Countering Three Denial-of-Sleep Attacks on ContikiMAC

Konrad-Felix Krentz
Hasso-Plattner-Institut, Germany
konrad-felix.krentz@hpi.de

Christoph Meinel
Hasso-Plattner-Institut, Germany
christoph.meinel@hpi.de

Hendrik Graupner
Bundesdruckerei, Germany
hendrik.graupner@bdr.de

Abstract

Like virtually all media access control (MAC) protocols for 802.15.4 networks, also ContikiMAC is vulnerable to various denial-of-sleep attacks. The focus of this paper is on countering three specific denial-of-sleep attacks on ContikiMAC, namely ding-dong ditching, pulse-delay attacks, and collision attacks. Ding-dong ditching is when attackers emit interference, inject frames, or replay frames so as to mislead ContikiMAC into staying in receive mode for extended periods of time and hence consuming much energy. Pulse-delay attacks are actually attacks on time synchronization, but can also be launched against ContikiMAC's phase-lock optimization to cause an increased energy consumption. Lastly, in collision attacks, an attacker provokes retransmissions via jamming. In this paper, to counter these three kinds of denial-of-sleep attacks, we propose two optimizations to ContikiMAC. The dozing optimization, on the one hand, significantly reduces the energy consumption under ding-dong ditching. Beyond that, the dozing optimization helps during normal operation as it reduces the energy consumption of true wake ups, too. The secure phase-lock optimization, on the other hand, is a version of ContikiMAC's phase-lock optimization that resists pulse-delay attacks. Additionally, the secure phase-lock optimization makes ContikiMAC resilient to collision attacks, as well as more energy efficient. We implemented and evaluated both optimizations using the Contiki operating system and OpenMotes.

Categories and Subject Descriptors

C.2.1. [**Network Architecture and Design**]: Wireless communication; C.2.0. [**General**]: Security and protection

General Terms

Security, Design.

Keywords

Denial-of-sleep, MAC security, low-power listening.

International Conference on Embedded Wireless Systems and Networks (EWSN) 2017
20–22 February, Uppsala, Sweden
© 2017 Copyright is held by the authors.
Permission is granted for indexing in the ACM Digital Library
ISBN: 978-0-9949886-1-4

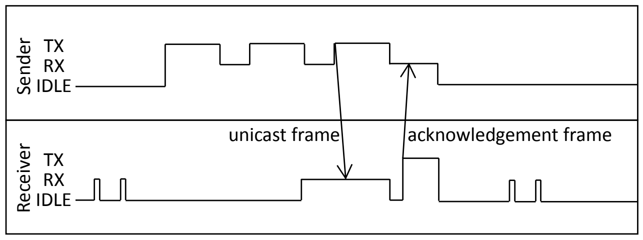
1 Introduction

ContikiMAC is a widely-used media access control (MAC) protocol for 802.15.4 networks [6, 1]. Its features include high energy efficiency and an easy-to-implement design. These features render ContikiMAC very suitable for use on Internet of things (IoT) devices as IoT devices typically are energy and resource constrained.

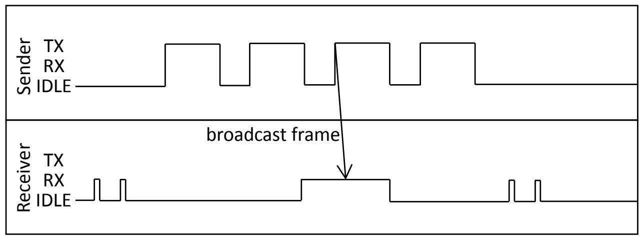
1.1 Operation of ContikiMAC

The energy efficiency of ContikiMAC stems from leaving an 802.15.4 node's transceiver off most of the time. Instead, as shown in Figure 1, receivers regularly perform two clear channel assessments (CCAs). If any of these CCAs returns negative, receivers stay in receive mode to potentially receive a frame. Senders, on the other hand, repeatedly transmit each frame (aka strobing) for a whole wake-up interval, plus once to accommodate corner cases. As for unicast frames, senders may stop strobing a unicast frame prematurely when the intended receiver replies with an acknowledgement frame. For this, senders have to scan for acknowledgement frames in between successively strobed unicast frames. However, if no acknowledgement frame comes back, senders may retry by sending another strobe of unicast frames.

Also, ContikiMAC comprises two optimizations. The phase-lock optimization, on the one hand, lets senders learn the wake-up times of neighboring nodes in order to start the strobing of a unicast frame right before the intended receiver wakes up. Furthermore, once a sender has learned when a receiver wakes up, the sender no longer needs to strobe unicast frames to the receiver for a whole wake-up interval plus once if no acknowledgement frame returns, but only for a shorter time span plus once. However, to handle clock drift, a sender must relearn the wake-up time of a receiver (i) if a critical number of consecutive unicast transmissions to the receiver were unacknowledged, as well as (ii) if, within a certain period of time, all unicast transmissions to the receiver were unacknowledged. The fast-sleep optimization, on the other hand, allows receivers to go back to sleep after they obtained a negative CCA at three occasions. First, if radio noise lasts longer than for transmitting a maximum-length frame, receivers can turn off their 802.15.4 transceiver. Second, if the silence period after the radio noise takes longer than ContikiMAC's inter-frame period, receivers can go back to sleep, too. Third, if no frame is detected after the silence period, receivers can go back to sleep, as well.



(a)



(b)

Figure 1: Operation of (a) a unicast and (b) a broadcast transmission in ContikiMAC

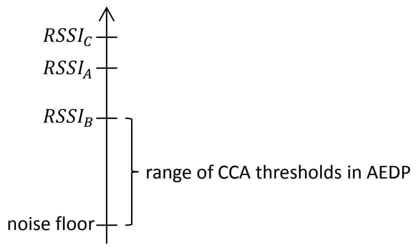


Figure 2: AEDP caps CCA thresholds to fit in between the minimum RSSI among all incoming links and the floor noise

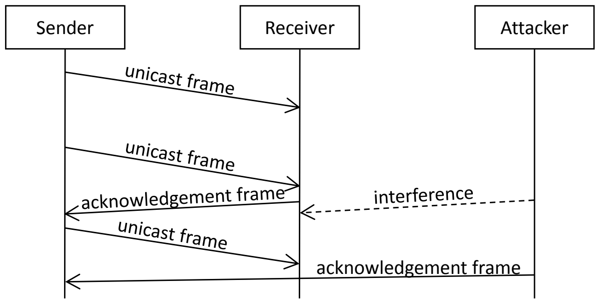
1.2 Three Denial-of-Sleep Attacks on ContikiMAC

While being easy to implement, ContikiMAC's CCAs incur two problems. First, it is crucial to set the CCA threshold properly. If the CCA threshold is too low, receivers often wake up unnecessarily, thereby expending their limited energy. Conversely, if the CCA threshold is too high, transmissions may go undetected [25]. Second, attackers may simply emit interference, which misleads ContikiMAC into staying in receive mode for extended periods of time and hence consuming much energy. Such attacks can also be carried out by injecting or replaying 802.15.4 frames [21]. We collectively refer to attacks that mislead ContikiMAC into staying in receive mode as ding-dong ditching.

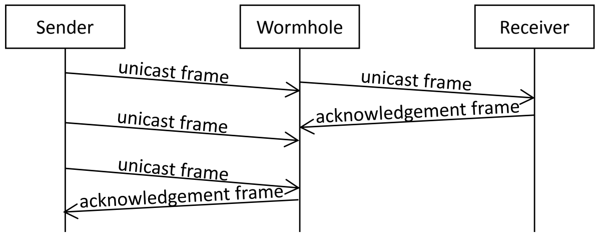
Sha et al. considered the first problem [25]. They showed that setting a static CCA threshold is insufficient to cope with false wake ups and undetected transmissions. Rather, the CCA threshold should be adapted dynamically at runtime. Accordingly, they proposed the Adaptive Energy Detection Protocol (AEDP), which adjusts the CCA threshold to trade off false wake ups against undetected transmissions.

At first glance, AEDP might mitigate ding-dong ditching since AEDP increases the CCA threshold as false wake ups occur. However, AEDP caps CCA thresholds to fit in between the minimum received signal strength indicator (RSSI) among all incoming links and the floor noise like shown in Figure 2. Thus, as long as attackers do ding-dong ditching with a strong enough transmission power, receivers still wake up. On the other hand, if ding-dong ditching causes the floor noise to increase beyond the RSSI of an incoming link, AEDP's behavior is undefined. In such cases, it is reasonable to set the CCA threshold slightly above the level of the floor noise. This is because (i) communication on links with lower RSSIs than the floor noise is error-prone anyway and (ii) it requires attackers to do ding-dong ditching with higher and higher transmission powers as otherwise victim nodes will not wake up. Nevertheless, attackers can bypass this defense by doing ding-dong ditching without affecting the level of the floor noise, such as by injecting or replaying 802.15.4 frames. Altogether, AEDP can not mitigate all kinds of ding-dong ditching.

