flashflood**. It consists of 8 TelosB motes connected to a USB hub. This setup exactly replicates the topology from Fig. 8. We switch on a bright light (not shown in Fig. 10) above mote 1 to trigger a state switch.

In the bottom part of the figure, there is a logic analyzer that we use to capture the timestamps of different events happening along the protocol execution.

2) Debug Pins: Throughout the **Flashflood** source code, we add statements to switch several pins of the TelosB board high or low. We use a Saleae logic analyzer[11], shown at the bottom of Fig. 10 to visualize and trace the activity of the motes it is connected to. We use 6 different pins, and assign a specific meaning to each:

[10] http://www.iar.com/

[11] https://www.saleae.com/

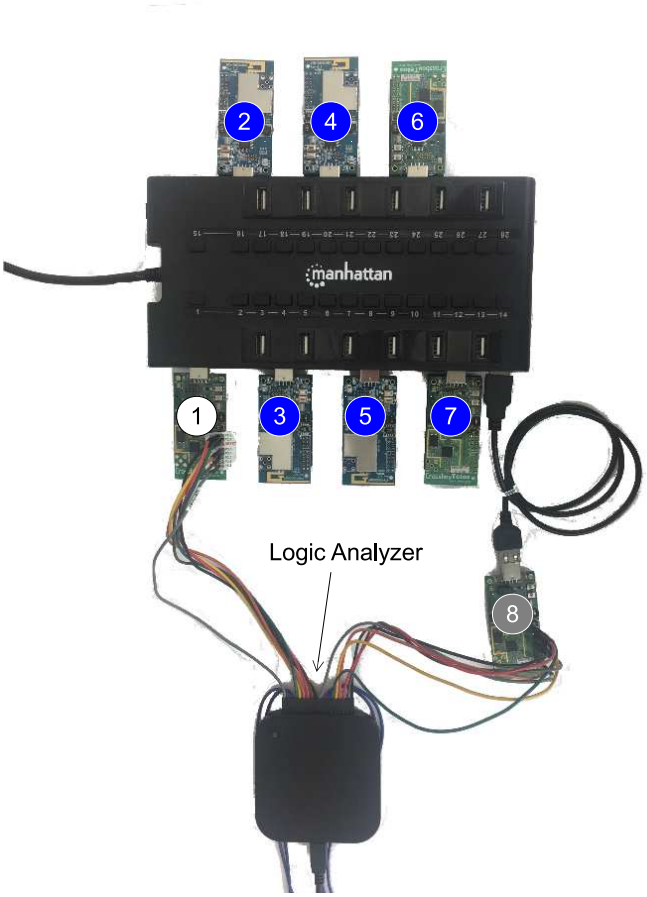


Fig. 10. Bench-top setup. 8 TelosB motes are connected to a USB hub A logic analyzer monitors the state of the 6 debug pins of different motes.

- The `light` pin indicates the state of the light of the source mote: it is high when the light is on, low when the light is off;
- The `timerAisr` pin is high when the micro-controller executes a Timer A Interrupt Service Routine, low otherwise;
- The `timerBisr` pin is high when the micro-controller executes a Timer B Interrupt Service Routine, low otherwise;
- The `rxforme` pin is low by default, and pulses high/low when the incoming frame passes address recognition;
- The `sfd` pin mimics the status of SFD pin of CC2420. This is the pin used to create Fig. 9;
- The `radio` pin is set high when the radio's oscillator is on, low otherwise. It can be used to see when the radio is on and to visualize the radio duty cycle.

Tab. IV shows where those pins are located on the TelosB board, i.e. where to connect the probes of your logic analyzer.

3) Forcing Multi-hop: Without modification, all motes hear one another in the bench-top setup, and the communication between the source and destination motes is single-hop. We, therefore, hard-code two identifiers in each mote: `my_addr` – its IEEE802.15.4 short address – and `my_hop` – the number of hops it is from the source. Both are used to filter received frame, artificially creating multi-hop communication.

TABLE IV
THE PINS USED FOR VISUALIZING THE ACTIVITY OF THE MOTES USING A LOGIC ANALYZER, AND THEIR LOCATION ON THE TELOSB BOARD.

name	MSP430f1611 pin	TelosB expansion header pin
`light`	`P2.3`	6-pin U28 header, pin 3
`timerAisr`	`P3.4`	10-pin U2 header, pin 4
`timerBisr`	`P6.6`	6-pin U28 header, pin 1
`rxforme`	`P2.6`	6-pin U28 header, pin 4
`sfd`	`P3.5`	10-pin U2 header, pin 2
`radio`	`P6.7`	6-pin U28 header, pin 2

The **Flashflood** firmware indicates to the mote's radio its `my_addr` short address and configures it so that it drops any DATA frame with a destination address different from `my_addr`. The result is that, for example, mote 4 drops the DATA frames it receives from mote 1. When receiving an ACK, the **Flashflood** firmware verifies the "hop" field it contains (see Section VIII-C2) and drops any frame with a value different from `my_hop`. The result is that, for example, mote 6 drops the ACK frames it receives from node 2.

In a real deployment, we disable this feature by (i) configuring each mote with the same `my_addr` short address and (ii) disabling the filtering based on the `my_hop` value. This is done in the source code by *not* defining the `LOCAL_SETUP` symbol as a pre-compiler directive, as explained in the `README.md` file.

C. *Flashflood* Implementation Details

The source code consists of only a single 1114-line C file. It is written and annotated as a tutorial, the numerous inline comments should allow you to follow the computation done in the different functions. Rather than repeating the annotations in the source code, this section details the high-level implementation principles and "tricks" that make CI work, and the implementation efficient. We recommend you open the `flashflood.c` file alongside reading this section.

1) Event-Driven Execution: The source code is organized around a `main()` function – which handles the initialization of a board – and two "Interrupt Services Routines" which handle interrupts from the two onboard timers: `TIMERA1_ISR` for Timer A and `TIMERB1_ISR` for Timer B. This means that, once the initialization is done, the micro-controller is only woken up for handling one of those two timers. The micro-controller then operates exclusively in interrupt mode, resulting in pure event-driven execution.

The code in the interrupt handlers is written very "linearly", with very little branching (e.g. `if` and `switch` statements), and without relying on complex operations such as the modulo operation, which can take hundreds of microseconds to execute. The goal is that we want the interrupt handler to execute fast and take roughly the same time to execute every time (low jitter).

2) Frame Formats: Fig. 11 shows the format of the two frames used in **Flashflood**. Both are compliant with the IEEE802.15.4 standard.

