LIDOR : A Lightweight DoS-Resilient Communication Protocol for Safety-Critical IoT Systems

Stute, Milan; Agarwal, Pranay; Kumar, Abhinav et al.

(2020)

LIDOR: A Lightweight DoS-Resilient Communication Protocol for Safety-Critical IoT Systems

Milan Stute*, Pranay Agarwal†, Abhinav Kumar†, Arash Asadi*, and Matthias Hollick*
*Secure Mobile Networking Lab, Technical University of Darmstadt, Germany
{mstute, aasadi, mhollick}@seemoo.tu-darmstadt.de
†IIT Hyderabad, India
{ee15resch11001, abhinavkumar}@iith.ac.in

***Abstract*—IoT devices penetrate different aspects of our life including critical services such as health monitoring, public safety, and autonomous driving. Such safety-critical IoT systems often consist of a large number of devices and need to withstand a vast range of known denial-of-service (DoS) network attacks to ensure reliable operation while offering low-latency information dissemination. As the first solution to jointly achieve these goals, we propose LIDOR, a secure and lightweight multihop communication protocol designed to withstand all known variants of packet dropping attacks. Specifically, LIDOR relies on an end-to-end feedback mechanism to detect and react on unreliable links and draws solely on efficient symmetric-key cryptographic mechanisms to protect packets in transit. We show the overhead of LIDOR analytically and provide the proof-of-convergence for LIDOR which makes LIDOR resilient even to strong and hard-to-detect wormhole-supported greyhole attacks. In addition, we evaluate the performance via testbed experiments. The results indicate that LIDOR improves reliability under DoS attacks by up to 91 % and reduces network overhead by 32 % compared to a state-of-the-art benchmark scheme.**

I. Introduction

Safety-critical IoT systems has become an integrated part of our professional and personal lives in the areas of health monitoring, public safety, and autonomous driving. Such systems require wireless networking solutions that are: (i) scalable, (ii) robust and secure, and (iii) offer low latency (see Fig. 1). To date, these objectives have been individually tackled, however, there is no comprehensive solution that addresses these objectives jointly.

The key to *scalability* is using multi-hop communication with low-overhead routing strategies [1]. In fact, existing practical solutions such as Bluetooth Mesh [2], [3] rely on multi-hop communication but their scalability is limited by a flooding-based routing approach. *Robustness and security* is achieved by protection against attacks on availability, i. e., denial-of-service (DoS) attacks. Furthermore, this protection should be provided both on control plane and data plane. Prior works on joint control and data plane protection [4]–[8] do not provide a provably comprehensive solution against all well-known variants of blackhole and greyhole DoS attacks. Finally, *low latency* is important for safety-critical systems which require timely dissemination of information. Low latency can only be achieved via a low overhead routing protocol design and an efficient implementation especially when using cryptographic operations.

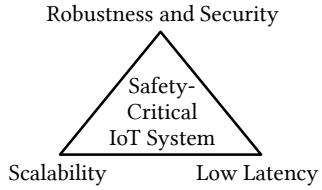


Fig. 1: Design objectives for safety-critical IoT

In this paper, we propose *LIDOR*, a lightweight multi-hop protocol that secures communication among IoT devices. While LIDOR provides authenticated and (optionally) confidential communication, more importantly, it uses an end-to-end feedback mechanism to quickly detect and locally repair broken paths, thus, comprehensively mitigating different variants of DoS attacks. LIDOR's path selection provably converges even in the presence of hard-to-detect wormhole-supported greyhole attacks. By leveraging symmetric-key cryptographic primitives, we ensure efficient operation of LIDOR even on embedded devices [9] leading to low end-to-end delivery delays. We validate our proposal by running testbed experiments. Our main contributions are:

- We present LIDOR, the first multi-hop communication protocol that comprehensively protects against all well-known variants of blackhole and greyhole attacks.
- We analytically prove that LIDOR's communication overhead is lower than Castor [8], which is the most secure multi-hop solution in the state-of-the-art. Although Castor is not the latest work on the topic, it is the strongest work to date which provides protections against different packet dropping attacks. Recent follow-up works pursue solution for specific attacks only [10], [11].
- We provide analytical proof that LIDOR converges even in the presence of a wormhole-supported greyhole attack.
- We implement a LIDOR prototype in C++ using lightweight symmetric cryptographic primitives and make it available as open source software.
- We conduct experiments in our testbed to validate our analytical findings and show that LIDOR does not incur additional overhead under attack and significantly increases delivery rates under attack compared to the Castor protocol.

