# Denial-of-Sleep-Resilient Session Key Establishment for IEEE 802.15.4 Security: From Adaptive to Responsive

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

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

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

## Abstract

Battery-powered and energy-harvesting IEEE 802.15.4 nodes are subject to so-called denial-of-sleep attacks. Such attacks generally aim at draining the energy of a victim device. Especially, session key establishment schemes for IEEE 802.15.4 security are susceptible to denial-of-sleep attacks since injected requests for session key establishment typically trigger energy-consuming processing and communication. Nevertheless, Krentz et al.'s Adaptive Key Establishment Scheme (AKES) for IEEE 802.15.4 security is deemed to be resilient to denial-of-sleep attacks thanks to its energy-efficient design and special defenses. However, thus far, AKES' resilience to denial-of-sleep attacks was presumably never evaluated. In this paper, we make two contributions. First, we evaluate AKES' resilience to denial-of-sleep attacks both theoretically and empirically. We particularly consider two kinds of denial-of-sleep attacks, namely `HELLO` flood attacks, as well as what we introduce in this paper as "yo-yo attacks". Our key finding is that AKES' denial-of-sleep defenses require trade-offs between denial-of-sleep resilience and the speed at which AKES adapts to topology changes. Second, to alleviate these trade-offs, we devise and evaluate new denial-of-sleep defenses. Indeed, our newly-devised denial-of-sleep defenses turn out to significantly accelerate AKES' reaction to topology changes, without incurring much overhead nor sacrificing on security.

## 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*

Internet of things, link layer security, key management, denial-of-service, denial-of-sleep.

International Conference on Embedded Wireless Systems and Networks (EWSN) 2018
14–16 February, Madrid, Spain

ISBN: 978-0-9949886-2-1

## 1 Introduction

IEEE 802.15.4 well established as a radio standard for implementing Internet of things (IoT) applications [1]. Main features of IEEE 802.15.4 are reliable mesh topologies, cheap radio modules, and energy-efficient operation. Furthermore, complementary protocols from the Internet Engineering Task Force (IETF) enable the seamless integration of IEEE 802.15.4 networks with IPv6 networks [13, 33].

Part of the IEEE 802.15.4 radio standard is IEEE 802.15.4 security, which specifies mechanisms for detecting injected frames, encrypting the payload of frames, and detecting replayed frames. Internally, to detect injected frames, IEEE 802.15.4 security adds message integrity codes (MICs) to frames. These MICs are generated using a tweaked version of Counter with CBC-MAC (CCM) [31]. Further, as CCM requires nonces, IEEE 802.15.4 security adds an incrementing frame counter to each frame and derives a frame's CCM nonce therefrom. Also, IEEE 802.15.4 security uses CCM for encrypting the payloads of frames. To detect replayed frames, on the other hand, IEEE 802.15.4 security compares the frame counter of an incoming frame with the one of the last accepted frame from the sender. Yet, what is left unspecified by IEEE 802.15.4 security is session key establishment.

Establishing session keys, rather than using predistributed keys unchanged, is not strictly necessary. However, if predistributed keys were used unchanged, each IEEE 802.15.4 node would have to persist all its anti-replay data of any previous interaction across reboots so as to prevent replay attacks after reboots [16, 3]. This would be highly problematic since the only non-volatile memory on most IEEE 802.15.4 nodes is flash memory, which is energy consuming, slow, as well as prone to wear [29]. Moreover, after a reboot, a node's frame counter could not be reset as, otherwise, frames from a rebooted node would be considered as replayed and CCM nonces would reoccur [27, 16, 3]. Hence, Sastry et al. considered storing a node's frame counter in non-volatile memory, too [27]. Establishing session keys, by contrast, frees IEEE 802.15.4 nodes of storing anti-replay data and frame counters across reboots [16, 27].

But, establishing session keys for IEEE 802.15.4 security is not straightforward. In particular, one special requirement that arises in this context is to resist so-called denial-of-sleep attacks. Such attacks generally aim at expending the limited energy reserves of battery-powered and energy-harvesting devices [2]. Another special requirement that arises in this

context is to not just establish session keys with neighboring nodes once at start up, but also at runtime so as to support mobile nodes and changing surroundings.

Krentz et al.'s Adaptive Key Establishment Scheme (AKES) addresses these special requirements comparatively well [16]. In fact, unlike public-key cryptography (PKC)- and key distribution center (KDC)-based alternatives [25, 23, 24, 28, 22], AKES operates in a distributed manner and dispenses with PKC. This already makes AKES relatively resilient to denial-of-sleep attacks since requests for session key establishment, called HELLOs in AKES, do neither trigger energy-consuming processing nor communication. Additionally, AKES starts to shed HELLOs if they arrive in bursts. After all, AKES currently appears to be the only session key establishment scheme for IEEE 802.15.4 security that supports mobile nodes and changing surroundings. For this, AKES deletes uncommunicative neighbors on the one hand and discovers new neighbors by occasionally broadcasting HELLOs on the other hand. Notably, to save energy, AKES reduces the rate of HELLOs when the network topology is stable and increases the rate of HELLOs when the network topology is instable. However, an attacker may destabilize a network's topology and hence cause AKES to consume more energy. As we will detail in Section 2.2, such attacks manifest themselves in transient links, which is why we collectively refer to them as "yo-yo attacks". To some extent, AKES mitigates yo-yo attacks by not removing uncommunicative neighbors immediately, but only after a hysteresis.

In this paper, we make two contributions:

- First, we give the presumably first evaluation of AKES' resilience to HELLO flood and yo-yo attacks. Regarding HELLO flood attacks, we find AKES' defense effective, but only when compromising on the speed at which AKES establishes session keys. Similarly, as for yo-yo attacks, we find AKES' defense effective, but only when compromising on the speed at which AKES deletes uncommunicative neighbors. In sum, AKES' denial-of-sleep defenses require trade-offs between denial-of-sleep resilience and the speed at which AKES adapts to topology changes.
- Second, to alleviate these trade-offs, we devise and evaluate a new defense against HELLO flood attacks, as well as a new defense against yo-yo attacks. As a result, our defense against HELLO flood attacks significantly lowers delays to session key establishment, without incurring much overhead nor sacrificing on security. Likewise, our defense against yo-yo attacks significantly lowers delays to the deletion of uncommunicative neighbors, without incurring much overhead nor sacrificing on security.

The rest of this paper is organized as follows. Section 2 first provides background information on AKES. Section 3 then reviews AKES' original denial-of-sleep defenses and presents our newly-devised ones. Section 4 sketches our implementation, followed by a comparative evaluation between AKES' original denial-of-sleep defenses and ours in Section 5. Section 6 discusses related work and Section 7 concludes.

# 2 Background on the Adaptive Key Establishment Scheme (AKES)

This section (i) recaps the design of AKES, (ii) outlines the operation of HELLO flood and yo-yo attacks on AKES, and (iii) details AKES' original denial-of-sleep defenses[1].

## 2.1 Design

Rather than using KDCs or PKC, AKES derives session keys from predistributed symmetric keys. This is done in the course of a three-way handshake, whose details are shown in Figure 1. Such a three-way handshake begins if a node $A$ broadcasts a HELLO, which contains a cryptographic random number $R_A$. Any receiver $B$ that wishes to establish session keys with $A$ stores $A$ as a tentative neighbor and, after a random back off period $T_{\text{bac}} < M_{\text{bac}}$, replies with a HELLOACK. This HELLOACK contains another cryptographic random number $R_B$ and a MIC. The MIC is generated using a temporary pairwise key $K'_{A,B}$ that is derived from a predistributed symmetric key $K_{A,B}$ between $A$ and $B$, as well as the two cryptographic random numbers $R_A$ and $R_B$. Upon receipt of this HELLOACK, $A$ checks the contained MIC and, if successful, stores $B$ as a permanent neighbor, and, lastly, acknowledges with an ACK. This ACK also contains a MIC that is generated using the temporary pairwise key $K'_{A,B}$ between $A$ and $B$ once more. As $B$ receives an authentic ACK from a tentative neighbor $A$, $B$ turns $A$ into a permanent neighbor. After this three-way handshake, $A$ and $B$ store each other as permanent neighbors and can use $K'_{A,B}$ as their pairwise session key.