The DATA frame is 9 bytes long, including the 2-byte CRC. Per IEEE802.15.4 standard, the two first bytes are the Frame Control Field, which indicates how the rest of the fields are

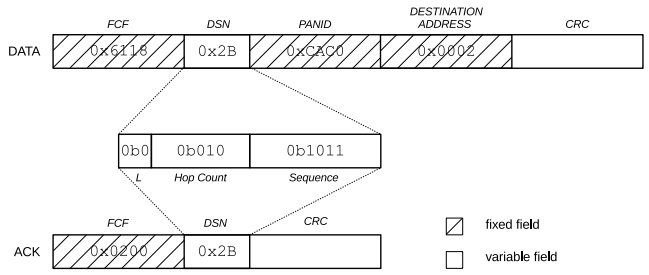


Fig. 11. The IEEE802.15.4 DATA and ACK frame formats used in the **Flashflood** protocol.

TABLE V
FRAME CONTROL FIELD (FCF) CONTENTS OF THE DATA FRAME.

bit	name	value	meaning
0-2	frame type	`b001`	DATA frame
3	security enabled	`b0`	no security
4	frame pending	`b0`	no frame pending
5	ACK requested	`b1`	ACK requested
6	intra PAN	`b1`	source PANID elided
7-9	reserved	`b000`	—
10-11	destination address mode	`b10`	16-bit destination addr.
12-13	frame version	`b01`	IEEE802.15.4-2006
14-15	source address mode	`b00`	elided

organized. The contents are detailed in Table V. Specifically, there is no source address present, but an ACK is always requested.

As detailed in Section VIII-A the data is carried in the DSN field. This field is designed to match DATA and ACK frames, but it is overloaded in **Flashflood**. As shown in Fig. 11 the 1-byte field is sub-divided into 3 fields:

- the *light* (`L`) bit which contains the state of the light, the actual data carried.
- the *hop count* field which indicates the hop count of the transmitter of this frame. This is used in the bench-top setup to artificially create a multi-hop topology (see Section VIII-B3).
- The *sequence number* field is incremented by the source node at each new packet flooded, and used for channel hopping (see Section VIII-C7).

The ACK frame is also IEEE802.15.4-compliant, but carries only the DSN field besides the FCF and the CRC. This frame is auto-generated by the hardware (see Section VIII-C4). The ACK frame is 5-byte long.

3) Pre-loading `TXFIFO`*:* The CC2420 radio has a `TXFIFO` buffer which stores the frame to be sent. The radio does *not* use this buffer when auto-generating an ACK frame, so in the context of the **Flashflood**, the `TXFIFO` will always hold DATA frames. As long as the radio is powered (including when the front-end is off), the state of the `TXFIFO` is kept. In addition, the CRC is calculated on-the-fly, i.e., while the frame is being transmitted. As a result, to speed up the execution, the **Flashflood** firmware pre-loads a DATA frame into the `TXFIFO`. Whenever that frame needs to be transmitted, the firmware overwrites the DSN field of the pre-loaded frame and has the radio transmit it. This requires a single-byte SPI write operation into the `TXFIFO` at each transmission, rather than a full reload of the whole frame.

4) Hardware Auto-ACK: Per IEEE802.15.4 standard, a device which receives a frame that has the "ACK requested" flag set must send back an acknowledgment frame, echoing the DSN field. The CC2420 has hardware support for that, so, given the right configuration, it generates and sends an ACK without micro-controller intervention. This auto-ACK feature was added to the CC2420 to simplify the code running on the micro-controller. We are using it for another benefit: the duration between the end of the DATA frame and the transmission of the ACK is always exactly the same value, to within a handful of tens of ns, allowing CI.

For this to work, three bits in the `MDMCTRL0` configuration register of the CC2420 need to be set to enable (i) address recognition and (ii) auto-ACK. The exact behavior of the radio is then as follows. When a DATA frame with the "ACK requested" flag set is received, the destination address of the DATA frame is checked against the short address of the mote. If address recognition passes, the radio creates an ACK by copying the DSN field from the DATA and appending a 2-byte CRC. The ACK is sent exactly 12 symbols after the end of the reception of the DATA frame. If address recognition fails, the CC2420 aborts the reception of the DATA frame without sending an ACK.

5) Timer Usage: **Flashflood** uses both timers from the micro-controller: TimerA and TimerB.

TimerA is used as the slow timer. It is clocked by the 32 kHz crystal which is present on the TelosB board. That is, every 1/32768 second (roughly 30.5 μs), it increments by 1. The micro-controllers' low-power mode is set to `LPM3`, meaning that TimerA never stops, even when the micro-controller does not execute code. The advantage of TimerA configured like this is that it provide a constant sense of time for a very low power consumption (around 2 μA). The disadvantage is that the speed of the timer does not give nearly enough precision to kick off actions with $<$ 500 ns timing accuracy.

For that, a second timer is used: TimerB. It is clocked from a 4.9 MHz Digitally Controller Oscillator (DCO) inside the micro-controller. At that speed, it gives a time accuracy of 1/4.9 MHz = 200 ns, which is below the 500 ns limit for CI to work. The disadvantage is that the timer consumes close to 1 mA when on. The firmware hence switches on TimerB only parsimoniously to conserve energy. TimerB also suffers from a large drift, see Section VIII-C6.

The operation of both timers is as follows. TimerA continuously runs. When TimerA signals the beginning of a cycle (which happens every T_{cycle}, see Fig. 9), the firmware switches TimerB on. TimerB is used to timestamp the reception of an ACK time, and precisely measure D_{sw} (see Table III), the duration from the end of the ACK frame reception to the beginning of the DATA frame transmission. This measurement

must be deterministic so different motes having received the same ACK send a DATA frame within 500 ns of one another, resulting in CI. After the mote has participated in the flood, it turns its radio and TimerB off until the start of the next cycle.

6) DCO Calibration: While the DCO clocking TimerB is very fast, it is also very unstable. That is, because it consists of a simple resonating circuit, it is very susceptible to temperature and supply voltage differences, and while its nominal frequency is 4.9 MHz, it can swing between 4.4 MHz and 5.4 MHz. The challenge is that TimerB is used for triggering the transmission of the DATA frame. If two different nodes have their DCO run, one at 4.9 MHz, the other at 5.4 MHz, counting 1000 ticks would take 204.08 μs and 185.19 μs, respectively. The difference, 18.89 μs, is well above the 500 ns limit, and CI would not work.