The rest of this paper is structured as follows: in Section II, we give background information on DoS attacks and related work. In Section III, we introduce our LIDOR protocol. We provide analytical proofs for LIDOR's overhead and convergence in Section IV and Section V, respectively. In Section VI, we shed light on our implementation on which we base our experimental evaluation in Section VII. We discuss our results in Section VIII and finally conclude in Section IX.

II. Background

In this section, we first review major denial-of-service attacks on multi-hop communication schemes and then discuss existing secure multi-hop protocols that thwart (part of) the attack space. We provide an overview of the protocols and their attack resilience in Table I.

A. Denial-of-Service Attacks

We adopt the taxonomy of [12] for attacks on multi-hop communication. We differentiate between attacks on the control plane (path discovery) and attacks on the data plane (payload transmission). Denial-of-Service (DoS) attacks on multi-hop schemes aim to disrupt or completely prevent communication between two target nodes. By manipulating the path discovery mechanism, an attacker might be able to divert traffic (*spoofing* or *replay*) or attract traffic towards itself by exploiting performance-based distance metrics such as hop count or round-trip time (*rushing* [13], *tunneling*, or *wormhole*[1] [14]). In addition, the attacker can operate under multiple identities to evade detection (*Sybil* attack [15]). On the data plane, the attacker can decide to drop all packets (*dropping*), or to only drop temporarily or certain types of packets (*selective dropping*). By combining attacks on control and data plane, an attacker can create a devastating *blackhole* or *greyhole* which essentially attracts and then (selectively) drops packets. Note that, consequently, there are several flavors of blackhole and greyhole attacks that depend on an supporting attack on the control plane.

Regarding attacks aiming at resource starvation, Castor [8] is the only solution that proposes a rate-limiting mechanism tied to the node's reliability to thwart flooding attacks. Since LIDOR uses a similar reliability-based distance metric, adopting Castor's mitigation would be straightforward, but is not discussed in this paper.

B. Related Work

Initial works on routing protocols [16]–[20] only secure the control plane of a communication protocol and, therefore, cannot comprehensively protect against (selective) dropping attacks. Similarly, protocols only protecting the data plane [21], [22], can only detect packet loss but not react on it. Therefore, the best approach against blackhole and greyhole attacks is a holistic approach that protects both control and data plane. We provide a summary of such holistic approaches in Table I. As we are interested in a comprehensive DoS resilience, we briefly point out the drawbacks of each approach in the following.

[1]We consider *rushing* and *tunneling* to be weaker variants of the *wormhole* attack, so we do not explicitly consider them in the following.

TABLE I: Resilience of holistic multi-hop protocols to different variants of (selective) packet dropping attacks. We differentiate between resilient (✓), limited resilient (✓*), not resilient (no mark) and unknown (?).

(Selective) dropping w/ supporting control plane attack	Blackhole w/ Spoofing	Sybil	Replay	Wormhole	Greyhole w/ Spoofing	Sybil	Replay	Wormhole
2ACK [4]	✓		?					
ODSBR [5]	✓		?		✓		?	
Sprout [6]	✓	✓*	?		✓	✓*	?	
BTFR [7]	✓	?	✓	?	✓	?	✓	?
Castor [8]	✓	✓		✓	✓	✓		
LIDOR	✓	✓	✓	✓	✓	✓	✓	✓

2ACK [4] selectively acknowledges data packets and is thus vulnerable to all types of greyhole attacks. In addition, the protocol is vulnerable to colluding attackers. ODSBR [5] uses authentic end-to-end acknowledgments for data packets and resorts to path probing to identify broken links. The latter makes the protocol vulnerable to Sybil and wormhole attacks where a large number of fictitious links are created and all have to be explicitly identified. Sprout [6] uses path probing to evaluate the quality of entire paths instead of links. Since the protocol relies on source routing, the source needs to be able to identify all other nodes. In addition, Sprout was shown to perform worse than Castor under the wormhole attack. BTFR [7] is similar to Sprout in design (source routing and end-to-end acknowledgments). Castor [8] has an elegant design to use end-to-end acknowledgments, and achieves higher resilience against sophisticated attacks such as blackholes and wormholes by incorporating an implicit and independent route discovery.