Moreover, ding-dong ditching is not the only way to make ContikiMAC consume more energy. Alternatively, attackers can mislead ContikiMAC into staying more time in transmit mode. One way to do so is by launching pulse-delay attacks against ContikiMAC's phase-lock optimization. Such attacks come in two flavors [11]. First, an attacker can jam during the transmission of an acknowledgement frame and replay the jammed acknowledgement frame later on, as shown in Figure 3a. Second, an attacker can tunnel the traffic between two distant nodes verbatim, i.e. set up a hidden wormhole [4], and deliberately delay acknowledgement frames, as shown in Figure 3b. In either case, senders strobe more often, and, since ContikiMAC's phase-lock optimization learns the wake-up times of neighboring nodes based on



(a)



(b)

Figure 3: Pulse-delay attack via (a) interference and (b) a hidden wormhole

when acknowledgement frames are received, ContikiMAC's phase-lock optimization may need to relearn the wake-up time of the affected receiver, which expends additional energy. Another way to make ContikiMAC spend more time in transmit mode is via collision attacks. In collision attacks, an attacker prevents the reception of a unicast or acknowledgement frame by means of jamming [30]. In effect, as no acknowledgement frame is detected, ContikiMAC strobes for long. The incurred energy consumption increases further if the attacker interferes with subsequent retransmissions, too. To some extent, ContikiMAC's phase-lock optimization mitigates collision attacks by shortening the maximum duration of a strobe of unicast frames once a receiver's wake-up time is known. Yet, for handling clock drift, ContikiMAC's phase-lock optimization resorts to strobe unicast frames to a receiver for a whole wake-up interval plus once if unicast transmissions to the receiver tend to fail. Attackers can hence aggravate collision attacks by launching them repeatedly.

1.3 Contributions

To counter ding-dong ditching, pulse-delay attacks, as well as collision attacks, we propose the dozing optimization and the secure phase-lock optimization:

- The dozing optimization inserts sleeping periods into the time span between a negative CCA and the detection of a frame. This way, the dozing optimization mitigates all kinds of ding-dong ditching. Beyond that, the dozing optimization helps during normal operation as it reduces the energy consumption of true wake ups, too.
- The secure phase-lock optimization is a version of ContikiMAC's phase-lock optimization that resists pulse-delay attacks. Additionally, the secure phase-lock optimization obviates the need for relearning wake-up times if clocks drift apart. Instead, senders only need to learn a receiver's wake-up time once in the course of session key establishment. Within a session, senders then strobe unicast frames as often as is reasonable according to the current uncertainty about the phase offset in relation to the receiver. This greatly mitigates collision attacks.

1.4 Threat Model

For the scope of this paper, we assume an external attacker, i.e. an attacker that has not compromised a single node of the victim network and does not possess any keys of any node of the victim network [11]. Still, external attackers can inject, replay, modify, and jam frames and thus launch sophisticated attacks, such as hidden wormhole and pulse-delay attacks. An internal attacker, by contrast, has compromised one or more nodes of the victim network and possesses their keys [11]. Hence, internal attackers can carry out further attacks compared to external attackers. We leave an extension of this paper to internal attackers for future work.

2 Related Work

This section sums up related work on ding-dong ditching, pulse-delay attacks, and collision attacks. For an overview of alternatives to ContikiMAC and their security issues, we refer to [13] and [31, 18], respectively.

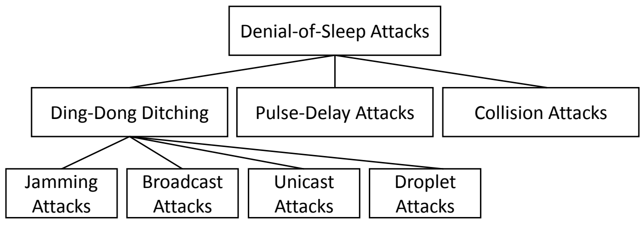


Figure 4: Taxonomy of Denial-of-Sleep Attacks

2.1 Ding-Dong Ditching

Ding-dong ditching belongs to the broader class of denial-of-sleep attacks, which generally cause an increased energy consumption on victim nodes [3]. As depicted in Figure 4, our term ding-dong ditching subsumes various denial-of-sleep attacks, namely jamming, broadcast, unicast, and droplet attacks. In jamming attacks, attackers constantly or intermittently emit interference [21]. In broadcast attacks, attackers inject or replay broadcast frames [3]. Likewise, in unicast attacks, attackers inject or replay unicast frames. In droplet attacks, an attacker only injects or replays the beginning of 802.15.4 frames and then stops transmitting. Nevertheless, receivers will detect such frames and stay in receive mode until the pretended end of such frames [12].

Raymond et al. devised a reactive defense against broadcast attacks, entitled Clustered Adaptive Rate Limiting (CARL) [22]. CARL detects broadcast attacks by caching whether recent wake ups led to receiving an inauthentic or replayed broadcast frame. In response to broadcast attacks, CARL prolongs the wake-up interval. Furthermore, Raymond et al. addressed the problem that prolonging the wake-up interval should be done in a coordinated manner as senders need to be aware of the wake-up interval of receivers. To this end, they suggest that nodes do not prolong their wake-up interval immediately when they detect a broadcast attack. Instead, nodes that detected a broadcast attack and have a frame to send, send this frame and only thereafter prolong their wake-up interval. Other nodes that also detected a broadcast attack wait until they receive a frame and thereupon prolong their wake-up intervals. However, if a node does not receive any frame, the node eventually prolongs its wake-up interval after a timeout. Unfortunately, this way of coordinating the response to broadcast attacks (i) defers the reaction to broadcast attacks and (ii) does not exclude the possibility that some nodes temporarily use different wake-up intervals than other nodes, thereby potentially causing an increased energy consumption due to retransmissions.

Krentz et al. proposed a mitigation technique against broadcast, unicast, and droplet attacks, named Practical On-the-fly Rejection (POTR) [16]. The basic approach of POTR is to embed one-time passwords (OTPs) in the headers of 802.15.4 frames. This enables POTR to cancel the reception of injected, replayed, and overheard 802.15.4 frames early on. However, POTR does not leave the receive mode during the time span between a negative CCA and the detection of a frame. During this time span, ContikiMAC still only leaves the receive mode if the fast-sleep optimization decides so. Hence, our dozing optimization complements POTR very well. In fact, in Section 6.1, we demonstrate that POTR and the dozing optimization better mitigate ding-dong ditching than any of these two mitigation techniques alone.

2.2 Pulse-Delay Attacks

Song et al. defined delay attacks as attacks where an "attacker deliberately delays some of the time messages [...] so as to fail the time synchronization process" [26]. Especially internal attackers can launch delay attacks since compromised nodes can send time messages at will. Moreover, Ganeriwal et al. found that external attackers can launch so-called pulse-delay attacks like explained in the introduction. In order to counter pulse-delay attacks, Ganeriwal et al. suggested setting an upper bound on the round-trip time [11]. Our secure phase-lock optimization refines this approach.

2.3 Collision Attacks

Ren et al. put forward a reactive defense against collision attacks [23]. Specifically, Ren et al. use an intricate threshold rule to detect collision attacks, as well as many other denial-of-sleep attacks. If a node detects any denial-of-sleep attack, they suggest switching to a low-power sleep mode for some time. However, unlike Raymond et al., Ren et al. neglected the problem that transmissions fail if a receiver is currently reacting to denial-of-sleep attacks. Moreover, if a receiver is sleeping, a sender may detect a collision attack due to retransmissions and, if so, enter a low-power sleep mode for some time, too. Thus, a single denial-of-sleep attack may propagate through a whole network, thereby potentially aggravating denial-of-sleep attacks. We avoid such issues since our approach is to keep the energy consumption of unicast transmissions low. For this, we estimate the uncertainty about phase offsets and adapt the maximum number of strobed unicast frames accordingly. While this idea already came up [10], we seem first to tailor it to ContikiMAC, as well as to consider pulse-delay attacks in this context.