The **Flashflood** firmware therefore periodically calibrates the (unstable) DCO with the (very stable) crystal. It does so by counting how many TimerB ticks there are in a TimerA tick. It then uses a scaled value of the result to measure the time between receiving an ACK frame and sending a DATA frame.

7) Channel Hopping: To combat external interference and multi-path fading, **Flashflood** implements a simple channel hopping scheme: at each cycle, the motes switch to a different frequency. The idea is that, if communication at 2.405 GHz did not succeed, rather than retrying on the same frequency, the devices retry at a different frequency, e.g. 2.485 GHz [53]. To achieve this, the motes use the current sequence number to loop through the frequencies according to some pre-agreed hopping sequence.

The first challenge with this is that it takes longer for a node to hear its first packet and learn the current sequence number to be able to join the communication. With a 16-channel hopping sequence, it takes on average 8 cycles for a node to hear its first packet when listening on a random channel. Assuming a T_{cycle} = 20 ms (the default in **Flashflood**), this means that after being switched on, a mote listens on average for 160 ms before "joining" the network.

A second challenge is that packets can be lost. That is, a mote can relay a packet with sequence number 10, but not hear any packet with sequence number 11. In that case, **Flashflood** has to artificially increment the sequence number every T_{cycle} to be listening on the right frequency when the flood with sequence number 12 traverses the network.

8) Radio Duty Cycling: Because each frame contains a hop count field, any mote can compute when the current cycle started, based on the hop count value. And because T_{cycle} is known, it can also compute when the *next* cycle starts. The **Flashflood** firmware has a mote switch off its radio after having relayed a frame, and switch it back on at the beginning of the next cycle.

D. Measured *Flashflood* Performance

The performance of **Flashflood** is measured using the bench-top setup depicted in Fig. 10. We connect the logic analyzer to different boards for different measurements and automate the measurements to present statistically relevant results gathered over a large number of runs.

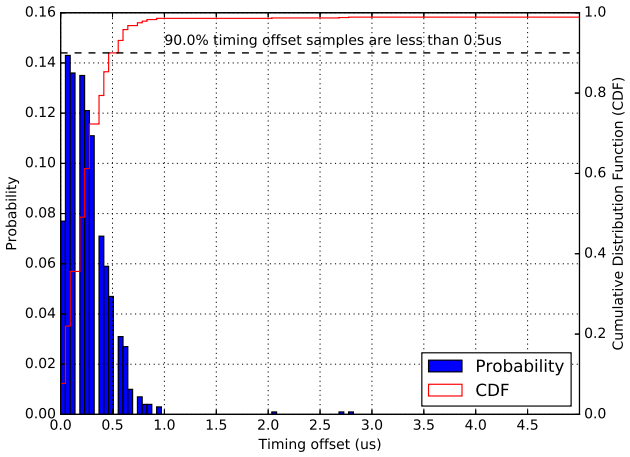


Fig. 12. Histogram of the time offset between both 2-hop motes. For 90% of the packet, the time offset is below 500 ns.

1) Time Offset: The time offset is the difference in time between different boards at the same hop transmitting frames. For CI, we want this time offset to be below 500 ns. Per the discussion in Section VIII-C5, the hard part is to get different motes to relay a DATA frame at the same time. We, therefore, connect the logic analyzer to motes 4 and 5 (both at hop 2) and timestamp the instant at which they transmit the DATA frame using the `sfd` debug pin. The logic analyzer samples at 16 MHz, giving us a timestamp granularity with an accuracy of 62.5 ns.

We have the source node to transmit 1000 packets, and plot in Fig. 12 the histogram of the absolute time offset between both motes. We see that for 90% of the packet, the time offset is below 500 ns, our target.

2) Reliability: We call reliability the portion of data sent by the source that reaches the destination. In Fig. 8, that is the percentage of packets sent by mote 1 that reaches mote 8. We measure the reliability by having mote 1 send 10,000 packets, and counting the number os packets received by mote 8.

The experiment is conducted in an office environment with heavy 2.4 GHz WiFi and IEEE802.15.4 traffic. We use unmodified **Flashflood** with channel hopping on all 16 IEEE802.15.4 frequencies. The experiment results in 94.88% end-to-end reliability over 4 hops.

Packet loss happens because of external interference. Sections VIII-E and X discuss the improvements which could be done to bring the reliability to 100%.

3) Latency: We call end-to-end latency the amount of time between the moment the light switches on and the moment the destination mote receives the packet containing the information that the light has switched on.

End-to-end latency consists of three parts: (i) the time it takes the source mote to detect the light is switched on; (ii) the time it takes for the source mote to send the packet containing that information; (iii) the time it takes for that packet to travel across the network. The end-to-end latency is expressed in (4), in which D_{sensor}, D_{source} and $D_{network}(H)$ account for the

TABLE VI
DATASHEET CURRENT CONSUMPTION NUMBERS OF THE CC2420 RADIO

state	I_{off}	I_{idle}	I_{tx}	I_{rx}
current	0.02 μA	426 μA	17.4 mA	18.8 mA

3 components. H is the number of hops between source and destination.

$$L_{end2end}(H) = D_{sensor} + D_{source} + D_{network}(H) \quad (4)$$

Since the light sensor is read every T_{cycle}, $D_{sensor} = T_{cycle}/2$ as it takes over average half T_{cycle} for a light change to be detected.

Once the light change is detected, the mote calibrates the radio and loads the packet to be sent. This takes $D_{source} = 901\ \mu s$, a value measured experimentally.

The network's delay $D_{network}(H)$, expressed in (5), is the sum of the delay at the first hop, at every odd hop and at every even hop. The delay at the first hop, D_{first} is expressed in (6). The delay at each odd and even hop is expressed in (7) and (8), respectively. These equations directly result from Fig. 9.

$$D_{network}(H) = D_{first} + \sum_{\#\ odd\ hops} D_{odd} + \sum_{\#\ even\ hops} D_{even} \quad (5)$$

$$D_{first} = T_{pr} + T_{Data} \quad (6)$$

$$D_{odd} = D_{sw} + T_{pr} + T_{Data} \quad (7)$$

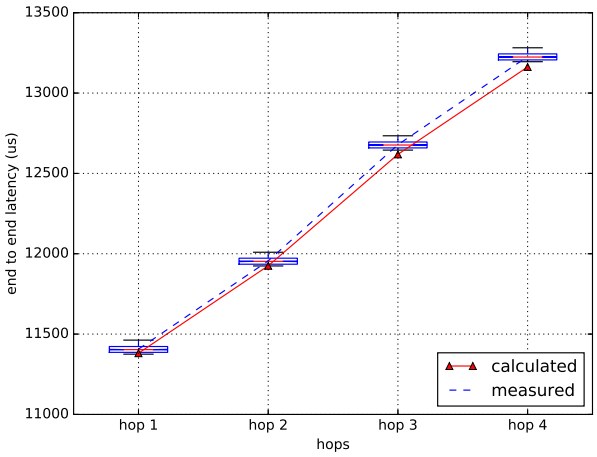
$$D_{even} = D_{hw} + T_{pr} + T_{ACK} \quad (8)$$