### *2.1.1 Adaptation to Threat Models*

Instead of, or in addition to, pairwise session keys, AKES also supports the use of group session keys. Taking this option is appropriate if keys can not leak to an attacker. Under this threat model, it also suffices to predistribute a network-wide symmetric key, rather than pairwise symmetric keys. If pairwise session keys are required on the other hand, but predistributing pairwise symmetric keys is too memory consuming, AKES can also be configured to use a more memory-efficient scheme for predistributing pairwise symmetric keys, such as Blom's scheme [4]. This adaptability allows users to select the most efficient key predistribution scheme for their use case and enables different key predistribution schemes to share a common code base [16].

### *2.1.2 Adaptation to Topology Changes*

For scheduling the broadcasting of HELLOs, AKES adopts the Trickle algorithm [20]. This algorithm takes three parameters:

$I_{\text{min}}$**:** the minimum interval duration

$I_{\text{max}}$**:** the maximum interval duration

$k$**:** the redundancy constant
and maintains three variables:

$c$**:** a counter

$I$**:** the current interval duration

$t$**:** an instant within the second half of the current interval

[1] Our description of AKES reflects the current open-source implementation of AKES, which slightly deviates from the corresponding publication [16]. We list the differences between the open-source implementation and the publication in Appendix A.

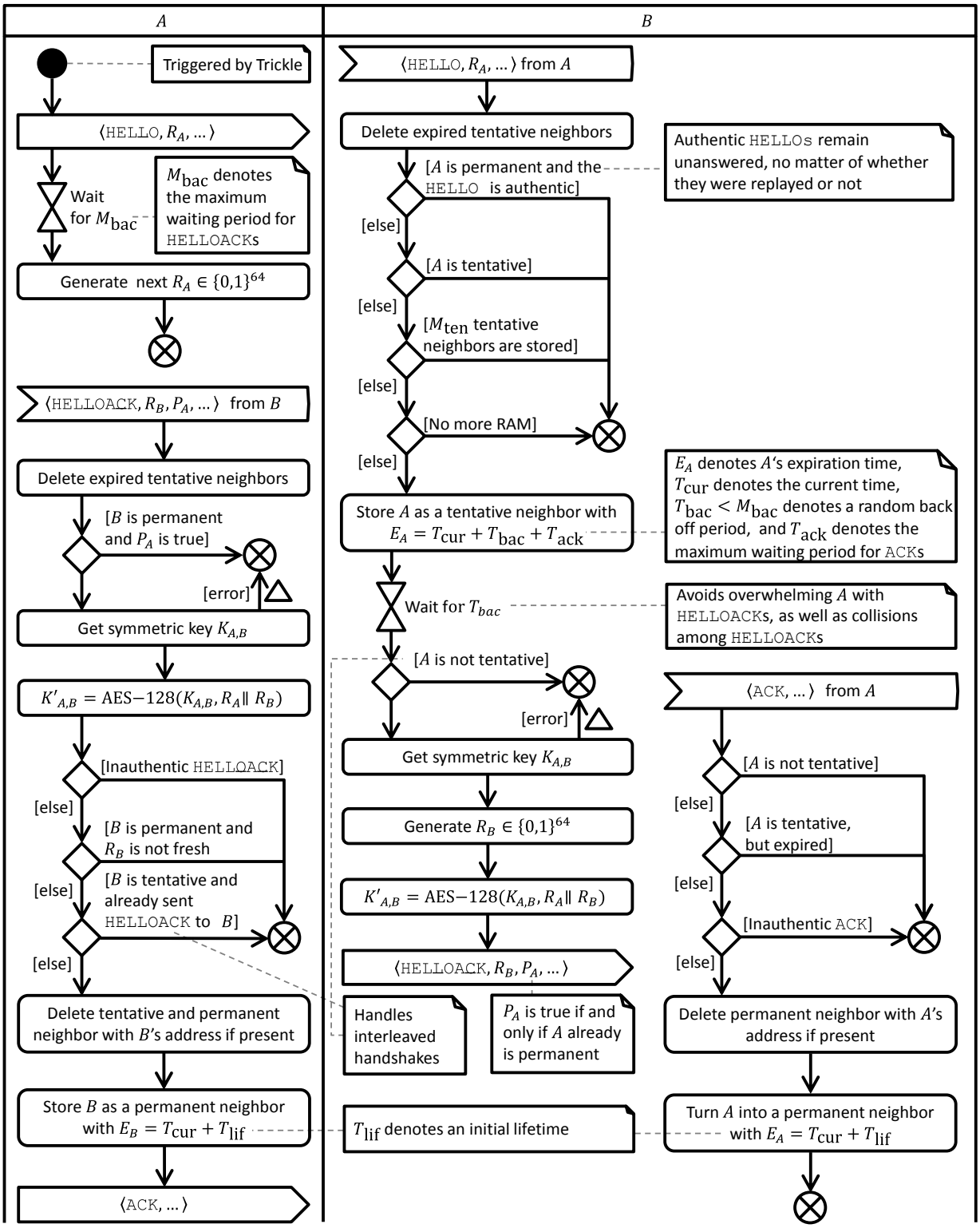


**Figure 1. Detailed instructions for processing HELLOs, HELLOACKs, and ACKs**

At start up, Trickle sets $c$ to 0, $I$ to a random duration within the range $[I_{\min}, I_{\max}]$, and $t$ to a random instant within the range $[\frac{I}{2}, I)$. Up until time $t$, Trickle increments $c$ upon receipt of a consistent broadcast, where the notion of consistent is left application specific. At time $t$, Trickle broadcasts if and only if $c < k$. What Trickle broadcasts is also left application specific. At the end of the current interval $I$, Trickle starts a new interval with $c = 0$, $I = \min\{I \times 2, I_{\max}\}$, and a random instant $t \in [\frac{I}{2}, I)$. A new interval also begins immediately if a reset is issued - unless $I$ already is at its minimum $I_{\min}$. In the event of such resets, Trickle starts over with $c = 0$, $I = I_{\min}$, and a random instant $t \in [\frac{I_{\min}}{2}, I_{\min})$. Altogether, Trickle reduces its broadcast rate exponentially while consistency is achieved and increases its broadcast rate when observing inconsistencies. Further energy is saved by suppressing broadcasts if $c < k$.

AKES tailors Trickle to broadcast HELLOs as follows. By default, AKES sets $I_{\min} = \max\{30\text{s}, 2 \times M_{\text{bac}} + 1\text{s}\}$, $I_{\max} = I_{\min} \times 2^8$, and $k = 2$. Setting $I_{\min}$ to $\max\{30\text{s}, 2 \times M_{\text{bac}} + 1\text{s}\}$ avoids broadcasting another HELLO while still waiting for HELLOACKs. As consistent broadcasts, AKES considers fresh authentic HELLOs, as long as they do not originate from a permanent neighbor that already sent a fresh authentic HELLO since the last time the receiver broadcasted a HELLO. Furthermore, AKES resets Trickle if $\max\{\lfloor \frac{n}{4} \rfloor, 1\}$ permanent neighbors were added during the current interval, where $n$ is the current number of permanent neighbors. Yet, when establishing a new session key with a permanent neighbor (which, e.g., happens if a permanent neighbor reboots), AKES does not count this permanent neighbor as new. To speed up joining a network, AKES broadcasts one HELLO at start up, as well as resets Trickle thereafter.

Besides, to detect whether a permanent neighbor got out of range, AKES sends an UPDATE frame to permanent neighbors that sent no fresh authentic frame for a critical period of time $T_{\text{lif}}$. If such an uncommunicative neighbor does not reply with an UPDATEACK after a couple of retransmissions, AKES deletes that neighbor.

## 2.2 Denial-of-Sleep Vulnerabilities

### *2.2.1 HELLO Flood Attacks*

In a HELLO flood attack, an attacker injects a HELLO [15], thereby causing each receiver to store a tentative neighbor with the address that is pretended to be the source address of the HELLO. More severely, receivers will also send a HELLOACK after a random back off period. This consumes a good amount of energy, especially if the attacker does not acknowledge the receipt of HELLOACKs because victim nodes usually retransmit HELLOACKs a few times [17].