3 The Dozing Optimization

In the following, we will use the following variables:

- t_l is the time to transmit a maximum-length 802.15.4 frame of 127 bytes. When using 802.15.4 channels in the 2.4-GHz band, t_l is 4.256ms. This is because 802.15.4 has a transmission rate of 250kbit/s in the 2.4-GHz band, and because each frame is prefixed with a 5-byte synchronization header (SHR) and a 1-byte Frame Length field.
- t_r is the duration of one CCA. When using 802.15.4 channels in the 2.4-GHz band, t_r is 0.128ms since 802.15.4 specifies that CCAs shall take 8 symbol periods in the 2.4-GHz band.
- t_c is the configurable time span in between the two regular CCAs, as depicted in Figure 5.
- t_i is ContikiMAC's inter-frame period, as shown in Figure 5. Though t_i is configurable, t_i can not be set arbitrarily low to allow for sending and detecting acknowledgement frames in between successively strobed unicast frames. Neither can t_i be set arbitrarily high as otherwise the two regular CCAs of ContikiMAC may fall in between successively strobed frames.

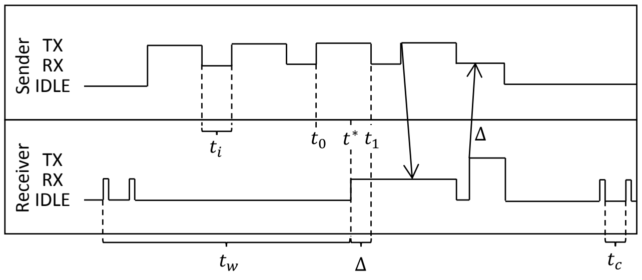


Figure 5: Illustration of t_c, t_i, t_0, t_1, t^*, t_w, and Δ

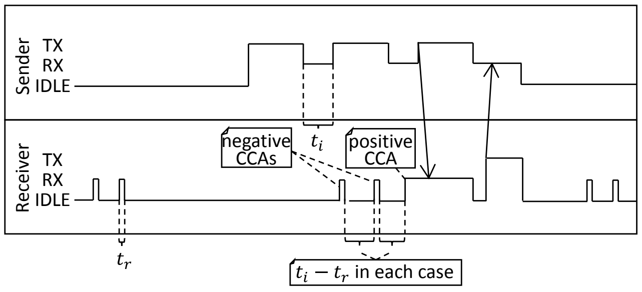


Figure 6: Operation of a unicast transmission with dozing enabled

- t_d is the time to detect an SHR. Assuming a transmission rate of 250kbit/s, t_d is 0.16ms.
- t_p is the time that POTR needs to decide if a detected frame is to be rejected. The actual value of t_p depends on the length of addresses, the length of OTPs, as well as on whether using 802.15.4 channels in the 2.4-GHz band or sub-GHz band. According to experimental results, t_p can get as low as 0.253ms [16].

Recall that ContikiMAC's fast-sleep optimization lets receivers go back to sleep (i) after t_l if the radio noise lasts longer than t_l, (ii) after $t_l + t_i$ if the silence period takes longer than t_i, and (iii) after $t_l + t_i + t_d$ if no SHR is detected. Attackers can misuse this behavior by emitting interference. Moreover, an attacker can act like a normal sender and inject a frame, causing receivers to stay in receive mode as long as indicated by the Frame Length field. This is where POTR helps. POTR rejects injected, replayed, and overheard frames t_p after the detection of an SHR. Altogether, in the worst case, an attacker can cause ContikiMAC to stay in receive mode for $t_r + t_r + t_l + t_i + t_d + t_p = w_{\text{original}}(t_i)$, where "$t_r+$" accounts for the occasion when the first of the two regular CCAs returns positive and "$+t_r+$" allows for a minimal overlap between the ending of the second CCA and the beginning of a frame. Analogously, the maximum time that ContikiMAC stays in receive mode for receiving a legitimate frame is $w_{\text{original}}(t_i) - t_p + t_l$.

The operation of the dozing optimization is exemplified in Figure 6. As opposed to the original design of ContikiMAC, if a CCA returns negative, the dozing optimization goes back to sleep and schedules another CCA after $t_i - t_r$. If this CCA also returns negative, the dozing optimization goes back to sleep again and performs another CCA after $t_i - t_r$ and so on.

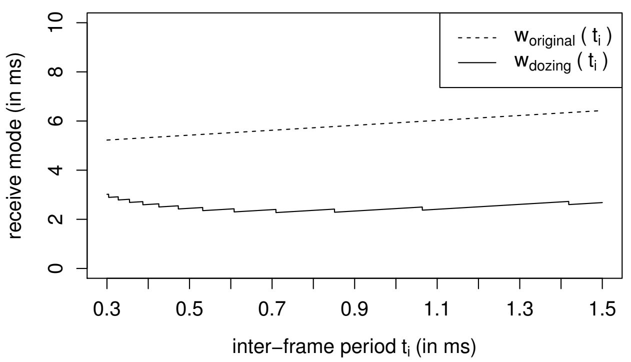


Figure 7: Worst-case duration that ContikiMAC stays in receive mode under ding-dong ditching ($t_l = 4.256$ms, $t_r = 0.128$ms, $t_d = 0.16$ms, $t_p = 0.253$ms)

In accordance with the fast-sleep optimization, if the radio noise lasts longer than t_l, the dozing optimization schedules no further CCAs. If, however, a subsequent CCA returns positive, the dozing optimization stays in receive mode as this indicates that a silence period in between two successively strobed frames is found. Also, in accordance with the fast-sleep optimization, a receiver goes back to sleep if no radio noise is detected after $t_l + t_i$ and if no SHR is detected after $t_l + t_i + t_d$. Essentially, the dozing optimization thus mimics the fast-sleep optimization except for dozing after a negative CCA when searching a silence period. Therefore, the dozing optimization has no adverse effects on reliability.

Observe that, if t_i is long, the dozing optimization dozes longer and hence performs less CCAs, while, if t_i is short, the dozing optimization needs to spend less time in receive mode once a CCA returns positive. Concretely, in the adversarial case, the dozing optimization performs up to $2 + \lceil \frac{t_l}{t_i} \rceil + 1$ CCAs and stays at most $t_i + t_d + t_p$ in receive mode. Again, "2+" accounts for a positive first CCA and a negative second CCA. Then, up to $\lceil \frac{t_l}{t_i} \rceil$ negative CCAs and one positive CCA may follow. Next, the dozing optimization stays up to $t_i + t_d + t_p$ in receive mode. Altogether, the worst-case time in receive mode in the adversarial case is $(3 + \lceil \frac{t_l}{t_i} \rceil) \times t_r + t_i + t_d + t_p = w_{\text{dozing}}(t_i)$. Similarly, upon receiving a legitimate frame, the dozing optimization stays in receive mode for at most $w_{\text{dozing}}(t_i) - t_p + t_l$.

As shown in Figure 7, $w_{\text{dozing}}(t_i)$ has a minimum at $t_i = 0.7094$. Figure 7 also shows that, from a theoretical standpoint, the dozing optimization significantly reduces the time spent in receive mode when waking up. In Section 6.1, we will experimentally determine the actual energy savings.

4 The Secure Phase-Lock Optimization

Let us define the following variables like illustrated in Figure 5:

- t_0 is the time when the transmission of the next to last acknowledged frame from a sender to a receiver began.
- t_1 is the time when the transmission of the next to last acknowledged frame from a sender to a receiver ended.
- t^* is the time when a receiver woke up when receiving the last acknowledged frame from a sender.
- t_w is ContikiMAC's wake-up interval.

The original phase-lock optimization schedules the start of a strobe of unicast frames right before the intended receiver wakes up. For this, the original phase-lock optimization exploits the fact that if an acknowledgement frame is received, the next to last strobed unicast frame must have been transmitted while the receiver did a CCA. As another strobe of unicast frames to the receiver shall be sent, the original phase-lock optimization starts strobing at $t_0 + t_w \times n - t_g$ and only strobes once more after $t_0 + t_w \times n - t_g + t_h$, where t_g and t_h are configurable time spans, and the integer n is chosen so that $t_0 + t_w \times n - t_g$ is in the future.

4.1 Securing Acknowledgement Frames

In contrast to the original phase-lock optimization, the secure phase-lock optimization ensures both the timeliness and authenticity of acknowledgement frames. Doing so is desirable for two reasons. On the one hand, if a sender accepted delayed acknowledgement frames, its phase-lock optimization might save less energy than usual, as discussed in the introduction. On the one hand, if a sender accepted unauthenticated acknowledgement frames, an attacker could prevent a receiver from receiving a unicast frame by injecting an acknowledgement frame before the receiver wakes up. Moreover, the sender of the unicast frame would wrongly believe that the unicast frame was successfully received [24].