We measure the end-to-end latency by attaching the logic analyzer to the source and destination nodes and timestamp the instant that the light switches on, and the high-to-low transition of the `sfd` at the destination (indicating it just received a full frame). We have the source node transmit 1,000 packets to the destination node. We verify that D_{sensor} follows a uniform distribution in $[0, T_{cycle}]$, with an average value of 10 ms ($T_{cycle} = 20\ ms$ by default)

Fig. 13 shows the calculated and measured end-to-end latencies side by side, for a destination node located 1, 2, 3, 4 hops from the source. It shows that calculated and measured values match. Latency increases quasi-linearly with the number of hops (not perfectly linearly as $D_{odd} \neq D_{even}$).

4) Battery Lifetime: The energy consumption of a mote depends on how far it is from the source. The more hops between the source and the mote, the more energy it consumes.

The charge $C(h)$ consumed at every cycle by a mote at hop h is given by (9). I_{idle} is the current of the radio when in the idle state, i.e. when the radio has been switched on but not yet in transmitting or reception state. I_{tx} and I_{rx} is the current of the radio when in transmitting or reception state, respectively. The radio consumes I_{off} when switched off. Table VI gives the datasheet current consumption numbers of the CC2420 radio.



hop	**calculated**	**measured**	
		average	**std.dev.**
1	11381.00	11406.15	21.36
2	11925.00	11955.53	21.15
3	12619.00	12678.75	20.80
4	13163.00	13226.91	21.77

Fig. 13. Measured and calculated end-to-end latency of **Flashflood**.

TABLE VII
MOTE LIFETIME WHEN POWERED BY A 2000 MAH PAIR OF AA BATTERIES

hop	**source**	**1**	**2**	**3**	**4**
charge per cycle (uC)	11	21	35	45	49
lifetime (days) 20 ms cycle	151	78	48	37	34

$$C(h) = D_{init} I_{idle} + T_{tx}(h) I_{tx} + T_{rx}(h) I_{rx} \quad (9)$$

$$T_{tx}(h) = \begin{cases} T_{pr} + T_{DATA} & (source) \\ T_{pr} + T_{ACK} & (odd) \\ T_{pr} + T_{DATA} & (even) \\ 0 & (destination) \end{cases} \quad (10)$$

$$T_{rx}(H) = \begin{cases} 0 \\ (source) \\ T_{rx}(H-1) + T_{DATA} + D_{HW} + T_{pr} + T_{ACK} \\ (odd) \\ T_{rx}(H-1) + T_{ACK} + D_{SW} + T_{pr} + T_{DATA} \\ (even) \end{cases} \quad (11)$$

Between cycles, **Flashflood** turns the radio off, and leaves the micro-controller in LPM3 mode, a low-power mode in which only the 32 kHz crystal runs, consuming 2 μA. Assuming a cycle every 20 ms, Table VII gives the lifetime of a mote when powered by a 2000 mAh pair of AA batteries.

5) Throughput: The achievable throughput depends on the value of T_{cycle}, which is 20 ms by default. With this setting, the maximum throughput of **Flashflood** is 50 pkts/s. Changing the value of T_{cycle} trades off throughput, energy consumption and end-to-end latency.

E. Where Next?

With a single 1114-line C file, **Flashflood** is a simple implementation built to illustrate the different concepts developed in Section II. It offers a balanced performance between reliability, end-to-end latency, battery lifetime and throughput. This section contains a number of ideas for modifying the base **Flashflood** source code to optimize one of those parameters.

Battery lifetime could be improved by having the motes wake up later. That is, as depicted in Fig. 9, a hop 2 mote could wake up right before it expects to receive the hop 2 frame, rather than at the beginning of the cycle. The downside is that a hop 2 mote could not become a hop 1 mote at a future cycle.

Throughput could be improved by having T_{cycle} changed dynamically. That is, the packet flooded could contain the value the next T_{cycle} to use, allowing the source node to change the rate at which it can generate data packets. The downside is that this scheme would require additional signaling, in an already well-used DSN field.

Reliability could be improved by having each mote monitor whether it overhears its frame being retransmitted. If not, it could serve as a local source during the "off" period before the next cycle starts. The downside is that this would require significant additional signaling to coordinate multiple re-transmitters.

It is clear that CI opens the door for numerous intuitive *delta*-improvements, yielding very specific solutions. Yet, regardless of the improvements that are added, the question is whether CI can be useful at all, in particular in critical industrial applications.

IX. Issues and Challenges of Constructive Interference in IEEE802.15.4

IEEE802.15.4 is a standard for wireless low-power, low data-rate networks. All CI techniques discussed in Sections IV, V and VI are based on the IEEE802.15.4 physical layer. The IEEE802.15.4 standard specifies four different MAC layer protocols focusing on different applications, including TSCH that targets industrial networking. CI being integrated into MAC layer protocols specified in the IEEE802.15.4 standard would be beneficial for a large adoption of the technique and its employment in more practical scenarios.

For industrial applications, reliability, latency, energy consumption and security are important KPIs (Key Performance Indicators). The nature of flooding in CI provides a protocol design that has a very low latency. The energy consumption can also be optimized when using CI, given that proper tuning is done between energy and latency. Through a sophisticated design, latency and energy consumption should not be a major issue in CI technology.

We discuss in this section the issues and challenges of CI in the scope of IEEE802.15.4 industrial standards, mainly focusing on security and reliability aspects, which are the two most critical yet-to-solve aspects of CI-based protocols. We also discuss the amount of data CI could carry with the IEEE802.15.4 and the new IEEE802.15.4g (sub-GHz) protocols and the possibility of standardizing CI. We base this discussion on the state-of-the-art articles in Sections IV, V and VI, and the hands-on experience from the tutorial in Section VIII. The goal of this section is to look for the possibility of bringing CI from purely academic world to industry.