### *2.2.2 Yo-Yo Attacks*

Apart from launching HELLO flood attacks, an attacker can also make victim nodes consume more energy by carrying out what we refer to as yo-yo attacks. Yo-yo attacks share the idea of destabilizing the topology of the victim network so as to provoke additional HELLO, HELLOACK, and ACK transmissions and receptions. Let us, e.g., consider an external attacker, i.e., an attacker who possesses no key of the victim network [10]. There appear to be exactly two methods for him to launch a yo-yo attack. First, in a collision attack, an external attacker jams certain frames so as to cause bit errors and hence prevent their successful reception [35]. For example, if an attacker jams all frames except HELLOs, HELLOACKs, and ACKs, AKES will delete permanent neighbors and later reestablish session keys. As a result, victim nodes send and receive additional HELLOACKs and ACKs. Moreover, victim nodes potentially reset Trickle due to adding permanent neighbors. Thus, this example yo-yo attack may ensue more HELLO transmissions and receptions, too. Also, collision attacks open up plenty of other strategies for launching yo-yo attacks, as we will detail in Section 5.2. Second, in a hidden wormhole attack [5], an external attacker tunnels frames between distant parts of the victim network verbatim. In effect, this tricks distant nodes to believe they were neighbors, thereby causing AKES to establish session keys between them and possibly reset Trickle, too. Moreover, if we consider an internal attacker, i.e. an attacker who possesses one or more keys of the victim network [10], he may, in addition, be able to make AKES add permanent neighbors by injecting HELLOs and authentic ACKs, or by replying to HELLOs with authentic HELLOACKs. This provokes additional HELLOACK and ACK transmissions and receptions, and potentially Trickle resets, as well.

## 2.3 Denial-of-Sleep Defenses

### *2.3.1 Defense against HELLO Flood Attacks*

AKES' defense against HELLO flood attacks is twofold. On the one hand, AKES dispenses with both a KDC and PKC. Thus, unlike when using a KDC, requests for session key establishment are not routed to the KDC, which would aggravate HELLO flood attacks [11]. Likewise, using PKC would aggravate HELLO flood attacks since PKC involves more computation than symmetric-key cryptography [11]. On the other hand, as shown in Figure 1, AKES sheds HELLOs in three occasions, namely (i) if the current number of tentative neighbors reaches a limit $M_{\text{ten}}$, (ii) if the sender of a HELLO is already stored as a tentative neighbor, and (iii) if there is no more random-access memory (RAM) for storing additional permanent neighbors available. In a follow-up effort, which we will touch on in Section 6, Krentz et al. further reduced AKES' energy consumption under HELLO flood attacks by shedding HELLOs already during reception [18].

### *2.3.2 Defense against Yo-Yo Attacks*

Furthermore, AKES' defense against yo-yo attacks is to not delete uncommunicative permanent neighbors immediately, but only after a hysteresis $T_{\text{lif}}$. This limits the rate at which AKES readds the same permanent neighbor and hence rate-limits repeated yo-yo attacks.

# 3 Review and Revision of AKES' Denial-of-Sleep Defenses

In this section, we first argue that AKES requires trade-offs between denial-of-sleep resilience and the speed at which AKES adapts to topology changes. Subsequently, we explain our newly-devised denial-of-sleep defenses, which turn out to alleviate these trade-offs.

## 3.1 Review

### *3.1.1 Defense against HELLO Flood Attacks*

Recall the following parameters of AKES:

**$M_{\text{ten}}$:** the maximum number of tentative neighbors

$M_{\text{bac}}$: the maximum back off period of HELLOACKs

$T_{\text{ack}}$: the maximum waiting period for ACKs

Hence, by injecting HELLOs with random source addresses, external attackers can cause AKES to send HELLOACKs at a mean rate of

$$\frac{M_{\text{ten}}}{\frac{1}{2}M_{\text{bac}} + T_{\text{ack}}} \tag{1}$$

, not counting retransmissions separately. Unfortunately, this rate can not be tuned without affecting the speed at which AKES reacts to topology changes at the same time. Specifically, lowering $M_{\text{ten}}$ reduces the number of neighbors that AKES can add in parallel. Increasing $M_{\text{bac}}$, on the other hand, delays session key establishment. Lastly, increasing $T_{\text{ack}}$ entails the following issue. Consider that a node $A$ broadcasted a HELLO and that session key establishment with a neighbor $B$ did not complete due to a missed HELLOACK or ACK. Now, if $A$ sends another HELLO before $B$ may delete $A$ from its list of tentative neighbors, $B$ will ignore $A$'s HELLO.

Moreover, internal attackers may be able to cause a victim node to send HELLOACKs at a higher rate than in Equation (1). This is because, once an internal attacker obtained the predistributed symmetric key between a victim node and another node, he can (i) establish session keys with the victim node and (ii) reestablish session keys with the victim node over and over, with the only delay being $T_{\text{bac}} < M_{\text{bac}}$. Concretely, if an internal attacker controls $k$ of a victim node's $n \geq k$ permanent neighbors, the internal attacker can force the victim node to send HELLOACKs at a mean rate of

$$\frac{k}{\frac{1}{2}M_{\text{bac}}} \tag{2}$$

, not counting retransmissions separately. Thus, to also withstand internal attackers, $M_{\text{bac}}$ has to be chosen long.

#### 3.1.2 Defense against Yo-Yo Attacks

As for AKES' defense against yo-yo attacks, a similar conflict arises. While extending $T_{\text{lif}}$ increases AKES' resilience to yo-yo attacks, it also defers the deletion of uncommunicative permanent neighbors and hence the freeing of allocated RAM. Moreover, if much RAM is allocated for storing uncommunicative permanent neighbors, this can deprive AKES of establishing session keys with actual neighbors.

Nevertheless, $T_{\text{lif}}$ must be chosen long. This is because an external attacker may set up multiple hidden wormholes or launch collision attacks on several links. In effect, victim nodes will not always readd the same permanent neighbor during a yo-yo attack, but different permanent neighbors over and over. In such occasions, AKES will still issue many Trickle resets, unless $T_{\text{lif}}$ is chosen very long so that AKES either abstains from readding any of a whole set of permanent neighbors for long or quickly abstains from adding further permanent neighbors because of running out of RAM.

### 3.2 Revision

Both of our newly-devised denial-of-sleep defenses are based on leaky bucket counters (LBCs) [12]. The intuition behind an LBC is a bucket with a hole in it, as shown in Figure 2. Events drop into the bucket and increase its filling level. It is possible to associate different events with different drop sizes. As the bucket has a hole, its filling level decreases as long as there is water in it. Pressure is neglected so that the filling level of the bucket decreases with a constant rate ρ. In our use case, we want to avoid that the bucket overflows by taking appropriate actions beforehand, i.e., before the filling level exceeds the bucket's capacity β.

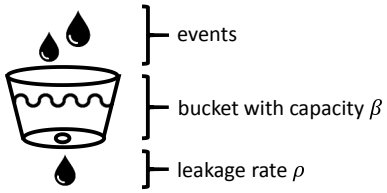


**Figure 2. Intuition behind leaky bucket counters**

#### 3.2.1 Defense against HELLO Flood Attacks

Specifically, to defend against HELLO flood attacks, we suggest that each node maintains an LBC $\text{LBC}_{\text{HELLOACK}}$ that is defined as follows:

**Capacity:** Its capacity $\beta_{\text{HELLOACK}}$ corresponds to the maximum number of HELLOACKs that AKES may send in the short run, not counting retransmissions separately.

**Leakage rate:** Its leakage rate $\rho_{\text{HELLOACK}}$ corresponds to the maximum rate at which AKES may send HELLOACKs in the long run, not counting retransmissions separately.

**Drop sizes:** In the event that a HELLOACK is scheduled to be sent, $\text{LBC}_{\text{HELLOACK}}$ is incremented by one. When retransmitting a HELLOACK, however, $\text{LBC}_{\text{HELLOACK}}$ is not incremented.