4.1.1 Authenticity

For ensuring the authenticity of acknowledgement frames, the secure phase-lock optimization adds a message integrity code (MIC) to each acknowledgement frame. These MICs are generated using the authenticated encryption with associated data (AEAD) algorithm CCM*, which is also used for securing other 802.15.4 frames [1]. CCM* takes a 128-bit key, a 13-byte nonce, data to authenticate and encrypt, as well as the desired MIC length as inputs, and outputs a MIC, as well as the encrypted data if any. As key, we use the same as was used for securing the unicast frame whose receipt is being acknowledged. As nonce, we concatenate the receiver's address, the strobe index of the unicast frame whose receipt is being acknowledged (details follow in Section 4.1.2), the frame counter of the unicast frame whose receipt is being acknowledged, and, to differentiate such nonces from others, 0xfe [15]. As data, we require the contents of acknowledgement frames to be authenticated. Finally, as MIC length, we, again, use the same as was used in the unicast frame whose receipt is being acknowledged.

Yet, before sending an authenticated acknowledgement frame, a receiver should check the authenticity of the received unicast frame. Otherwise, the following attack trace would be possible. Let A and B be neighboring nodes. An attacker can (i) send a unicast frame with A's address as source address, some frame counter c, and some strobe index i to B, (ii) capture the authentic acknowledgement frame that B replies with, (iii) wait until A sends a unicast frame with frame counter c and strobe index i to B, and (iv) replay the captured authentic acknowledgement frame before B wakes up. As a result, A wrongly thinks that B received A's unicast frame with frame counter c and strobe index i.

It is, however, problematic to check the authenticity of a received unicast frame before replying with an authenticated acknowledgement frame since ContikiMAC's inter-frame period is pretty short. To this end, we propose four measures, which accelerate the transmission of authenticated acknowledgement frames. A first measure is, e.g., to check the authenticity of a received unicast frames directly within an interrupt context, rather than waiting for other processes to finish. A second measure is to exploit the common feature of 802.15.4 transceivers to accelerate CCM* operations in hardware. A third measure is to prepare authenticated acknowledgement frames already during the reception of a unicast frame. Finally, a forth measure is to start the transmission of an authenticated acknowledgement frame before the authenticity of a received unicast frame is checked. Yet, this only works if the transmission of the authenticated acknowledgement frame can be aborted early enough if the unicast frame turns out to be inauthentic. Early enough here means that the abortion must happen before the MIC of the authenticated acknowledgement frame is being transmitted.

Another issue with authenticating acknowledgement frames is that session keys are not in place while establishing session keys. In fact, the session key establishment protocol that we use in our implementation, namely the Adaptive Key Establishment Scheme (AKES) [15], involves two unicast transmissions. To resolve this conflict, we integrated a special method for authenticating acknowledgement frames into AKES, which we will specify in Section 4.3.

4.1.2 Timeliness

For ensuring the timeliness of acknowledgement frames, the secure phase-lock optimization employs two complementary mechanisms. First, it inserts a 1-byte Strobe Index field into the headers of unicast frames. The Strobe Index field indicates how often a unicast frame was strobed already. Further, since the Strobe Index field changes in each consecutively strobed unicast frame, CCM* has to be rerun over each consecutively strobed unicast frame. To avoid a nonce reuse in this process, the secure phase-lock optimization incorporates the strobe index into the CCM* nonce of unicast frames, too. Second, the secure phase-lock optimization confines the reception window for acknowledgement frames by setting an upper bound and, deviating from Ganeriwal et al.'s Secure Pairwise Synchronization (SPS) protocol [11], also a lower bound on the round-trip time. We use t_a to denote the duration of this reception window.

These two mechanisms can not prevent pulse-delay attacks per se, but enable us to upper-bound the maximum delay due to pulse-delay attacks by t_a. This is because (i) an authentic acknowledgement frame must belong to the unicast frame that was just sent and (ii) the duration of the reception window for acknowledgement frames is t_a. (i) holds, on the one hand, because the CCM* nonce of an authenticated acknowledgement frame includes both the strobe index and the frame counter of the unicast frame whose receipt is being acknowledged and, on the other hand, because the authenticity of unicast frames is being checked before replying with authenticated acknowledgement frames.

4.2 Bounding Strobes of Unicast Frames

While in the original phase-lock optimization the guard time t_g is static, the secure phase-lock optimization splits t_g into a static portion and a dynamic portion. The static portion, denoted by t_s, should account for inaccuracies. Additionally, t_s must accommodate pulse-delay attacks and therefore $t_s > t_a$. The dynamic portion, denoted by t_u, can be chosen according to the current uncertainty about the phase offset in relation to the intended receiver. This uncertainty can be upper bounded as follows. Let θ be the frequency tolerance of the employed clocks and let t be the current time. Then, $t_u = (t - t_0) \times (\theta + \theta)$ [10].

The above consideration also implies when the intended receiver must have woken up, namely before $t_1 + t_w \times n + (t_s + t_u)$. Hence, when starting a strobe of unicast frames at $t_0 + t_w \times n - (t_s + t_u)$, only one additional unicast frame must be strobed after $t_1 + t_w \times n + (t_s + t_u)$. That is, we can limit the maximum duration of a strobe of unicast frames like the original phase-lock optimization, but without needing a fallback mechanism if unicast transmissions tend to fail.

To keep t_u low and therefore strobes of unicast frames short, we can send keep-alive messages. However, in our implementation, keep-alive messages are sent by AKES anyway to detect if a neighbor got out of range [15], thus obviating the need for sending dedicated keep-alive messages.

Note that t_0 and t_1 serve as lower and upper bounds of the true wake-up time t^*. To allow for a better estimation of t^*, the secure phase-lock optimization reports back on $\Delta = t_1 - t^*$ to senders by piggybacking Δ on acknowledgement frames, similar to what was suggested by Michel et al. [19]. With the knowledge of Δ, the secure phase-lock optimization starts strobing at $t^* + t_w \times n - (t_s + t_u^*)$, where $t_u^* = (t - t^*) \times (\theta + \theta)$. Analogously, the secure phase-lock optimization only strobes one additional unicast frame after $t^* + t_w \times n + (t_s + t_u^*)$. This interval is more narrow than $[t_0 + t_w \times n - (t_s + t_u), t_1 + t_w \times n + (t_s + t_u)]$, thereby mitigating collision attacks even better.

In the exceptional case that $t_s + t_u^* \geq \frac{t_w}{2}$, the secure phase-lock optimization starts strobing unicast frames instantly for at most a whole wake-up interval plus once. This is because this uncertainty is unreasonably large. Usually, this condition should not take effect as keep-alive messages should be sent early enough so that $t_s + t_u^*$ is always well below $\frac{t_w}{2}$.

4.3 Initializing Wake-Up Times

Figure 8a sketches how AKES establishes group session keys. Initially, a node A broadcasts a `HELLO`, containing a random number R_A. Any receiver B that has not yet established session keys with A also generates a random number R_B. Then, after a random back off period, B replies with a `HELLOACK`, carrying R_B, B's group session key $K_{B,*}$ encrypted, as well as a MIC. For encrypting $K_{B,*}$ and generating the MIC, B derives a temporary pairwise key $K'_{A,B}$ from a predistributed shared secret $K_{A,B}$ between A and B, as well as the two random numbers R_A and R_B. Upon receipt of B's `HELLOACK`, A decrypts $K_{B,*}$ and checks, amongst others, the MIC by deriving $K'_{A,B}$ analogously. If successful, A sends an `ACK` to B, which includes A's group session key $K_{A,*}$ encrypted and a MIC. Likewise, A encrypts $K_{A,*}$ and generates

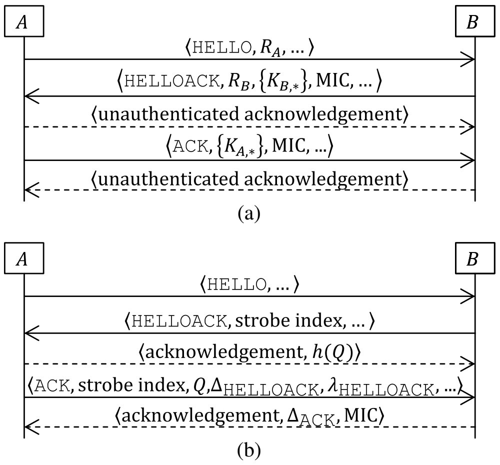

A
B
⟨HELLO, R_A, ...⟩
⟨HELLOACK, R_B, $\{K_{B,*}\}$, MIC, ...⟩
⟨unauthenticated acknowledgement⟩
⟨ACK, $\{K_{A,*}\}$, MIC, ...⟩
⟨unauthenticated acknowledgement⟩
(a)
A
B
⟨HELLO, ...⟩
⟨HELLOACK, strobe index, ...⟩
⟨acknowledgement, $h(Q)$⟩
⟨ACK, strobe index, Q, Δ_{HELLOACK}, $\lambda_{\text{HELLOACK}}$, ...⟩
⟨acknowledgement, Δ_{ACK}, MIC⟩
(b)