Although Castor is not the latest work on the topic, it is the strongest work to date which provides protections against different packet dropping attacks. Follow-up works pursue solutions for specific attacks, i. e., spoofing [11], greyhole [10], [23] or Sybil [24]. Because of its comprehensiveness, we opted for Castor [8] as our benchmark. Compared to Castor, we significantly reduce the protocol overhead and complete the protection scope to greyholes supported by replay and wormhole attacks. Finally, we provide experimental evidence on the practicality of our scheme.

III. LIDOR Protocol

In this section, we first provide the requirements and a high-level overview of the LIDOR protocol. Next, we describe the protocol in detail, and, finally, highlight differences from the benchmark protocol.

A. Requirements

Our security assumptions consist of a trust model and an attacker model. We further require each node to perform certain basic cryptographic operations. We elaborate on them in the following.

Trust Model. LIDOR devises an end-to-end communication scheme. Hence source and destination nodes need to trust each

TABLE II: Symbols and Notations

SYMBOL	NOTATION
s	source node
d	destination node
H	flow identifier (root of Merkle tree)
l	height of Merkle tree
n	nonce
b_k	kth packet identifier
$K_{s,d}$	key
m	packet digest
σ	authentication tag
f_k	flow authenticator for the kth packet
a_k	ACK authenticator (preimage of b_k)
T_{ACK}	ACK timeout
$O_k^{\text{B}}(O_k^L)$	overhead for the kth packet for the benchmark (LIDOR) protocol
$O_{\text{B}}(O_L)$	total overhead from all the packets of the flow for benchmark (LIDOR) protocol
ΔO	difference between total overhead in benchmark and LIDOR protocol
η	number of hash values required
t_k	sending time of the kth packet
λ_n	number of packets for which nonce is re-transmitted
p_n	probability of retransmission of nonce
$\mu_{x,j}^H$	average reliability metric for the jth neighbor of the node x for the flow H
$\mu_{x,j}^{\text{all},H}(\mu_{H,j}^{\text{first},H})$	reliability metric for the jth neighbor of the node x for the flow H for all ACKs (first ACK)
$\alpha_{x,j}^{\text{all},H}(\alpha_{x,j}^{\text{first},H})$	proportion of all ACKs (first ACK) received successfully from the jth neighbor of the node x for the flow H
$\beta_{x,j}^{\text{all},H}(\beta_{x,j}^{\text{first},H})$	proportion of all ACKs (first ACK) received unsuccessfully from the jth neighbor of the node x for the flow H
δ	adaptivity parameter
ϵ	suitable threshold on reliability
B_x^M	successive broadcast of M packets by the node x
B^M	successive broadcast of M packets by all the nodes in the network
U_x^y	unicast of a packet from the node x to the node y
Δ_M	the reliability metric of a node after M packets transmitted successfully
$M_{\text{min}}(M_{\text{max}})$	Minimum (maximum) number of packets required to achieve convergence
A	number of attacker nodes in a relay layer
N	number of non-attacker nodes in a relay layer
G_ν	minimum number of broadcasts required in terms of Γ and Υ
I	number of hops
ξ_I	number of packets unicast for I hops

other and be able to share a cryptographic key. LIDOR does not enforce a specific mechanism for key establishment and distribution similar to other works, e. g., they can be mediated by a trusted third party that certifies public keys [5], [6], [8] or via secure device pairing methods [25]. In the IoT scenario, the device manufacturer could pre-deploy certified keys such that devices from the same manufacturer could perform key derivation without an active third party. However, we do not assume trust relationships between source/destination and other intermediate nodes. By not relying on a network-wide key, LIDOR is robust to the compromise of individual nodes, e. g., in case that certain device models expose vulnerabilities.

Adversary Model. In this work, we consider attacks on the classic security triad confidentiality, integrity, and availability. However, LIDOR focuses on DoS attacks. In particular, our adversary is an entity controlling a portion of authenticated nodes within the network. They can consequently take part in normal network operations, but is not limited to, mounting localized jamming, packet injection, modification, and dropping attacks; specific attacks on the forwarding protocol; or any combination thereof (see Section II for details). However, the attacker cannot break cryptographic primitives and we assume that there is at least one attacker-free path between any source–destination pair that wishes to communicate.

Node Capabilities. Each LIDOR node: (i) has access to a pseudo-random number generator rng, (ii) can compute a cryptographic hash function $hash(\cdot)$, (iii) has access to a stream cipher $prf(K, n)$ which takes a key K and some nonce n as inputs, and (iv) can compute authentication tags $tag(K, \cdot)$ based on a shared key K. We discuss practical candidate functions in Section VI-B.