**Overflow prevention:** AKES shall shed an incoming HELLO if $\text{LBC}_{\text{HELLOACK}}$ may overflow otherwise.

An immediate benefit of our LBC-based HELLO flood defense is that its parameters are independent from other parameters of AKES. In fact, $M_{\text{bac}}$ can now be shortened - but should not be zeroed to avoid overwhelming senders of HELLOs with HELLOACKs, as well as collisions among HELLOACKs [37]. Likewise, $T_{\text{ack}}$ can now be minimized according to what is the maximum waiting period between the transmission of a HELLOACK and the reception of the corresponding ACK. Finally, $M_{\text{ten}}$ can now be configured independently from the aimed maximum rate of outgoing HELLOACKs.

For example, consider we aim for a mean rate of $\frac{1}{150}$Hz of outgoing HELLOACKs under continuous HELLO flood attacks by external attackers, not counting retransmissions separately. According to Equation (1), an appropriate configuration of AKES' original defense against HELLO flood attacks is $M_{\text{ten}} = 5$, $M_{\text{bac}} = 5$s, and $T_{\text{ack}} = 747.5$s. However, in order to also withstand HELLO flood attacks by, at least, one attacker-controlled permanent neighbor, choosing $M_{\text{ten}} = 5$, $M_{\text{bac}} = 300$s, and $T_{\text{ack}} = 600$s is necessary according to Equation (2). In either case, $T_{\text{ack}}$ is quite long, which can lead to delays to session key establishment if frame loss occurs, as described in Section 3.1. Moreover, raising $M_{\text{bac}}$ delays HELLOACKs and therefore session key establishment. By contrast, when using our defense against

HELLO flood attacks, a parameter set that provides an equal level of security is $M_{\text{bac}} = 5$s, $T_{\text{ack}} = 5$s, $\beta_{\text{HELLOACK}} = 20$, and $\rho_{\text{HELLOACK}} = \frac{1}{150}$Hz. Apparently, AKES reacts much faster to topology changes with these parameters. Beyond that, our defense against HELLO flood attacks protects against any number of attacker-controlled permanent neighbors. Table 1 lists these parameter sets with IDs for future reference.

#### 3.2.2 *Defense against Yo-Yo Attacks*

Similarly, to defend against yo-yo attacks, we suggest that each node maintains two additional LBCs $\text{LBC}_{\text{HELLO}}$ and $\text{LBC}_{\text{ACK}}$ that are defined as follows:

**Capacity:** The capacity $\beta_{\text{HELLO}}$ of $\text{LBC}_{\text{HELLO}}$ corresponds to the maximum number of HELLOs that AKES may broadcast in the short run.

**Leakage rate:** The leakage rate $\rho_{\text{HELLO}}$ of $\text{LBC}_{\text{HELLO}}$ corresponds to the maximum rate at which AKES may broadcast HELLOs in the long run.

**Drop sizes:** In the event that AKES broadcasts a HELLO, $\text{LBC}_{\text{HELLO}}$ is incremented by one. When retransmitting a HELLO, however, $\text{LBC}_{\text{HELLO}}$ is not incremented.

**Overflow prevention:** AKES suppresses a HELLO if $\text{LBC}_{\text{HELLO}}$ will overflow otherwise.

**Capacity:** The capacity $\beta_{\text{ACK}}$ of $\text{LBC}_{\text{ACK}}$ corresponds to the maximum number of ACKs that AKES may send in the short run, not counting retransmissions separately.

**Leakage rate:** The leakage rate $\rho_{\text{ACK}}$ of $\text{LBC}_{\text{ACK}}$ corresponds to the maximum rate at which AKES may send ACKs in the long run, not counting retransmissions separately.

**Drop sizes:** In the event that AKES sends an ACK, $\text{LBC}_{\text{ACK}}$ is incremented by one. When retransmitting an ACK, however, $\text{LBC}_{\text{ACK}}$ is not incremented.

**Overflow prevention:** AKES shall shed an incoming HELLOACK if $\text{LBC}_{\text{ACK}}$ may overflow otherwise.

Analogously, an immediate benefit of our LBC-based defense against yo-yo attacks is that $T_{\text{lif}}$ may now be minimized. Conversely, when using AKES' original defense against yo-yo attacks, $T_{\text{lif}}$ has to be chosen long so as to mitigate yo-yo attacks, as discussed in Section 3.1.2, and as is also empirically confirmed in Section 5.2.

## 4 Lightweight Implementation

We integrated our newly-devised denial-of-sleep defenses into the open-source implementation of AKES for the Contiki operating system in two steps [8]. First, we added an abstract data type for creating and managing LBCs to Contiki. For implementing the leaking of LBCs, we decided to lazily update the filling levels of LBCs upon enquiries. This not only reduces the processing overhead, but also avoids waking up a node from sleep mode for updating filling levels. Consequently, our implementation of LBCs is lightweight in terms of energy consumption. Also, our implementation of LBCs is lightweight in terms of RAM consumption. In fact, the RAM consumption per LBC is merely 12 bytes when using OpenMotes as target platform [30]. Second, we added the pouring of drops and the prevention of overflows

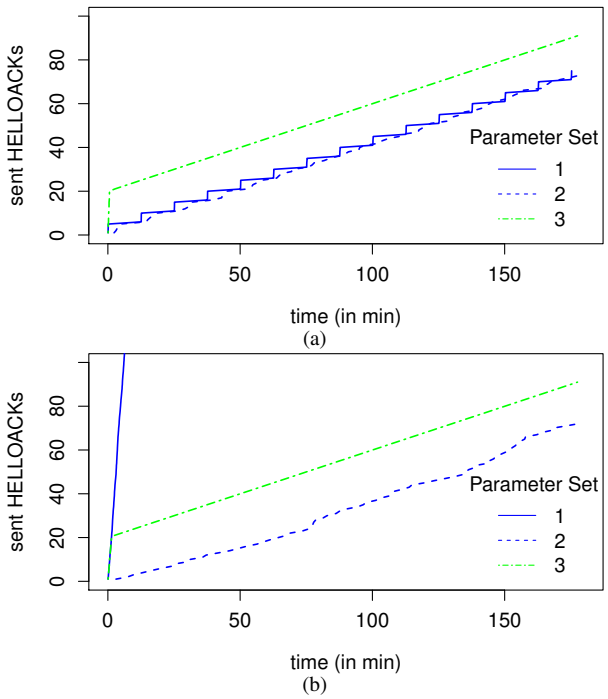


**Figure 3. Sent HELLOACKs under continuous HELLO flood attacks by an (a) external attacker and (b) internal attacker**

to AKES. Altogether, the addition of our denial-of-sleep defenses causes an overhead in program memory of just 248 bytes when using OpenMotes as target platform.

## 5 Comparative Evaluation

Using our implementation, we empirically compared AKES' original denial-of-sleep defenses and ours concerning effectiveness and responsiveness. In the following, we report on these experiments.

### 5.1 Defenses against HELLO Flood Attacks

#### 5.1.1 *External Attackers*

To compare the resilience to external attackers of AKES' original defense against HELLO flood attacks and ours, the following experiment was conducted. A Cooja simulation with two nodes was run for three virtual hours. The first node acted as an external attacker who continuously injected HELLOs with random source addresses at the rate of 1Hz. The second node acted as a victim and initially ran the original version of AKES with Parameter Set 1 shown in Table 1. This simulation was then rerun using (i) AKES' original defense against HELLO flood attacks with Parameter Set 2 and (ii) our defense against HELLO flood attacks with Parameter Set 3. In each run, the victim node logged its number of sent HELLOACKs. Throughout, the frame loss was 0%.

Figure 3a shows the results. At the beginning, the victim node answers five HELLOs in a row when using AKES' original defense against HELLO flood attacks. This is because AKES' original defense only comes into play as the number of tentative reaches the threshold $M_{\text{ten}} = 5$. Similarly, when using our defense, the bucket initially is empty, thus causing the victim node to answer the first $\beta_{\text{HELLOACK}} = 20$ HELLOs.