Figure 8: Establishment of group session keys (a) as per AKES and (b) like adapted by the secure phase-lock optimization to securely initialize wake-up times in parallel

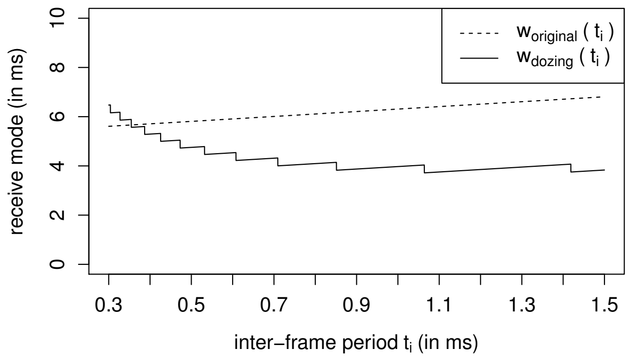


Figure 9: Worst-case duration that ContikiMAC stays in receive mode under ding-dong ditching ($t_l = 4.256$ms, $t_r = 0.32$ms, $t_d = 0.16$ms, $t_p = 0.253$ms)

the MIC using the temporary pairwise key $K'_{A,B}$ once more.

To enable A and B to securely learn each other's wake-up time t^* in parallel to establishing group session keys, the secure phase-lock optimization authenticates acknowledgement frames and adds additional data to HELLOACKs and ACKs like shown in Figure 8b. Specifically, as A receives a HELLOACK, A generates a random number Q, and acknowledges by replying $h(Q)$, where h is a one-way function. This acknowledgement frame is sent immediately without executing AKES's checks. As B receives $h(Q)$, B stops strobing and stores $h(Q)$, as well as the current strobe index λ. Next, if the HELLOACK passes AKES's checks, A sends an ACK to B, which includes Q, Δ_{HELLOACK}, and the received strobe index $\lambda_{\text{HELLOACK}}$ of the HELLOACK. Upon receipt of A's ACK, B ensures, in addition to AKES's checks, that $h(Q) = Q$ and that $\lambda_{\text{HELLOACK}} = \lambda$. If the ACK passes all checks, B calculates and stores t^* by means of Δ_{HELLOACK}. If, however, any check fails, B ignores subsequent ACKs from A until retrying session key establishment all over again. In any case, prior to executing these checks, B acknowledges A's ACK by sending an acknowledgment frame that is authenticated using AKES's temporary pairwise key between A and B. Upon receipt of this acknowledgement frame, A checks its authenticity and timeliness, and, if successful, calculates and stores t^*. Otherwise, if A's ACK remains unacknowledged, A aborts session key establishment with B.

Owing to acknowledging HELLOACKs and ACKs immediately without executing checks, the sender of a HELLOACK or ACK may wrongly think that its HELLOACK or ACK was successfully received, respectively. Consequently, session key establishment may fail due to this. However, session key establishment may also fail due to jamming attacks. In Section 6.2.1, we will discuss the severity of such an occasion.

5 Implementation

Unfortunately, the current implementation of ContikiMAC, which is part of the Contiki operating system [7], is subject to two issues. On the one hand, the current implementation blocks all other processes while sending frames. On the other hand, Uwase et al. reported timing issues [28]. In view of these issues, we opted for implementing the dozing and the secure phase-lock optimization as part of a whole new implementation of ContikiMAC. Our target platform are OpenMotes - a CC2538-based platform [29, 27].

In contrast to the current implementation of ContikiMAC, our reimplementation allows other processes to progress while sending and receiving frames. For this, we make use of various interrupts, as well as the rtimer module of Contiki. For example, we use the rtimer module for scheduling wake ups, transmissions, dozing, and other events. As for OpenMotes, the rtimer module is implemented on top of the sleep timer of the CC2538. The sleep timer makes either use of a 32,768Hz-RC oscillator or an external 32,768Hz-crystal oscillator. We select the 32,768Hz-crystal oscillator of OpenMotes. It has a frequency tolerance of 15ppm [29].

Furthermore, in our reimplementation, all timing issues appear to be solved. However, unlike specified in 802.15.4, the CC2538 needs 0.32ms per CCA. This is because the CC2538 stays 0.192ms in an energy-consuming intermediate state before it actually starts with a CCA. This affects the functions $w_{\text{original}}(t_i)$ and $w_{\text{dozing}}(t_i)$ in Figure 7. Figure 9 shows the updated functions. Now, $w_{\text{dozing}}(t_i)$ has a minimum at 1.064ms. Thus, 1.064ms constitutes the optimal value for t_i in terms of worst-case duration in receive mode. However, owing to the limited precision of the 32,768Hz-crystal oscillator, our reimplementation rounds up to $t_i = 1.068$ms. Other timing-related defaults are $t_c = 0.854$ms (time between the two regular CCAs), $t_w = 125$ms (wake-up interval), $t_a = 0.122$ms (reception window for acknowledgement frames), $t_s = 0.183$ms (static guard time), and $\theta = 15$ppm (frequency tolerance).

Figure 10 depicts how our reimplementation of ContikiMAC integrates into Contiki's network stack. At the RADIO layer, we use our version of the cc2538_rf_driver. Our version deviates from the existing one in that it (i) supports listening for interrupts, (ii) only contains code that is needed by our reimplementation of ContikiMAC, (iii) provides access to the RXFIFO as required by POTR, and

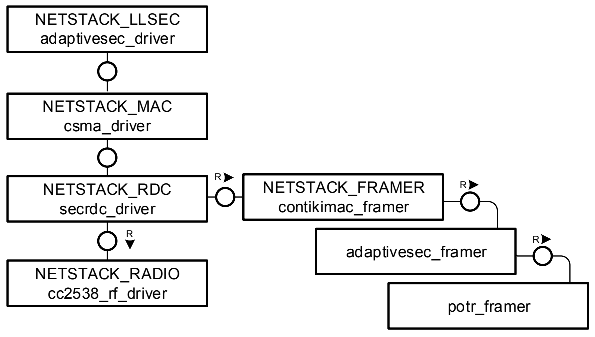


Figure 10: Integration of our reimplementation of ContikiMAC into Contiki's network stack

(iv) supports overwriting bytes in the `TXFIFO`, which we, e.g., use for setting the current strobe index. At the `RDC` layer, the `secrdc_driver` contains our reimplementation of ContikiMAC. Optionally, the `secrdc_driver` can also be configured to act like the original version of ContikiMAC. The `secrdc_driver` communicates with the `contikimac_framer`, which potentially adds and removes padding bytes to avoid that a frame can fall in between the two regular CCAs of ContikiMAC. The `contikimac_framer` calls the `adaptivesec_framer`, which, in turn, calls the `potr_framer`. Alternatively, we also support to disable POTR, in which case an 802.15.4-compliant `framer` is called instead of the `potr_framer`. Yet, the secure phase-lock optimization can only be enabled if POTR is also enabled. At the `MAC` layer, Contiki's `csma_driver` schedules retransmissions in adherence to 802.15.4. At the `LLSEC` layer, we run the `adaptivesc_driver`, which implements both AKES and 802.15.4 security [15]. On top of the `LLSEC` layer, upper-layer protocols can be run, such as the 6LoWPAN adaptation layer, which compresses and decompresses IPv6 packets, and potentially fragments and reassembles them [14].

6 Evaluation

Using our reimplementation of ContikiMAC, we now assess the effectiveness and efficiency of both of our optimizations. In particular, we demonstrate that the dozing optimization (i) noticeably reduces the energy consumption of true wake ups, (ii) significantly reduces the energy consumption under ding-dong ditching, and (iii) practically incurs no overhead in program memory. Subsequently, we show that the secure phase-lock optimization (i) mitigates collision attacks greatly, (ii) resists pulse-delay attacks, (iii) makes the reception and transmission of unicast frames a bit more energy consuming, and (iv) entails a moderate overhead in program memory.