This section organized as following. Section IX-A discuss the big challenges that CI may face to become industrially applicable. Section IX-B discuss the reliability that CI could achieve to meet the industrial requirements. Section IX-C discuss the amount of data CI is capable to convey, referring IEEE802.15.4 standard. Section IX-D discuss the possibility to bring CI to standard.

A. Security

One of the two most critical issues of CI is security. CI is often presented as a solution for industrial emergency buttons. Yet, we are not aware of any real attempt to come up with a security scheme for CI. Some of the literature surveyed in Section VI which falls under "security" merely present attacks on CI, but are far from defining a security solution.

A security solution, as defined for example by the ANIMA and ACE working groups in the IETF[12] rely on cryptographic mechanisms to secure frames (yielding confidentiality, integrity), and Key Management Protocols to manage the keying material used by the communicating devices (yielding secure joining and repudiation). In the security communities cited above, if there exists a single exploit, a solution is considered unsecured; "probabilistic security" does not exist. A CI solution which could be considered by the industrial community is one that ensures only trusted devices are accepted in a network, that a device can be removed from the trusted set at any time, and which ensures confidentiality, integrity, and authentication during communication.

Yet, the work needed does not appear technically overcomplicated, as all the security foundations are in place. The IEEE802.15.4 standard has shipped with built-in link-layer security mechanisms since its first version in 2003, and virtually all commercial chips complying with it include hardware acceleration. Using CCM*, the security mode most commonly used, an entire frame can be secured (encrypted and authenticated) in less time than it takes to transmit a single byte.

What does remain to be answered are questions including *How does a new device securely join a CI-based network? How does that device acquire link-layer keys? How are those keys used? How does the network repudiate a device?*

The flooding nature of CI presents an additional challenge. Since the different frames participating in CI need to be identical, they need to be secured using the same key. This translates into the need of using network-wide link-layer keys, rather than per-link link-layer keys.

B. Reliability

The second most critical issue with CI is reliability. To be precise, end-to-end reliability. Reliability has received significant attention from the academic community, and virtually

[12] http://tools.ietf.org/wg/anima/charters, https://tools.ietf.org/wg/ace/charters

all related work discussed in Section IV, Section V and Section V-A touch upon it. CI is presented essentially as an efficient flooding scheme; communication between source and destination is very fast when it works, but it does not always work.

Related work has put significant focus on increasing the probability that a message gets to its destination. Yet, what is missing is the ability for the device sending to know whether its data got there. In the emergency button use case, the button does not know whether the machine has stopped, using today's CI solutions. If it knew the machine had not stopped, it could restart a CI flood within a handful of milliseconds after the first try. And while worst-case latency necessary goes up with the number of retries, the end-to-end reliability (including retries) must be 100%.

C. Amount of Data

Based on the analysis of CI in [15], the longer the duration of the transmissions is, the worse packet delivery ratio CI would achieve. This is because the drift of clock between two radios increases proportionally to the size of the packet. IEEE802.15.4 allows a maximum length frame 127 bytes to be transmitted. According to the result from [15], transmitting 126 bytes introduces offset between two transmissions close to 0.10 μs. This is acceptable since does not exceeding the 0.5 μs limit of CI.

IEEE802.15.4g is an amendment for IEEE802.15.4 to supports multiple sub-GHz physical layers to enable long-range communication. It allows a maximum length frame of 2047 bytes. The large frame may introduce higher time offset between two concurrent transmissions. In addition, the lower chip rate (100 kcps or 1 Mcps depending on the frequency band used) increases even more the clock drift and makes it hard to comply with the 500 ns maximum offset. Theoretically, there should be *no* limitation on the amount of data for CI to be used with IEEE802.15.4g. However, the limits on the clock offset between two concurrent transmissions when sending long frames needs to be investigated experimentally.

D. Standardization

Large industrial end-users rarely use single-vendor solutions, i.e. solutions which are sold only by a single company. A good way to ensure interoperability is through standardization. Having a standard that covers a technology serves two purposes: (i) it acknowledges the interest of enough companies to push the technology through the standardization process and (ii) it ensures to a large extent multiple interoperable products on the market.

To the best of our knowledge, no communication standard exploits CI, and there is no ongoing standardization activity that proposes it.

X. Research Directions

According to the summary from the previous section, it is clear that CI has two main issues – on security and on reliability – for meeting the industrial requirements. CI simply does not provide a complete solution to be adopted by industrial standards. However, it could be a solution in the future if it becomes part of a standardized protocol. Time Synchronized Channel Hopping (TSCH), as a technique adopted by multiple industrial standards, could be a target protocol for standardization of CI.

In this section, we first introduced the TSCH protocol and its related challenges. Second, we provide a discussion on research directions to integrate CI with TSCH and possibly solve the challenges exposed.

A. Time Synchronized Channel Hopping

For a wireless low-power industrial network, the standard leading the market today is WirelessHART, a wireless extension of the HART standard. First published in 2008, WirelessHART is now ubiquitous in the industrial space. The core technology of WirelessHART is TSCH, a low-power wireless medium access technique which combines synchronization to achieve ultra-low power consumption and channel hopping to achieve wire-like reliability.

In 2016, TSCH has become standardized as part of the IEEE802.15.4 standard. 6TiSCH, a standardization working group at the IETF, was created to combine IEEE802.15.4 TSCH with IPv6. Today, pre-6TiSCH products, such as Analog Devices' SmartMesh product lines, are on the market with the following key performance indicators [8]:

- wire-like end-to-end reliability over 99.999%
- $<$ 50 μA average power consumption at 3.6 V, yielding over a decade of battery lifetime
- certified security

Tens of thousands of TSCH networks are operating today. One vendor alone, Emerson[13], announced at the time of writing over 35,400 networks deployed, with an accumulated mote operation time of over 10,616,346,164 hours[14].

TSCH has also become an important research topic. An indication of this is that the 3 leading open-source IoT stack implementations (OpenWSN, Contiki, RIOT) now use 6TiSCH at the core of their protocol stack. A largely unanswered question which is actively investigated is whether latency can be predicted in TSCH networks, which is essential for using TSCH networks in control applications. Early demonstrations showed a 6TiSCH network used to control an inverted pendulum [54].

B. Integrate CI with TSCH

Based on the discussion in Sec IX, CI will not replace established solutions such as TSCH, as it covers only a very small subset of the capabilities of TSCH. We *do* believe, however, that there is a benefit in having a system for transporting/flooding alarm messages across a network fast. We also believe CI offers a performance trade-off between the amount of data, latency and power consumption which TSCH does not attain.