**Table 1. Parameter sets**

| ID | $I_{\min}$ | $I_{\max}$ | with LBCs | $M_{\text{ten}}$ | $M_{\text{bac}}$ | $T_{\text{ack}}$ | $T_{\text{lif}}$ | $\beta_{\text{HELLO}}$ | $\rho_{\text{HELLO}}$ | $\beta_{\text{HELLOACK}}$ $(=\beta_{\text{ACK}})$ | $\rho_{\text{HELLOACK}}$ $(=\rho_{\text{ACK}})$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 30s | 128min | × | 5 | 5s | 747.5s | ∞ | n/a | n/a | n/a | n/a |
| 2 | 601s | 160min16s | × | 5 | 300s | 600s | ∞ | n/a | n/a | n/a | n/a |
| 3 | 30s | 128min | ✓ | 5 | 5s | 5s | ∞ | 10 | $\frac{1}{300}$Hz | 20 | $\frac{1}{150}$Hz |
| 4 | 30s | 128min | × | 5 | 5s | 747.5s | 5min | n/a | n/a | n/a | n/a |
| 5 | 30s | 128min | × | 5 | 5s | 747.5s | 30min | n/a | n/a | n/a | n/a |
| 6 | 30s | 128min | ✓ | 5 | 5s | 5s | 5min | 10 | $\frac{1}{300}$Hz | 20 | $\frac{1}{150}$Hz |

Then, our defense gradually answers further HELLOs since the bucket leaks at the rate of $\rho_{\text{HELLOACK}} = \frac{1}{150}$Hz. AKES' original defense answers further HELLOs in bursts since all initially stored tentative neighbors expire closely after one another. As conjectured in Section 3, both defenses restrict the rate of outgoing HELLOACKs to $\frac{1}{150}$Hz in the long run.

### 5.1.2 Internal Attackers

Next, to compare the resilience to one attacker-controlled permanent neighbor of AKES' original defense against HELLO flood attacks and ours, the following experiment was conducted. Again, a Cooja simulation was run for three virtual hours in which one node acted as an attacker-controlled node and another node acted as a victim. In the first run, the victim node used AKES' original defense against HELLO flood attacks with Parameter Set 1. In the second run, the victim node used AKES' original defense against HELLO flood attacks with Parameter Set 2. In the third run, the victim node used our defense against HELLO flood attacks with Parameter Set 3. During all runs, the victim node logged its number of sent HELLOACKs, the frame loss was 0%, and the attacker-controlled node operated as follows. At first, the attacker-controlled node established session keys with the victim node. Then, the attacker-controlled node broadcasted inauthentic HELLOs at the rate of 1Hz. In the case that the victim node replied with a HELLOACK, the attacker-controlled node completed the three-way handshake by sending an authentic ACK in response.

As shown in Figure 3b, the results differ from the previous experiment as far as AKES' original defense against HELLO flood attacks is concerned. Particularly, when using Parameter Set 1, the victim node ends up with 2993 sent HELLOACKs after three virtual hours. This disastrous result arises because the attacker-controlled node reestablishes session keys with the victim node again and again with a short waiting period in between. This waiting period takes $\frac{1}{2}M_{\text{bac}} = 2.5$s on average. Only when tuning $M_{\text{bac}}$ appropriately, AKES' original defense against HELLO flood attacks approaches the aimed maximum rate of one outgoing HELLOACK per 150s. However, if more than one attacker-controlled permanent neighbors shows up, $M_{\text{bac}}$ has to be extended further. On the other hand, our defense against HELLO flood attacks protects against any number of attacker-controlled permanent neighbors, even without configuration changes.

### 5.1.3 Responsiveness

To compare AKES' speed of establishing session keys when using either AKES' original defense against HELLO flood attacks or ours, another set of Cooja simulations was

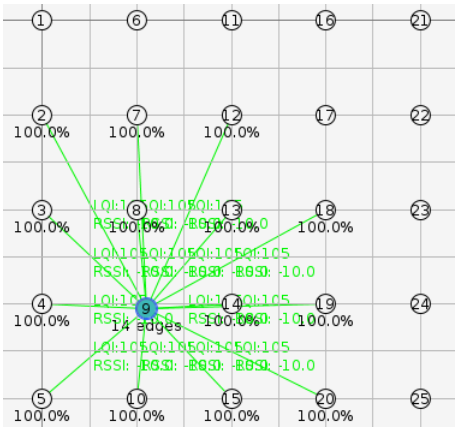


**Figure 4. Network topology of most of our Cooja simulations**

run. Throughout, the topology shown in Figure 4 was used, where every of the 25 nodes logged its number of permanent neighbors and booted at a pseudo-random point in time during the first 30 virtual minutes. In three successive runs over one virtual hour, all nodes were configured to use (i) AKES' original defense against HELLO flood attacks with Parameter Set 1, (ii) AKES' original defense against HELLO flood attacks with Parameter Set 2, and (iii) our defense against HELLO flood attacks with Parameter Set 3. These three runs were also repeated with (i) a frame loss of 10% (instead of 0%) without enabling retransmissions and (ii) a frame loss of 10% with three retransmissions at most.

Figure 5a shows the results with frame loss disabled. Overall, the speed of adding permanent neighbors does not differ greatly among the different HELLO flood defenses. Yet, Parameter Set 2 lags behind since it delays HELLOACKs by $\frac{1}{2}M_{\text{bac}} = 150$s on average. In the case of a frame loss of 10%, AKES' original defense against HELLO flood attacks becomes noticeably slower, as shown in Figure 5b. This is because of an issue we mentioned in Section 3.1 - if an ACK or HELLOACK is missed, tentative neighbors are kept for long if $T_{\text{ack}}$ is long, thus causing incoming HELLOs from tentative neighbors to be ignored for a long period of time. A remedy to this issue is to retransmit frames, as shown in Figure 5c. Despite this, Parameter Set 2 remains slower at adding permanent neighbors because it delays HELLOACKs.

### 5.1.4 Discussion

AKES' resilience to HELLO flood attacks comes at the cost of extending both $T_{\text{ack}}$ and $M_{\text{bac}}$. Extending $T_{\text{ack}}$, on the one hand, may be acceptable if retransmissions are enabled, as

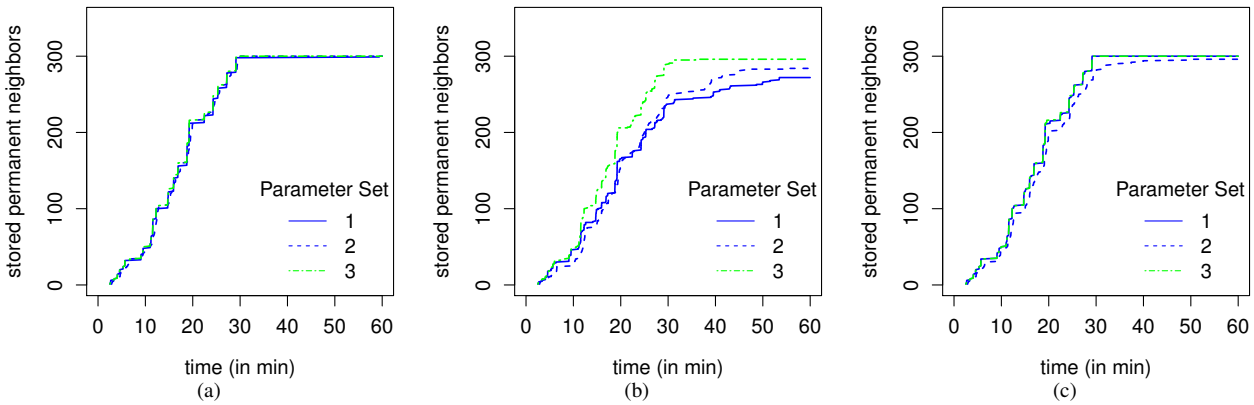


**Figure 5. Speed of adding permanent neighbors with (a) 0% frame loss, (b) 10% frame loss and without retransmissions, and (c) 10% frame loss and with three retransmissions at most**

shown in Figure 5c. Extending $M_{\text{bac}}$, on the other hand, severely deteriorates the user experience because session key establishment becomes very slow as a result. For comparison, our defense against HELLO flood attacks also protects against HELLO flood attacks with short durations for $T_{\text{ack}}$ and $M_{\text{bac}}$. Beyond that, our defense protects against any number of attacker-controlled nodes and is easier to configure.