6.1 The Dozing Optimization

6.1.1 *Energy Consumption of True Wake Ups*

To compare the energy consumption of OpenMotes during frame receptions with versus without the dozing optimization, an OpenMote A sent 100-byte broadcast frames at the rate of 8Hz. Another OpenMote B was placed 50cm away from A and woke up at a randomized rate. During 100 frame receptions, the current draw of B was recorded

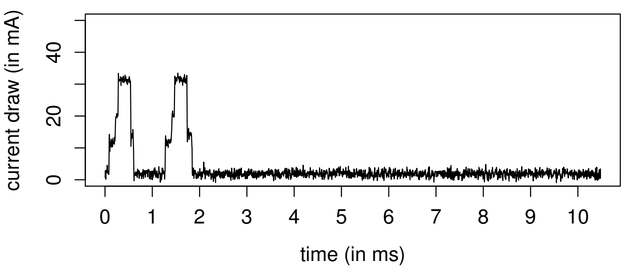


Figure 11: Current draw during ContikiMAC's two regular CCAs

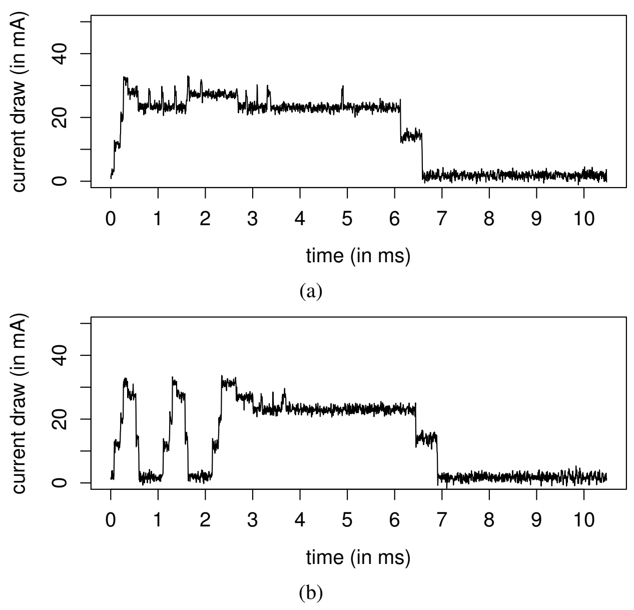
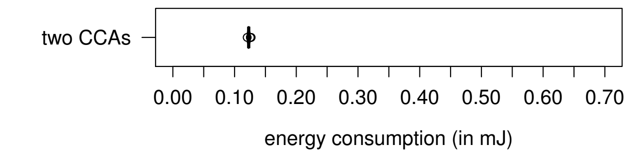


Figure 12: Current draw during a true wake-up when the dozing optimization is (a) off and (b) on

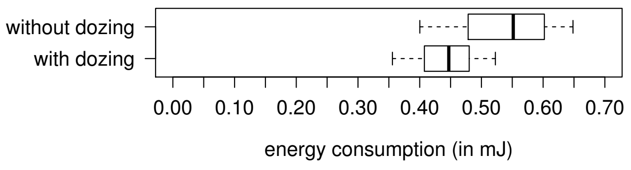
by connecting B, a μCurrent Gold, and a Rigol DS1000E oscilloscope in series. This method of measuring the current draw of OpenMotes is further detailed in [29]. Measurements were triggered by setting B's `AD3/DIO3` pin high 1ms before each wake up. This experiment was conducted two times for the cases of (i) disabling the dozing optimization and (ii) enabling the dozing optimization. Furthermore, as a baseline for comparison, 100 traces of B's current draw during ContikiMAC's two regular CCAs were recorded, too. POTR and upper-layer protocols were disabled throughout.

Figure 11 exemplifies the current draw during ContikiMAC's two regular CCAs. The area under the curve matches the consumed charge. According to Simpson's rule, the area in this example is 0.0423mAs. This measure can be used to calculate the energy consumption by multiplying by the supply voltage. In the following, we assume a constant supply voltage of 3V, yielding an energy consumption of $0.0423\text{mAs} \times 3\text{V} = 0.127\text{mJ}$ in this example.

Figure 12a exemplifies a frame reception, where the dozing optimization is off. Here, the first of the two regular



(a)



(b)

Figure 13: Energy consumption (a) for two CCAs and (b) per reception of a 100-byte broadcast frame

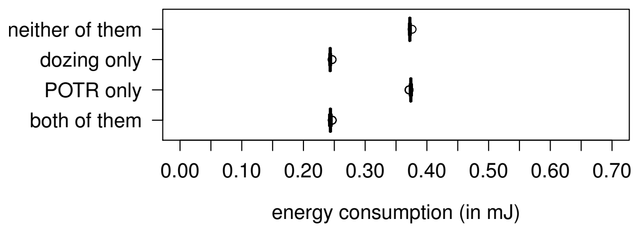
CCAs returns negative, which is why *B* stays in receive mode and periodically performs CCAs so as to find a silence period. As *B* finds a silence period at time 2ms, *B* stops doing CCAs and enables the SHR search of the CC2538. It would also be possible to enable the SHR search of the CC2538 right from the beginning, but then chances are that an SHR is found within radio noise or the previously transmitted frame [25]. Then, *B* stays in receive mode to receive the approaching 100-byte broadcast frame. Once the frame is received, *B* processes it and finally enters a low-power sleep mode.

Figure 12b exemplifies a frame reception, where the dozing optimization is on. Since the first of the two regular CCAs returns negative, *B* starts dozing. The second CCA also returns negative, which is why *B* continues to doze. Eventually, the third CCA returns positive, causing *B* to stay in receive mode. At this point, *B* also enables the SHR search of the CC2538. Again, one could enable the SHR search of the CC2538 in the first place. Despite this, we disable the SHR search of the CC2538 during CCAs. This is not because we want to avoid detecting SHRs (which is unnecessary since the dozing optimization disables the receive mode after negative CCAs anyway), but because the CC2538 appears to work more reliably this way. Lastly, *B* processes the received 100-byte broadcast frame and goes back to sleep.

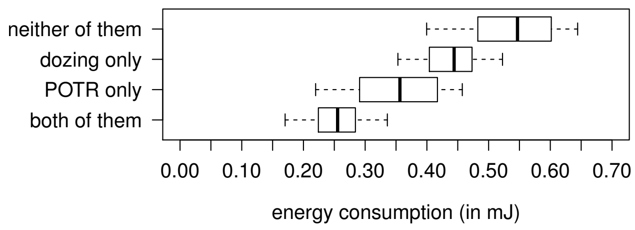
Figure 13 shows boxplots of the energy consumption per wake up. The baseline energy consumption, i.e. when just doing two CCAs, is very low. However, if a frame comes in, the energy consumption is high. Enabling the dozing optimization saves a good amount of energy.

6.1.2 *Energy Consumption under Ding-Dong Ditching*

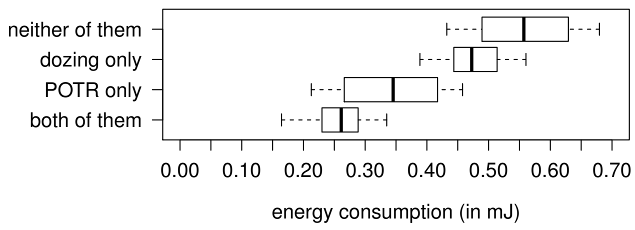
To determine the energy consumption under jamming, broadcast, and unicast attacks, the above experiment was adapted as follows. As for jamming attacks, *A* continuously emitted radio noise via a special transmit mode of its CC2538. In four subsequent runs, *B* was programmed to use (i) neither POTR nor the dozing optimization, (ii) not POTR, but the dozing optimization, (iii) POTR, but not the dozing optimization, and (iv) both POTR and the dozing optimization. In each run, 100 traces of *B*'s current draw during wake



(a)



(b)



(c)

Figure 14: Energy consumption per wake up under (a) jamming, (b) broadcast, and (c) unicast attacks

ups were recorded. As for broadcast attacks, the procedure was the same, except that *A* continuously strobed a 100-byte broadcast frame that is rejected by *B* due to an invalid MIC, or rather an invalid OTP. Likewise, as for unicast attacks, *A* continuously strobed a 100-byte unicast frame that is rejected by *B* due to an invalid MIC, or rather an invalid OTP.

Figure 14a shows the results for jamming attacks. Without any defense, ContikiMAC's fast sleep optimization leaves the receive mode only after $t_l = 4.256$ms. This causes a mean energy consumption of 0.372mJ. By contrast, when using the dozing optimization, jamming attacks become much less severe, especially when considering that the mean energy consumption of doing two CCAs already is 0.123mJ, as shown in Figure 13. Expectably, POTR does not help in mitigating jamming attacks since POTR only comes into effect once an SHR is detected.