We, therefore, state that there is a benefit in adding CI capabilities to TSCH. This could be done by (i) dedicated

[13] www.emerson.com

[14] http://www.emerson.com/en-us/expertise/automation/industrial-internet-things/pervasive-sensing-solutions/wireless-technology

TSCH cells to CI communications and (ii) adding a link-layer acknowledgment to CI, in a way similar to [17]. CI would benefit from the security solution being developed for example in the IETF 6TiSCH working group. CI could appear as an additional service offered by a TSCH network.

XI. Conclusion

This article starts by introducing the Constructive Interference (CI) technique and presents the state-of-the-art in this vibrant research topic. This tutorial covers related work which spans 10 years and is presented in a chronological order to highlight the progression of the technology.

It then provides a comprehensive hands-on tutorial about CI. The **Flashflood** implementation is the example presented and published as open-source as an online addition to this article. All low-level subtleties (and complexities) related to implementing CI are explained, including event-based programming, frame formats, autoAck configuration, Timer usage, DCO calibration, channel hopping strategy and low power duty cycling setting.

This article concludes with a discussion about the usefulness of CI for Industrial IoT applications. Its relevance is very small today. And although CI does not provide functionalities for the IIoT solutions, as TSCH does, CI has sufficient potential for justifying the effort of combining it with TSCH.

Acknowledgments

This work is partially supported by Inria through the OpenWSN project and the DIVERSITY associate team, the Spanish Ministry of Economy and the FEDER regional development fund through the SINERGIA project (TEC2015-71303-R) and CERNET through the CERNET Innovation Project (NO.NGII20160304).

Acronyms

Follow is the acronyms used in this paper and ordered by the subsequence they appear in the paper.

CI	Constructive Interference
LR-WPAN	Low-Rate Wireless Personal Area Networks
TSCH	Time slotted Channel Hopping
IIoT	Industrial Internet of Thing
ISM	Industrial, Scientific and Medical
TDMA	Time Division Multiple Access
FDMA	Frequency Division Multiple Access
CDMA	Coding Devision Multiple Access
MAC	Media Access Control
O-QPSK	Offset Quadrature Phase-shift Keying
HDD	Hard Decision Decoding
SDD	Soft Decision Decoding
MSK	Minimum Shift keying
DSSS	Discrete Sequence Spread Spectrum
SFD	Start of Frame delimiter
MPDU	MAC protocol Data Unit
PDR	Packet Delivery Ratio
SINR	Signal to Interference plus Noise Ratio
DoS	Denial of Service
DSN	Data Sequence Number
DCO	Digitally Controlled Oscillator

References

[1] IEEE, *802.15.4-2015: IEEE Standard for Local and Metropolitan Area Networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer*, IEEE Std., October 2015.

[2] Z. Alliance, *ZigBee Specification*, ZigBee Alliance Std., Rev. 20, 2012.

[3] ISA, *ISA-100.11a-2011: Wireless Systems for Industrial Automation: Process Control and Related Applications*, International Society of Automation (ISA) Std., May 2011.

[4] HartComm, *WirelessHART Specification 75: TDMA Data-Link Layer*, HART Communication Foundation Std., Rev. 1.1, 2008.

[5] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized Protocol Stack for the Internet of (Important) Things," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1389–1406, 12 December 2013.

[6] P. Thubert, T. Watteyne, M. R. Palattella, X. Vilajosana, and Q. Wang, "IETF 6TSCH: Combining IPv6 Connectivity with Industrial Performance," in *International Workshop on Extending Seamlessly to the Internet of Things (esIoT)*. Taiwan: IEEE, 3-5 July 2013, pp. 541–546.

[7] G. Zheng, L. Krikidis, C. Masouros, S. Timotheou, D.-A. Toumpakaris, and Z. Ding, "Rethinking the Role of Interference in Wireless Networks," *IEEE Communications Magazine*, pp. 152 – 158, 21 November 2014.

[8] T. Watteyne, J. Weiss, L. Doherty, and J. Simon, "Industrial IEEE802.15.4e Networks: Performance and Trade-offs," in *International Conference on Communications (ICC), Internet of Things Symposium*. London, UK: IEEE, 8-12 June 2015.

[9] L. Doherty, W. Lindsay, and J. Simon, "Channel-Specific Wireless Sensor Network Path Data," in *International Conference on Computer Communications and Networks ICCCN*. Turtle Bay Resort, Honolulu, Hawaii, USA: IEEE, 13-16 August 2007, pp. 89–94.

[10] K. Pister and L. Doherty, "TSMP: TIME SYNCHRONIZED MESH PROTOCOL," in *Proceedings of the IASTED International Symposium Distributed Sensor Network (DSN 2008)*, International Association of Science and Technology for Development (IASTED). Orlando, Florida, USA: ACTA press, 16–18, November 2008.

[11] Frost and Sullivan, "NC5E-65: US Machine-to-Machine (M2M) Communications Markets," pp. 0–86, 2013.

[12] K. Pister, P. Thubert, S. Dwars, and T. Phinney, *Industrial Routing Requirements in Low-power and Lossy Networks*, IETF Std. RFC5673, 2009.

[13] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient Network Flooding and Time Synchronization with Glossy," in *Information Processing in Sensor Networks (IPSN)*. Chicago, IL, USA: IEEE/ACM, 12-14 April 2011, pp. 73–84.

[14] M. Wilhelm, V. Lenders, and J. B. Schmitt, "On the Reception of Concurrent Transmissions in Wireless Sensor Networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 12, pp. 6756–6767, December 2014.

[15] D. Yuan and M. Hollick, "Let's Talk Together: Understanding Concurrent Transmission in Wireless Sensor Networks," in *Local Computer Networks (LCN)*. Sydney, Australia: IEEE, 21-24 October 2013, pp. 219–227.

[16] S. V. Rao, M. Koppal, R. V. Prasad, T. Prabhakar, C. Sarkar, and I. Niemegeers, "Murphy Loves CI: Unfolding and Improving Constructive Interference in WSNs," in *International Conference on Computer Communications (INFOCOM)*. San Francisco, CA, USA: IEEE, 10-14 April 2016.

[17] P. Dutta, R. Musaloiu-E, I. Stoica, and A. Terzis, "Wireless ACK Collisions Not Considered Harmful," in *Hot Topics in Networks (HotNets)*, Calgary, Alberta, Canada, 6-7 October 2008.