## 5.2 Defenses against Yo-Yo Attacks

### *5.2.1 Collision Attacks*

To compare the effectiveness of AKES' original defense against yo-yo attacks and ours under collision attacks, a Cooja simulation was set up. In each run, the network shown in Figure 4 was simulated for 12 virtual hours, where all 25 nodes booted at a random point in time during the first 30 virtual minutes, and the frame loss was 0%. During each run, every node logged its number of sent HELLOs, HELLOACKs, and ACKs. In the first run, no attack was launched as a baseline for comparison. In the second run, the nodes $\{1,2,3,6,7,8,11,12,13\}$ solely received HELLOs, HELLOACKs, and ACKs, thus simulating collision attacks on any other kinds of frames. In the third run, the same set of nodes did, in addition, not receive HELLOs from their permanent neighbors. This collision attack effectively disables AKES' suppression of redundant HELLOs. In the forth run, if any of the nodes $\{1,2,3,6,7,8,11,12,13\}$ had just reset Trickle, i.e. $I = I_{\min}$, it did not receive HELLOs at all. The idea behind this collision attack is to delay session key establishment in order to cause multiple Trickle resets instead of just one. Initially, all nodes used Parameter Set 4 and then all four runs were repeated with Parameter Set 5 and 6[2].

Figure 6a through 6d shows the results. Let us first look at the number of sent HELLOs. If no collision attack is launched, relatively few HELLOs are sent, regardless of the employed parameter set. This is because Trickle suspects that the network topology is stable and therefore reduces the rate of HELLOs. A lot more HELLOs are sent when jamming all frames except HELLOs, HELLOACKs, and ACKs as this causes AKES to delete permanent neighbors, reestablish session keys, and potentially reset Trickle. Expectably, AKES' original defense against yo-yo attacks greatly mitigates such collision attacks if $T_{\text{lif}} = 30\text{min}$. This is due to the fact that AKES then deletes uncommunicative permanent neighbors only after $T_{\text{lif}} = 30\text{min}$. Conversely, when configuring AKES more responsively by setting $T_{\text{lif}} = 5\text{min}$, AKES' original defense against yo-yo attacks protects much worse. For comparison, although our defense against yo-yo attacks also uses $T_{\text{lif}} = 5\text{min}$, it limits the rate of sent HELLOs to $\rho_{\text{HELLO}} = \frac{1}{300}\text{Hz}$. Another attack strategy is to jam HELLOs to permanent neighbors so as to abstain victim nodes from suppressing HELLOs. Indeed, the number of sent HELLOs increases under such collision attacks, at least when using AKES' original defense, as shown in Figure 6c. By contrast, our defense successfully limits the rate of sent HELLOs to $\frac{1}{300}\text{Hz}$. Another level of aggravation is to jam HELLOs to nodes that had just reset Trickle so as to cause multiple Trickle resets instead of just one. Again, our defense limits the rate of sent HELLOs under such kind of collision attacks to $\frac{1}{300}\text{Hz}$, as shown in Figure 6d. A secondary repercussion of yo-yo attacks is additional HELLOACK and ACK transmissions and receptions. In this regard, our LBC-based restriction on the rate of HELLOACKs also takes effect in this context. AKES' original defense against HELLO flood attacks, by contrast, fails to limit the rate of HELLOACKs to $\frac{1}{150}\text{Hz}$. For example, in Figure 6b, the rate of answered HELLOs of node 7 in between hour 1 and 12 is $\frac{1}{70.09}\text{Hz}$ for Parameter Set 4, whereas it is $\frac{1}{150.00}\text{Hz}$ for Parameter Set 6. In sum, AKES' original defense against yo-yo attacks only prevents collision attacks when extending $T_{\text{lif}}$ to undesirable durations.

### *5.2.2 Hidden Wormhole Attacks*

Next, to compare AKES' original defense against yo-yo attacks and ours under hidden wormhole attacks, the above experiment was repeated with a modified topology, which is shown in Figure 7. Accordingly, the nodes $\{3,4,5,9,10,15\}$ and $\{11,16,17,21,22,23\}$ could communicate with each other. However, the hidden wormhole exclusively tunneled

[2]Though it seems difficult for an attacker to prevent some nodes from receiving a HELLO while letting it pass to others, this is possible, e.g., if the victim network uses ContikiMAC. This is because ContikiMAC transmits broadcasts as a strobe of frames, selected ones of which may be jammed exactly when a certain node wakes up. Alternatively, an attacker may install multiple jammers and locally jam HELLOs. In Section 6, we discuss the feasibility of collision and hidden wormhole attacks in more detail.

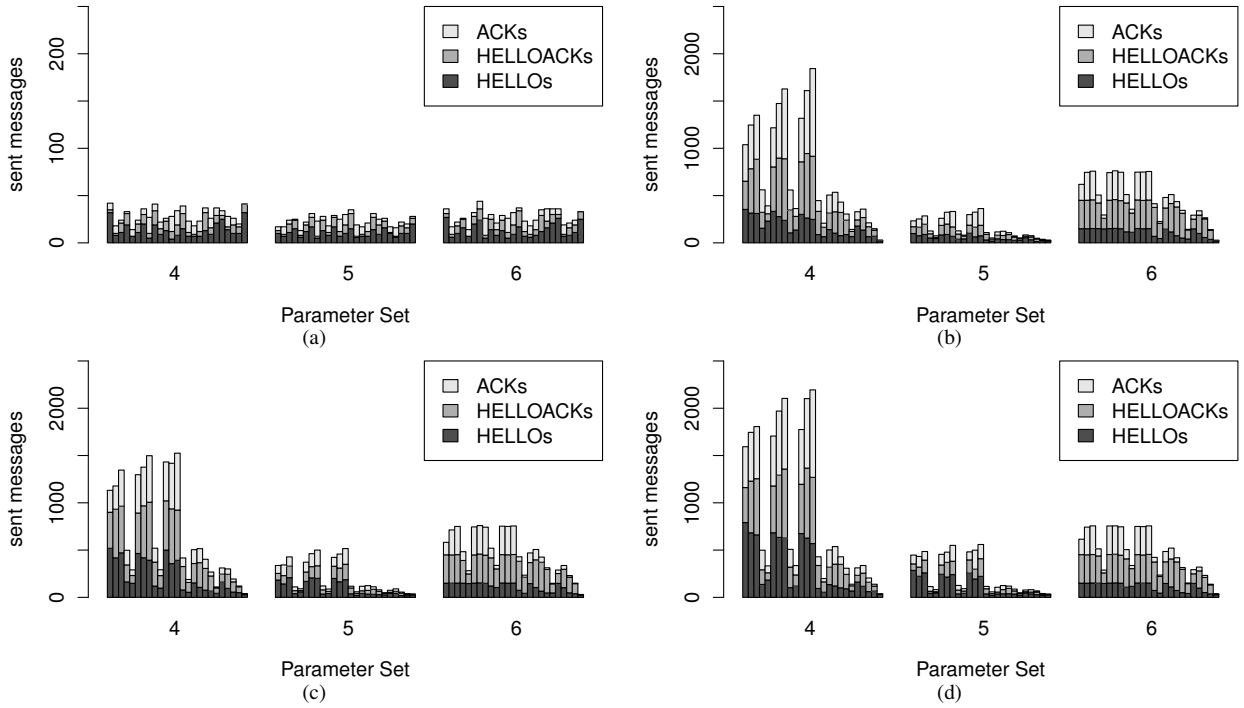


**Figure 6. Sent HELLOs, HELLOACKs, and ACKs per node after 12 virtual hours (a) without attacks, (b) when jamming all frames except HELLOs, HELLOACKs, and ACKs, (c) when jamming HELLOs to permanent neighbors in addition, and (d) when also jamming HELLOs to nodes that had just reset Trickle**