Figure 14b shows the results for broadcast attacks. If using no defense or only the dozing optimization, we get almost the same energy consumption as in Figure 13. On the one hand, this is because even frames with inauthentic MICs are fully received and validated before they get rejected. On the other hand, we disabled upper-layer protocols in both experiments. Normally, the processing of authentic frames is more energy consuming. Enabling POTR mitigates broadcast attacks significantly since POTR rejects unwanted frames on the fly. Even more energy can be saved by enabling both POTR and the dozing optimization.

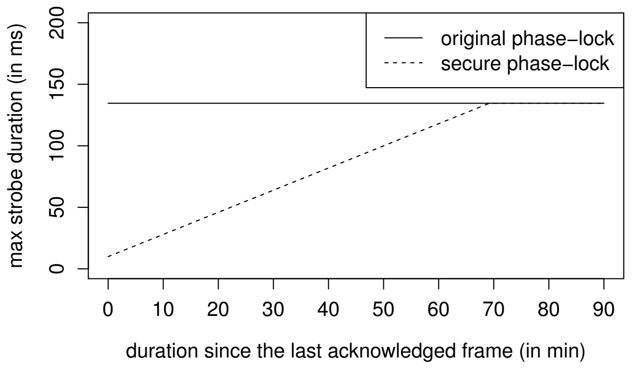


Figure 15: Maximum duration of a strobe of unicast frames ($t_l = 4.256$ms, $t_i = 1.068$ms, $t_s = 0.183$ms, $\theta = 15$ppm)

Figure 14c shows the results for unicast attacks. In contrast to broadcast attacks, unicast attacks may cause victim nodes to send acknowledgement frames. This actually happened in our experiment because, if POTR is off, even injected and replayed unicast frames are being acknowledged if they pass basic validity checks. Therefore, unicast attacks cause a higher energy consumption than broadcast attacks if POTR is off. If POTR is on, the 100-byte unicast frames are rejected on the fly and hence no acknowledgement frames are sent. Unicast attacks can also be further mitigated by enabling the dozing optimization in addition.

6.1.3 *Overhead in Program Memory*

The overhead in program memory of the dozing optimization was measured with the tool `arm-none-eabi-size`. Surprisingly, we found that enabling the dozing optimization incurs a marginal overhead in program memory of 8 bytes. The reason for this becomes apparent in the code. In fact, the dozing optimization requires only minor changes to the original version of ContikiMAC.

6.2 The Secure Phase-Lock Optimization

6.2.1 *Mitigation of Collision Attacks*

When using the original phase-lock optimization, senders resort to strobe unicast frames for a whole wake-up interval plus once if unicast transmissions to the intended receiver tend to fail. Thus, in the worst case, the duration of a strobe of unicast frames is $t_w + t_l + t_i + t_l$, where "$+t_l+$" allows for the situation that the transmission of the next to last unicast frame may have begun right before t_w elapsed.

Conversely, when using the secure phase-lock optimization, the severity of collision attacks is much lower. This is because the maximum duration of a strobe of unicast frames (other than `HELLOACK`s or `ACK`s) is $\min\{2 \times t_s + 2 \times t_u^*, t_w\} + t_l + t_i + t_l$. Taking the minimum over $2 \times t_s + 2 \times t_u^*$ and t_w accounts for the fact that the secure phase-lock optimization strobes unicast frames instantly for a full wake-up interval plus once if $t_s + t_u^*$ becomes greater or equal than $\frac{t_w}{2}$. Figure 15 shows how this duration compares to that of the original phase-lock optimization. Clearly, if the last acknowledged frame was sent a short time ago, the secure phase-lock optimization mitigates collision attacks greatly. For mitigating collision attacks throughout a session, we rely on the keep-alive messages of AKES, as mentioned in Section 4.2. By default, we configure AKES to send a keep-alive message to a neighbor that sent no timely authentic acknowledgement frame for 5min. This reduces the maximum duration of a strobe of unicast frames within a session from 134.58ms to 18.95ms in our implementation. On the other hand, if an attacker jams frames that pertain to AKES's three-way handshake, AKES's three-way handshake may either fail or succeed. If AKES's three-way handshake fails, victim nodes may even save energy because of not sending upper-layer traffic thereafter. However, it remains to be investigated if, by preventing AKES from establishing session keys, attackers can also cause a higher energy consumption since this prevents the upper-layer routing protocol from using certain routes. If AKES's three-way handshake succeeds despite collision attacks, at least subsequent collision attacks are benign. Similarly, if an attacker jams keep-alive messages of AKES, AKES will delete the affected neighbor and, at some point, try to establish new session keys with that neighbor. However, up until establishing new session keys, no more unicast frames will be sent to the affected neighbor. Thus, again, victim nodes may actually save energy when attackers interfere with keep-alive messages of AKES.

6.2.2 *Resistance to Pulse-Delay Attacks*

In the introduction, we introduced two methods for launching pulse-delay attacks against the original phase-lock optimization. Using either method, attackers can cause the original phase-lock optimization to strobe unicast frames more often. Moreover, pulse-delay attacks may necessitate relearning the wake-up time of the affected receiver.

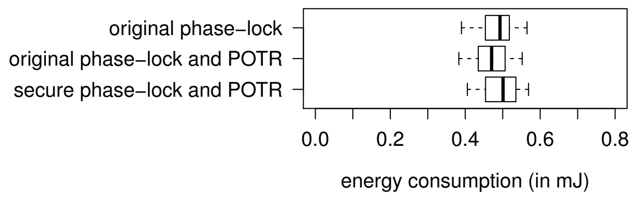
By contrast, our secure phase-lock optimization resists pulse-delay attacks by only accepting acknowledgement frames that are delayed up to t_a. Otherwise, if no timely authentic acknowledgement frame is received by a victim node, a pulse-delay attack actually degrades to a collision attack. As discussed above, the secure phase-lock optimization mitigates collision attacks greatly.

6.2.3 *Cost of Securing Acknowledgement Frames*

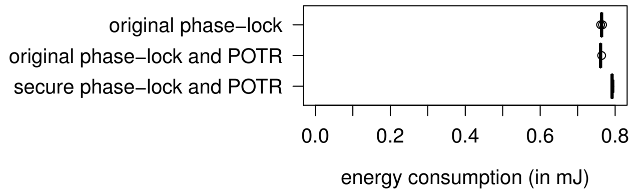
While the secure phase-lock optimization greatly mitigates collision attacks and even resists pulse-delay attacks, it comes at the cost of rendering the reception and transmission of unicast frames more energy consuming due to authenticating acknowledgement frames.

In order to get an overview of the increased energy consumption at the receiver side, the experiment in Section 6.1.1 was modified as follows. This time, A sent 100-byte unicast frames to B. In three subsequent runs, A and B were configured to use (i) the original phase-lock optimization, but not POTR, (ii) the original phase-lock optimization and POTR, and (iii) the secure phase-lock optimization and POTR. In each run, 100 traces of B's energy consumption during frame receptions were recorded. Throughout, 8-byte MICs and the dozing optimization were used.

Figure 16a shows the results. If POTR is disabled, our implementation sends 802.15.4-compliant acknowledgement frames in software. (Alternatively, one could leverage the built-in capability of the CC2538 to send 802.15.4-compliant acknowledgement frames. However, this feature does not seem to work when enabling the SHR search of the CC2538



(a)



(b)

Figure 16: Energy consumption per (a) reception and (b) transmission of a 100-byte unicast frame

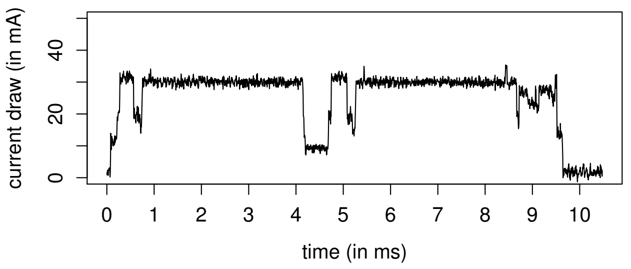


Figure 17: Current draw while the secure phase-lock optimization transmits a 100-byte unicast frame

without restarting the receive mode thereafter, which we do in order to save time.) If POTR is enabled, the energy consumption per reception decreases slightly because POTR uses shorter acknowledgement frames. Finally, if the secure phase-lock optimization is enabled, the energy consumption per reception increases only marginally since we accelerate the transmission of authenticated acknowledgement frames using the four measures that were listed in Section 4.1.1.

To also get an overview of the increased energy consumption at the sender side, the above experiment was repeated with the difference being that the energy consumption of A, rather than B, was measured during 100 transmissions per run. For triggering measurements, A set its `AD3/DIO3` pin high 1ms before strobing. Furthermore, the guard time t_g of the original phase-lock optimization was cut down so that both the secure and the original phase-lock optimization only needed to strobe each unicast frame at most twice.