[18] P. Dutta, S. Haggerty-Dawson, Y. Chen, M. C.-J. Liang, and A. Terzis, "Design and Evaluation of a Versatile and Efficient Receiver-Initiated Link Layer for Low-Power Wireless," in *Conference on Embedded Networked Sensor Systems (SenSys)*. Zurich, Switzerland: ACM, 3-5 November 2010, pp. 1–14.

[19] Y. Wang, H. Yuan, D. Cheng, Y. Liu, and X. Li, "TriggerCast: Enabling Wireless Constructive Collisions," in *International Conference on Computer Communications (INFOCOM)*. Centro Congressi Lingotto Turin, Italy: IEEE, 14-19 April 2013, pp. 480–484.

[20] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data Prediction + Synchronous Transmissions = Ultra-Low Power Wireless Sensor Networks," in *Conference on Embedded Network Sensor Systems (SenSys)*. Stanford, CA, USA: ACM, 14-16 November 2016, pp. 83–95.

[21] B. Raman, K. Chebrolu, S. Bijwe, and Gabale, "PIP: a Connection-Oriented, Multi-Hop, Multi-Channel TDMA-based MAC for High Throughput Bulk Transfer," in *Conference on Embedded Networked Sensor Systems (SenSys)*. Zurich, Switzerland: ACM, 03-05 November 2010, pp. 15–28.

[22] D. Wu, C. Dong, S. Tang, H. Dai, and Y. Chen, "Fast and Fine-grained Counting and Identification via Constructive Interference in WSNs ," in *IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IEEE. Berlin, Germany: IEEE, 15–17 April 2014.

[23] M. Bouaziz and A. Rachedi, "A Survey on Mobility Management Protocols in Wireless Sensor Networks based on 6LoWPAN Technology ," *Computer Communications, Elsevier*, vol. 74, pp. 3–15, January 2016.

[24] B. Rashid and M. H. Rehmani, "Applications of Wireless Sensor Networks for Urban Areas: A Survey," *Elsevier Journal of Network and Computer Applications*, vol. 60C, pp. 192–219, 2016.

[25] S. Misra, M. Reisslein, and G. Xue, "A Survey of Multimedia Streaming in Wireless Sensor Networks," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 4, pp. 18–39, 2008.

[26] A. Ahmad, S. Ahmad, H. Mubashir, and H. U. Naveed, "A Survey on Radio Resource Allocation in Cognitive Radio Sensor Networks," *IEEE Communication Surveys & Tutorials*, vol. 17, no. 2, pp. 888–917, 09 February 2015.

[27] W. Rehan, S. Fischer, M. Rehan, and M. H. Rehmani, "A Comprehensive Survey on Multichannel Routing in Wireless Sensor Networks," *Journal of Network and Computer Applications*, 2017.

[28] S. H. R. Bukhan, M. Husain, and S. Siraj, "A Survey of Channel Bonding for Wireless Networks and Guidelines of Channel Bonding for Futuristic Cognitive Radio Sensor Networks," *IEEE Communication Surveys & Tutorials*, vol. 18, no. 2, pp. 924–948, 30 November 2016.

[29] S. Amri, A. B. Fekher Khelifi, A. Rachedi, M. L. Kaddachi, and M. Atri, "A new fuzzy logic based node localization mechanism for Wireless Sensor Networks," *Future Generation Computer Systems, Elsevier*, 2017.

[30] T. Instruments, *CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver*, rev. c ed., Texas Instruments, Dallas, TX, USA, March 7 2013.

[31] D. Moss, J. Hui, and K. Klues, "TinyOS TEP105 - Low Power Listening," UC Berkeley, Tech. Rep., 2007. [Online]. Available: https://github.com/tinyos/tinyos-main/blob/master/doc/txt/tep105.txt

[32] Y. Sun, O. Gurewitz, and D. B. Johnson, "RI-MAC: A Receiver-initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '08. ACM, 2008, pp. 1–14.

[33] Y. Wang, H. Yuan, X. Mao, Y. Liu, and X.-y. Li, "Exploiting Constructive Interference for Scalable Flooding in Wireless Networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 6, pp. 1880–1889, December 2013.

[34] F. Ferrari, M. Zimmerling, L. mottola, and L. Thiele, "Low-Power Wireless Bus," in *Conference on Embedded Networked Sensor Systems (SenSys)*. Toronto, ontario, Canada: ACM, 06-09 November 2012, pp. 1–14.

[35] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. levis, "Collection Tree Protocol," in *Conference on Embedded Networked Sensor Systems (SenSys)*. Berkeley, CA, USA: ACM, 4-6 November 2009, pp. 1–14.

[36] U. Raza, A. Camerra, A. L. Murphy, T. Paipanas, and G. P. Picco, "Practical Data Prediction for Real-World Wireless Sensor Networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2231–2244, August 2015.

[37] M. Suzuki, Y. Yamashita, and H. Morikawa, "Low-Power, End-to-End Reliable Collection using Glossy for Wireless Sensor Networks," in *Vehicular Technology Conference (VTC)*. Dresden, Germany: IEEE, 2-5 June 2013.

[38] M. Doddavenkatappa, M. Choon, and B. leong, "Splash: Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Lombard, IL, USA: USENIX, 2-5 April 2013, pp. 269–282.

[39] M. Doddavenkatappa and M. C. Chan, "P3: A Practical Packet Pipeline using Synchronous Transmissions for Wireless Sensor Networks," in *Information Processing in Sensor Networks (IPSN)*. Berlin, Germany: IEEE, 15-17 April 2014, pp. 203–214.

[40] P. Sommer and Y.-A. Pignolet, "Competition: Dependable Network Flooding using Glossy with Channel-Hopping," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Graz, Austria: ACM, 15-17 February 2016, pp. 303–303.

[41] L. Beshr, Al Nahasand Olaf, "Competition: Towards Low-Latency, Low-Power Wireless Networking under Interference," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Graz, Austria: ACM, 15-17 February 2016, pp. 287–288.

[42] B. Al Nahas and O. Landsiedel, "Competition: Towards Low-Power Wireless Networking that Survives Interference with Minimal Latency," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Graz, Austria: ACM, 20-22 February 2017, pp. 268–269.

[43] J. Klaue, A. Corona, K. Martin, j. G. Jimenez, and A. Escobar, "Competition: RedFixHop," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Graz, Austria: ACM, 15-17 February 2016, pp. 289–290.