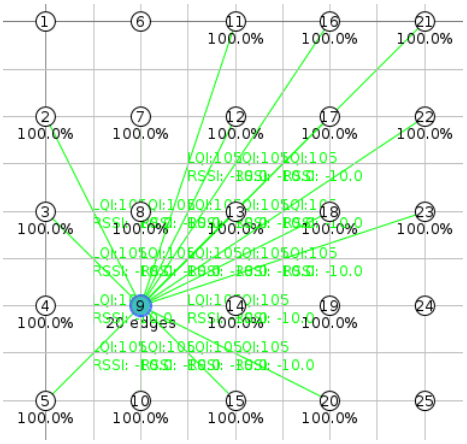


**Figure 7. Network topology with a hidden wormhole between $\{3,4,5,9,10,15\}$ and $\{11,16,17,21,22,23\}$**

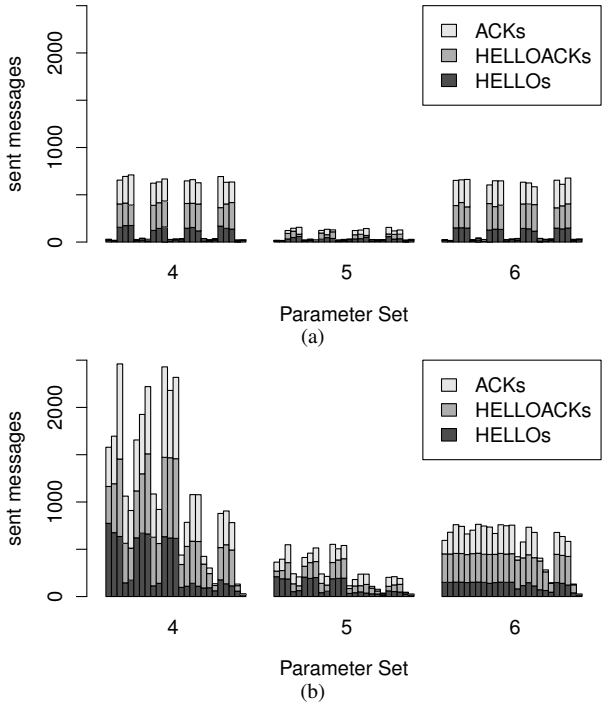


**Figure 8. Sent HELLOs, HELLOACKs, and ACKs per node after 12 virtual hours (a) in the presence of a hidden wormhole and (b) if launching collision attacks in parallel**

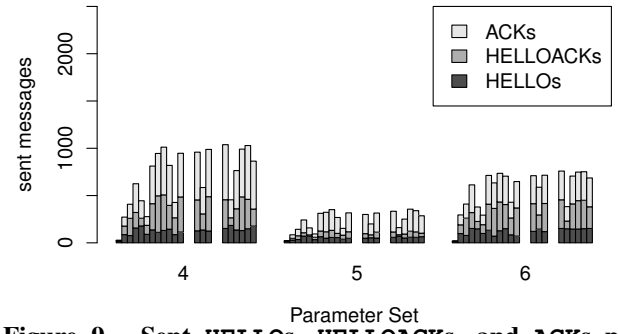


**Figure 9. Sent HELLOs, HELLOACKs, and ACKs per node after 12 virtual hours in the face of four attacker-controlled nodes**

HELLOs, HELLOACKs, and ACKs and not other kinds of frames. In the first run, no attacks other than the hidden wormhole attack was launched. In the second run, this attack was exacerbated by combining it with collision attacks on the nodes $\{1,2,3,6,7,8,11,12,13\}$. Specifically, like in the previous experiment, each of these nodes only received (i) ACKs, (ii) HELLOACKs, and (iii) HELLOs from non-permanent neighbors if the node had not just reset Trickle.

Figure 8a and 8b show the results. Interestingly, the hidden wormhole attack turned out to be relatively benign. Furthermore, the results suggest that such an attack only causes the nodes that are connected via the hidden wormhole to send more HELLOs, HELLOACKs, and ACKs, whereas collision attacks also cause other nodes of the victim network to send more HELLOs, HELLOACKs, and ACKs, as shown in Figure 6b through 6d. That said, Figure 8b demonstrates that combining hidden wormhole attacks with collision attacks exacerbates both of these attacks.

### *5.2.3 Internal Attackers*

Lastly, the protection of AKES' original defense against yo-yo attacks and ours against yo-yo attacks by internal attackers was assessed as follows. Once more, the topology shown in Figure 4 was used, where all 25 nodes booted at a random point in time during the first 30 virtual minutes, and the frame loss was 0%. This time, the nodes $\{13,14,18,19\}$ acted maliciously by (i) setting $I_{\text{min}} = I_{\text{max}} = 30s$, (ii) choosing $M_{\text{bac}} = T_{\text{ack}} = 5$s, (iii) not replying to UPDATEs, (iv) deleting any new permanent neighbor right after session key establishment, (v) not limiting their number of sent HELLOs, HELLOACKs, and ACKs, and (vi) not communicating among themselves. That is, these nodes frequently sent authentic HELLOs and completed three-way handshakes whenever possible by sending authentic HELLOACKs and ACKs. In three successive runs over 12 virtual hours, the legitimate nodes used Parameter Set 4 through 6 and logged their number of sent HELLOs, HELLOACKs, and ACKs.

The results are shown in Figure 9. Unsurprisingly, our defense against yo-yo attacks protects against yo-yo attacks by internal attackers equally well as against yo-yo attacks by external attackers. Yet, surprisingly, as for AKES' original defense against yo-yo attacks, these results indicate that yo-yo attacks by internal attackers are less effective than collision attacks by external attackers. This is in contrast to HELLO flood attacks, where internal attackers can cause worse repercussions than external attackers when using AKES' original defense against HELLO flood attacks.

### *5.2.4 Discussion*

AKES' resilience to yo-yo attacks comes at the cost of extending $T_{\text{lif}}$. As discussed in Section 3, extending $T_{\text{lif}}$ may deprive AKES from establishing session keys with actual neighbors in situations where a lot of RAM is allocated for storing uncommunicative permanent neighbors. Moreover, an open question is how to choose $T_{\text{lif}}$ so as to achieve a certain level of security. Conversely, our defense against yo-yo attacks enables freeing allocated RAM quickly and can directly be configured to meet any required level of security.

## 6 Related Work

As explained in the introduction, establishing session keys overcomes many issues related to frame counters. An entirely different approach to avoid the mentioned problems with frame counters is available for IEEE 802.15.4 networks that use the timeslotted channel hopping (TSCH) media access control (MAC) protocol of IEEE 802.15.4 [1]. The idea there is to use the index of the timeslot in which a frame is to be sent in lieu of a frame counter. This renders frame counters obsolete. However, TSCH's mechanisms for time synchronization are vulnerable to both internal and external attackers. In fact, Yang et al. pointed out that attacker-controlled nodes can launch various attacks on TSCH's mechanisms for time synchronization [36]. Moreover, TSCH's mechanisms for time synchronization do not even withstand external attackers. For example, jamming so-called enhanced beacons (EBs) constitutes a devastating denial-of-sleep attack on unsynchronized TSCH nodes since this holds unsynchronized TSCH nodes in the energy-consuming receive mode.

Karlof et al. identified HELLO flood attacks as a general attack on wireless sensor and actuator networks [15]. Actually, many protocols for wireless sensor and actuator networks use some kind of HELLO messages to discover neighboring nodes. Thus, a general attack on such protocols is to inject or replay their HELLO messages. Such HELLO flood attacks usually cause receivers to believe that they are in direct communication range of the node that is pretended to be the sender of the injected or replayed HELLO message [15]. In the context of session key establishment, a basic countermeasure is to perform a three-way handshake so as to raise the confidence of being in direct communication range of each other [6, 21]. But, this countermeasure opens the denial-of-sleep vulnerability that injected HELLOs trigger replies such as HELLOACKs. To avoid replying to injected HELLOs, Lim suggested authenticating HELLOs using hash chains [21]. However, when using his approach, it seems that attackers can force victim nodes to perform many hash computations by injecting HELLOs with late deployment intervals, which would be a denial-of-sleep vulnerability, too. Alternatively, AKES simply sheds HELLOs if they arrive in bursts. Furthermore, in a follow-up effort, Krentz et al. performed the shedding of HELLOs already during reception, which further reduces the energy consumption of victim nodes under HELLO flood attacks [18]. Our LBC-based defense against HELLO

flood attacks advances the shedding of HELLOs by reducing its incurred delays to session key establishment.