Figure 16b gives the results. Again, when switching the secure phase-lock optimization on, the overall increase in energy consumption is low. In this regard, we note that our implementation applies a tweak to the original operation of unicast transmissions. In between the first two unicast frames of a strobe, our implementation disables the receive mode like when strobing broadcast frames, as exemplified in Figure 17. This works because the first frame of a strobe only serves to wake up the receiver(s).

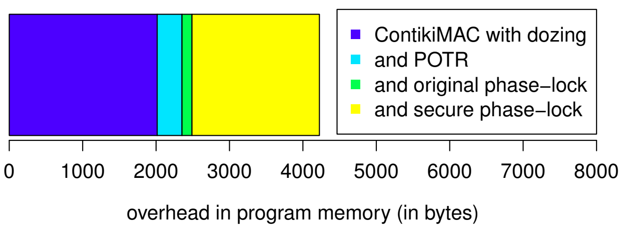


Figure 18: Overhead in program memory of the `secrdc_driver` compared to using the `nullrdc_driver`

6.2.4 *Overhead in Program Memory*

For measuring the overhead in program memory of the secure phase-lock optimization, the tool `arm-none-eabi-size` was used. In addition, to put this value into perspective, the overhead in program memory of the `secrdc_driver` compared to using Contiki's `nullrdc_driver`, which just always leaves the transceiver in receive mode, was measured. Specifically, the overhead in program memory of the `secrdc_driver` was measured in four different configurations, namely when (i) POTR, as well as any phase-lock optimization are off, (ii) POTR is on and any phase-lock optimization is off, (iii) both POTR and the original phase-lock optimization are on, and (iv) both POTR and the secure phase-lock optimization are on. Throughout, the dozing, as well as POTR's last bits optimization were enabled.

Figure 18 dissects the overhead in program memory in all four configurations. For example, enabling the secure phase-lock optimization consumes 1736 bytes of program memory. All configurations fit comfortably onto OpenMotes as they have 512KB of program memory [29].

7 Conclusions and Future Work

ContikiMAC falls short of countering denial-of-sleep attacks. We have proposed two optimizations to ContikiMAC to counter ding-dong ditching, collision attacks, as well as pulse-delay attacks. Our optimizations come at a low overhead and also save energy during normal operation. Both optimizations do, however, not prevent denial-of-sleep attacks entirely. For example, even though the secure phase-lock optimization resists hidden wormholes, hidden wormholes can still cause an energy-consuming reorganization of the routing topology [17]. Future work should thus investigate further preventive, detective, and reactive defenses against denial-of-sleep attacks. Besides, future work should integrate other supplements to the original version of ContikiMAC into our implementation, such as burst forwarding [9, 5], opportunistic routing [8, 20], or channel hopping [2].

8 References

[1] IEEE Standard 802.15.4, 2011.

[2] B. Al Nahas, S. Duquennoy, V. Iyer, and T. Voigt. Low-power listening goes multi-channel. In *Proceedings of the 2014 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 2–9. IEEE, 2014.

[3] M. Brownfield, Y. Gupta, and N. Davis. Wireless sensor network denial of sleep attack. In *Proceedings of the Sixth Annual IEEE SMC Information Assurance Workshop (IAW '05)*, pages 356–364. IEEE, 2005.

[4] H. S. Chiu and K.-S. Lui. DelPHI: wormhole detection mechanism for ad hoc wireless networks. In *Proceedings of the 1st International Symposium on Wireless Pervasive Computing*, pages 6–11. IEEE, 2006.

[5] B. Djamaa and M. Richardson. Improved broadcast communication in radio duty-cycled networks. Technical report, 2015.

[6] A. Dunkels. The ContikiMAC radio duty cycling protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[7] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN 2004)*, pages 455–462. IEEE, 2004.

[8] S. Duquennoy, O. Landsiedel, and T. Voigt. Let the tree bloom: scalable opportunistic routing with ORPL. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*, pages 2:1–2:14. ACM, 2013.

[9] S. Duquennoy, F. Österlind, and A. Dunkels. Lossy links, low power, high throughput. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*, pages 12–25. ACM, 2011.

[10] A. El-Hoiydi and J.-D. Decotignie. WiseMAC: an ultra low power MAC protocol for multi-hop wireless sensor networks. In *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004)*, pages 18–31. Springer, 2004.

[11] S. Ganeriwal, C. Pöpper, S. Čapkun, and M. B. Srivastava. Secure time synchronization in sensor networks. *ACM Transactions on Information and System Security (TISSEC)*, 11(4):23:1–23:35, 2008.

[12] Z. He and T. Voigt. Droplet: a new denial-of-service attack on low power wireless sensor networks. In *Proceedings of the 2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2013)*, pages 542–550. IEEE, 2013.

[13] P. Huang, L. Xiao, S. Soltani, M. Mutka, and N. Xi. The evolution of MAC protocols in wireless sensor networks: a survey. *IEEE Communications Surveys Tutorials*, 15(1):101–120, 2013.

[14] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, 2011. Updates RFC 4944.

[15] K.-F. Krentz and Ch. Meinel. Handling reboots and mobility in 802.15.4 security. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC '15)*, pages 121–130. ACM, 2015.

[16] K.-F. Krentz, Ch. Meinel, and M. Schnjakin. POTR: practical on-the-fly rejection of injected and replayed 802.15.4 frames. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES 2016)*. IEEE, 2016.

[17] K.-F. Krentz and G. Wunder. 6LoWPAN security: avoiding hidden wormholes using channel reciprocity. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices (TrustED '14)*, pages 13–22. ACM, 2014.

[18] A. Mauro, X. Fafoutis, S. Mödersheim, and N. Dragoni. Detecting and preventing beacon replay attacks in receiver-initiated MAC protocols for energy efficient WSNs. In *Proceedings of the 18th Nordic Conference (NordSec 2013)*.

[19] M. Michel, T. Voigt, L. Mottola, N. Tsiftes, and B. Quoitin. Predictable MAC-level performance in low-power wireless under interference. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN '16)*, pages 13–22. Junction, 2016.

[20] G. Z. Papadopoulos, A. Gallais, T. Noel, V. Kotsiou, and P. Chatzimisios. Enhancing ContikiMAC for bursty traffic in mobile sensor networks. In *Proceedings of IEEE SENSORS 2014*, pages 257–260. IEEE, 2014.

[21] D. R. Raymond, R. Marchany, M. Brownfield, and S. Midkiff. Effects of denial-of-sleep attacks on wireless sensor network MAC protocols. *IEEE Transactions on Vehicular Technology*, 58(1):367–380, 2009.

[22] D. R. Raymond and S. Midkiff. Clustered adaptive rate limiting: defeating denial-of-sleep attacks in wireless sensor networks. In *Proceedings of the Military Communications Conference (MILCOM 2007)*, pages 1–7. IEEE, 2007.

[23] Q. Ren and Q. Liang. Secure media access control (MAC) in wireless sensor networks: intrusion detections and countermeasures. In *Proceedings of the 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2004)*, pages 3025–3029. IEEE, 2004.

[24] N. Sastry and D. Wagner. Security considerations for IEEE 802.15.4 networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security (WiSe '04)*, pages 32–42. ACM, 2004.

[25] M. Sha, G. Hackmann, and C. Lu. Energy-efficient low power listening for wireless sensor networks in noisy environments. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN '13)*, pages 277–288. ACM, 2013.

[26] H. Song, S. Zhu, and G. Cao. Attack-resilient time synchronization for wireless sensor networks. In *Proceedings of the 2005 IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2005)*. IEEE, 2005.

[27] Texas Instruments. *CC2538 SoC for 2.4-GHz IEEE 802.15.4 & ZigBee/ZigBee IP Applications User's Guide (Rev. C)*. `http://www.ti.com/lit/ug/swru319c/swru319c.pdf`.

[28] M. P. Uwase, M. Bezunartea, J. Tiberghien, J.-M. Dricot, and K. Steenhaut. Poster: ContikiMAC, some critical issues with the CC2420 radio. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN '16)*, pages 257–258. Junction, 2016.

[29] X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister. OpenMote: open-source prototyping platform for the industrial IoT. In *Ad Hoc Networks*, volume 155, pages 211–222. Springer, 2015.

[30] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *IEEE Computer*, 35(10):54–62, 2002.

[31] W. Yang, Q. Wang, Y. Qi, and S. Sun. Time synchronization attacks in IEEE802.15.4e networks. In *Proceedings of the International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI 2014)*, pages 166–169. IEEE, 2014.