[44] A. Escobar, j. G. Jimenez, F. J.Cruz, J. Klaue, A. Corona, and D. Tati, "Competition: RedFixHop with Channel Hopping," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Graz, Austria: ACM, 15-17 February 2017, pp. 303–303.

[45] R. Lim, R. D. Forno, F. Sutton, and L. Thiele, "Competition: Robust Flooding using Back-to-Back Synchronous Transmissions with Channel-Hopping," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Graz, Austria: ACM, 15-17 February 2017, pp. 270–271.

[46] A. Escobar, F. Moreno, A. J. Cabrera, J. Garcia-Jimenez, F. J. Cruz, U. Ruiz, J. Klaue, A. Corona, D. Tati, and T. Meyerhoff, "Competition: BigBangBus," in *EWSN 18 Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*. Madrid, Spain: ACM, 14-16 February 2018, pp. 213–214.

[47] M. Trobinger, T. Istomin, A. L. Murphy, and G. P. Picco, "Competition: CRYSTAL Clear: Making Interference Transparent," in *EWSN 18 Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*. Madrid, Spain: ACM, 14-16 February 2018, pp. 217–218.

[48] X. Ma, P. Zhang, W. Tang, X. Li, W. He, F. Zhang, J. Wei, and O. Thee, "Competition: Using Enhanced OF∂COIN to Monitor Multiple Concurrent Events under Adverse Conditions," in *International Conference on Embedded Wireless Systems and Networks*. Madrid, Spain: ACM, 14-16 February 2018, pp. 211–212.

[49] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2013, pp. 1–14.

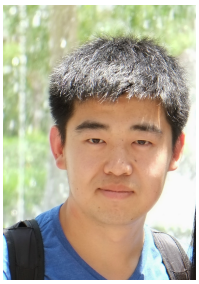
[50] X. Ma, W. Tang, W. He, F. Zhang, and J. Wei, "Competition: Using OFCOIN under Interference," in *International Conference on Embedded Wireless Systems and Networks*. Uppsala, Sweden: ACM, 20–22, February 2017, pp. 266–267.

[51] K. Hewage, S. Raza, and T. Voigt, "An Experimental Study of Attacks on the Availability of Glossy," *Elsevier Computers & Electrical Engineering*, vol. 41, pp. 115–125, January 2015.

[52] Z. He, K. Hewage, and T. Voigt, "Arpeggio: a Penetration Attack on Glossy Networks," in *International Conference on Sensing, Communication, and Networking (SECON)*. London, UK: IEEE, 27-30, June 2016.

[53] T. Watteyne, A. Mehta, and K. Pister, "Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense," in *Proceedings of the 6th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*. ACM, 2009, pp. 116–123.

[54] C. Schindler, T. Watteyne, X. Vilajosana, and K. Pister, "Implementation and Characterization of a Multi-hop 6TiSCH Network for Experimental Feedback Control of an Inverted Pendulum," in *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2017, pp. 1–8.



Tengfei Chang is a Postdoc Research Engineer at Inria-EVA, Paris. He obtained his Ph.D. degree in Computer System Architecture at 2017 from University of Science and Technology, Beijing. At 2014, he was visiting at the University of California, Berkeley as visiting scholar. From November 2015 to October 2017, he joined Inria-EVA team as a Pre-Postdoc Research Engineer, leading the project of OpenWSN, which is an Open Source project founded by UC Berkeley. At 2017, he joined the F-Interop project as a Postdoc Research Engineer, which is an H2020 European research project. He is also one of the main implementors of IETF 6TiSCH standard protocol stack. He has worked as technical support for 6TiSCH interoperability plugtest. He has huge interests on Wireless Sensor and Actuator Network, Swarm Robotic and any Embedded System design.

Thomas Watteyne is an insatiable enthusiast of low-power wireless mesh technologies. He holds an advanced research position at Inria in Paris, in the EVA research team, where he designs, models and builds networking solutions based on a variety of Internet-of-Things (IoT) standards. He is Senior Networking Design Engineer at Analog Devices, in the Dust Networks product group, the undisputed leader in supplying low power wireless mesh networks for demanding industrial process automation applications. Since 2013, he co-chairs the IETF 6TiSCH working group, which standardizes how to use IEEE802.15.4e TSCH in IPv6-enabled mesh networks, and is a member of the IETF Internet-of-Things Directorate. Prior to that, Thomas was a postdoctoral research lead in Prof. Kristofer Pister's team at the University of California, Berkeley. He founded and co-leads Berkeley's OpenWSN project, an open-source initiative to promote the use of fully standards-based protocol stacks for the IoT. Between 2005 and 2008, he was a research engineer at France Telecom, Orange Labs. He holds a Ph.D. in Computer Science (2008), an MSc in Networking (2005) and a MEng in Telecommunications (2005) from INSA Lyon, France. He is a Senior member of IEEE. He is fluent in 4 languages.

Xavier Vilajosana is the principal investigator at the WINE research group at UOC and professor at the Computer Science, Telecommunications, and Multimedia department. In addition, Xavier is a co-founder of Worldsensing, and OpenMote Technologies, 2 prominent startups developing cutting-edge IoT related technology. Until March 2016 Xavier has been a senior researcher at the HP R&D Labs. From January 2012 to January 2014, Xavier was visiting Professor at the University of California Berkeley holding a prestigious Fulbright fellowship. In 2008, he was visiting researcher of France Telecom R&D Labs, Paris. Xavier has been one of the main promoters of low power wireless technologies, co-leading the OpenWSN.org initiative at UC Berkeley, and promoting the use of low power wireless standards for the emerging Industrial Internet paradigm. He also contributed to the industrialization and introduction of Low Power Wide Area Networks to urban scenarios through Worldsensing. Xavier is the author of different Internet Drafts and RFCs, as part of his standardization activities for low power industrial networks. Xavier is contributing actively at the IETF 6TiSCH, 6Lo and ROLL Working Groups. Xavier holds more than 20 patents, more than 30 high impact journal publications and have contributed with several demos, tutorials, and courses in the field of low power wireless networks. Finally, Xavier is IEEE Senior Member and founding member and vocal of the IEEE Sensors Council in Spain.

Pedro Henrique Gomes is currently an experienced researcher at Ericsson Research, Brazil. He is also a Ph.D. candidate at Ming Hsieh Department of Electrical Engineering, University of Southern California. His research is focused on developing algorithms to optimize reliability, delay and energy efficiency of networks based on Timeslotted Channel Hopping (TSCH) protocol. He received his MSc in Computer Science (2011) and BSc in Computer Engineering (2007) from the University of Campinas, Brazil.