The feasibility of collision attacks on IEEE 802.15.4 frames was shown by Wood et al. and Wilhelm et al.. Wood et al., on the one hand, configured an off-the-shelf radio chip to issue interrupts upon detecting the synchronization header (SHR) of an IEEE 802.15.4 frame [34]. Then, if an SHR interrupt is issued, the radio chip was instructed to jam for a short period of time. Wilhelm et al., on the other hand, implemented collision attacks based on software-defined radio, which minimizes the time between the analysis of incoming radio traffic and the decision to jam [32]. Both these methods seem applicable to launch collision attacks on AKES.

Related work on mitigating collision attacks focussed on the short-term repercussion that jamming unicast frames ensues energy-consuming retransmissions [26, 17], rather than long-term repercussions as we did. Ren et al., e.g., conceived a reactive defense, which detects collision attacks and, upon detection, temporarily switches to a low-power sleep mode [26]. Unfortunately, they neglected that the transmissions of neighboring nodes fail if a victim node is currently in a low-power sleep mode. Thus, neighbors of a victim node may also suspect a collision attack and enter a low-power sleep mode, too. Consequently, a single collision attack may paralyze a whole network if using Ren et al.'s reactive defense. Besides, Krentz et al. proposed the secure phase-lock optimization (SPLO) to mitigate collision attacks on ContikiMAC [7, 17]. Originally, ContikiMAC strobed unicast frames for long if no information about the wake-up time of the intended receiver is available or if this information seems to be outdated as unicast transmissions to him tend to fail. Thus, a critical denial-of-sleep vulnerability of the original version of ContikiMAC is that repeated collision attacks on unicast frames to a certain receiver not only ensue retransmissions, but also longer strobes. SPLO mitigates this vulnerability by limiting the maximum duration of a strobe of unicast frames throughout a session, regardless of whether unicast transmissions tend to fail. Yet, SPLO does not limit the maximum duration of strobes of HELLOACK and ACK frames and necessitates using short values for $T_{\text{lif}}$. Accordingly, our LBC-based restriction on the number of outgoing HELLOACKs and ACKs complements SPLO very well.

The feasibility of hidden wormhole attacks was demonstrated by Francillon et al. [9]. They achieved delays of the order of nanoseconds by using cut-through forwarding, i.e., by tunnelling incoming frames already during reception. Such kind of hidden wormhole attacks are hard to detect using approaches based on measuring delays. Therefore, more sophisticated methods for detecting hidden wormhole attacks were considered. A recent idea is, e.g., to use channel reciprocity for detecting hidden wormhole attacks, but current channel reciprocity-based wormhole detection schemes incur a high overhead and can be bypassed [19, 14]. As an alternative way of dealing with hidden wormhole attacks, we opted for mitigating the repercussions of such attacks.

## 7 Conclusions and Future Work

Denial-of-sleep attacks pose a severe threat to battery-powered and energy-harvesting IEEE 802.15.4 nodes. While AKES was designed to withstand denial-of-sleep attacks, its denial-of-sleep defenses were presumably never evaluated, yet. We have addressed this gap and have found AKES' denial-of-sleep defenses to successfully mitigate HELLO flood attacks, as well as several kinds of yo-yo attacks, provided one compromises on the speed at which AKES adapts to topology changes. Furthermore, we have devised new denial-of-sleep defenses for AKES and have shown our defenses to accelerate AKES' reaction to topology changes at a low overhead and without security trade-offs. Nevertheless, the tension between responsiveness and denial-of-sleep resilience is not entirely resolved by our defenses. Hence, future work may try to reach even better responsiveness.

## 8 References


[1] IEEE Standard 802.15.4, 2015.

[2] 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.

[3] X. Cao, D. M. Shila, Y. Cheng, Z. Yang, Y. Zhou, and J. Chen. Ghost-in-ZigBee: energy depletion attack on ZigBee-based wireless networks. *IEEE Internet of Things Journal*, 3(5):816–829, 2016.

[4] C.-Y. Chen and H.-C. Chao. A survey of key distribution in wireless sensor networks. *Security and Communication Networks*, 2011.

[5] 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.

[6] J. Deng, C. Hartung, R. Han, and S. Mishra. A practical study of transitory master key establishment for wireless sensor networks. In *Proceedings of the First IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, pages 289 – 302. IEEE, 2005.

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

[8] 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.

[9] A. Francillon, B. Danev, S. Capkun, S. Capkun, and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2011.

[10] 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.

[11] J. Großschädl, A. Szekely, and S. Tillich. The energy cost of cryptographic key establishment in wireless sensor networks. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS '07)*, pages 380–382. ACM, 2007.

[12] K. B. Hein, L. B. Hörmann, and R. Weiss. Using a leaky bucket counter as an advanced threshold mechanism for event detection in wireless sensor networks. In *Proceedings of the Tenth Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pages 51–56. IEEE, 2012.

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

[14] S. Jain, T. Ta, and J. Baras. Wormhole detection using channel characteristics. In *Proceedings of the 2012 IEEE International Conference on Communications (ICC 2012)*, pages 6699–6704. IEEE, 2012.

[15] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Elsevier's AdHoc Networks Journal*, 1(2–3), 2003.

[16] 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.

[17] K.-F. Krentz, Ch. Meinel, and H. Graupner. Countering three denial-of-sleep attacks on ContikiMAC. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN 2017)*, pages 108–119. Junction, 2017.

[18] 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.

[19] 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.

[20] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The Trickle Algorithm. RFC 6206, 2011.

[21] C. H. Lim. LEAP++: a robust key establishment scheme for wireless sensor networks. In *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops (ICDCS '08)*, pages 376–381. IEEE, 2008.

[22] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. E. Culler. SPINS: security protocols for sensor networks. *Wireless networks*, 8(5), 2002.

[23] A. d. l. Piedra, A. Braeken, and A. Touhafi. Extending the IEEE 802.15.4 security suite with a compact implementation of the NIST P-192/B-163 elliptic curves. *Sensors*, 13(8):9704–9728, 2013.

[24] G. Piro, G. Boggia, and L. A. Grieco. Layer-2 security aspects for the IEEE 802.15.4e MAC. Internet-Draft, 2014. Version 3.

[25] S. Raza, T. Voigt, and V. Juvik. Lightweight IKEv2: a key management solution for both compressed IPsec and IEEE 802.15.4 security. In *Proceedings of the IETF International Workshop on Smart Object Security*. IETF, 2012.

[26] 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.

[27] 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.

[28] S. Sciancalepore, G. Piro, E. Vogli, G. Boggia, L. A. Grieco, and G. Cavone. LICITUS: a lightweight and standard compatible framework for securing layer-2 communications in the IoT. *Computer Networks*, 108(Supplement C):66–77, 2016.

[29] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling large-scale storage in sensor networks with the coffee file system. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks (IPSN '09)*, pages 349–360. IEEE, 2009.

[30] 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.

[31] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610, 2003.

[32] M. Wilhelm, I. Martinovic, J. B. Schmitt, and V. Lenders. Short paper: Reactive jamming in wireless networks: How realistic is the threat? In *Proceedings of the Fourth ACM Conference on Wireless Network Security (WiSec '11)*, pages 47–52. ACM, 2011.

[33] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, 2012.

[34] A. Wood, J. Stankovic, and G. Zhou. DEEJAM: defeating energy-efficient jamming in IEEE 802.15.4-based wireless networks. In *Proceedings of the 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '07)*, pages 60–69. IEEE, 2007.

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

[36] 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.

[37] S. Zhu, S. Setia, and S. Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 62–72. ACM, 2003.

# A Open-Source Fixes to AKES

- Originally, AKES did not send a `HELLO` at start up, but only reset Trickle at start up, which resulted in a slower user experience.
- Originally, replayed `HELLO`s were not discarded by receivers, which caused unnecessary `HELLOACK`s to be sent.
- Originally, replayed `HELLOACK`s that pretend to originate from a permanent neighbor were recognized using frame counters. Yet, this led to long deadlocks after reboots. Therefore, replayed `HELLOACK`s are now recognized by checking whether the contained challenge is fresh.
- Originally, `ACK`s from expired tentative neighbors were not discarded.
- Originally, AKES only required $I_{\min}$ to be greater than $M_{\text{bac}}$, which was insufficient to ensure not sending another `HELLO` while still waiting for `HELLOACK`s. Now, AKES chooses $I_{\min}$ so that $\frac{I_{\min}}{2} > M_{\text{bac}}$.