B. LIDOR Overview

LIDOR leverages established concepts [8], [26], [27] to provide DoS-resilient communication. At its core, LIDOR: (i) uses an acknowledgment-based feedback mechanism to rate the reliability of neighbors and effectively detect faulty links, (ii) lets intermediate nodes individually decide whether to conduct path exploration (broadcast) or exploitation (unicast) to quickly react to changes in reliability, and (iii) separates state of different flows that only source and destination nodes can influence to prevent adversarial state pollution. With its generic design, LIDOR is agnostic to the *cause* of disruptions but will detect the *existence* of failures and react on them. Thereby, LIDOR *comprehensively* thwarts any type of dropping attack. Further, LIDOR solely relies on *lightweight* symmetric cryptographic primitives (e. g., hash functions) [9] that are also feasible on less powerful nodes. In particular, we rely on a Merkle tree-based commitment scheme, where all packets are committed to belong to the same flow and the destination reveals the secret only after receiving it in form of an acknowledgment. Since all intermediate nodes are able to verify the secret, they can be sure that the destination has received the packet if we receive the acknowledgment. In the following, we describe LIDOR's workflow in detail and elaborate on packet format and processing.

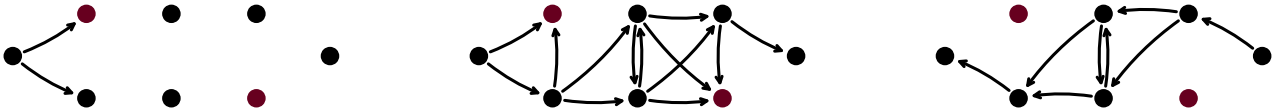
C. LIDOR Workflow

Next, we elaborate on the protocol workflow of LIDOR. The main processing steps are: (i) packet generation, (ii) packet verification, (iii) packet forwarding, (iv) packet reception, and (v) acknowledgment handling. We depict the various stages in Fig. 2.

1) Packet Generation: LIDOR establishes flows for end-to-end communication similar to [8]. The cryptographic material securing each flow is drawn from a Merkle tree, an accepted cryptographic tool for secure multi-hop communication [8], [19], [26], [27]. We first introduce the packet format and then discuss peculiarities of Merkle tree usage and construction.

Packet Format. The LIDOR PKT in Eq. (1) contains source s and destination d identifiers, flow identifier H which is the root of a Merkle tree of height l, the kth PKT identifier b_k, and an authentication tag σ. A nonce n is included until the first ACK of the flow is received. The user payload may be encrypted using the stream cipher $prf(K_{sd}, hash(n + k))$. The σ is computed over all fields except the flow authenticator f_k and length l'. Additional meta data (packet type, length of hash values, and length of the entire packet) is excluded for brevity.

$$PKT = \left\langle s, d, H, b_k, f_k^{l'<l}, n, \mathcal{P}, \sigma \right\rangle \tag{1}$$



(a) The source (1) generates a packet and (3) forwards it to its most reliable neighbor. If reliability is low, it probabilistically broadcasts to all neighbors to explore new paths.

(b) Receiving nodes (2) discard duplicates and verify that the packet belongs to a certain flow. Then, they (3) make forwarding decisions the same way the source node does.

(c) The destination (4) verifies the authenticity of the packet and replies with an acknowledgment on the reverse path. All receiving nodes (5) verify its authenticity and update the reliability rating of their respective neighbors. Neighbors which do not return an acknowledgment receive a penalty.

Fig. 2: Overview of LIDOR's protocol workflow showing which operations are made in which stage: (1) packet generation, (2) packet verification, (3) packet forwarding, (4) packet reception, and (5) acknowledgment handling. Attacker nodes are marked in red and drop all packets in this example.

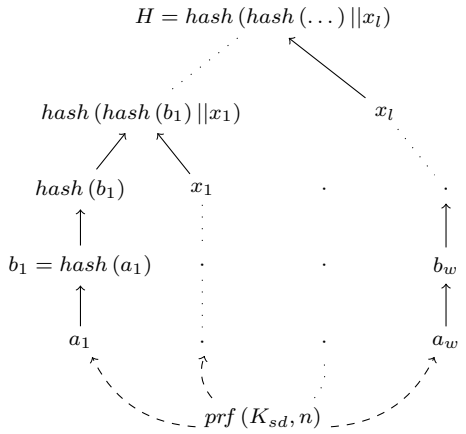


Fig. 3: LIDOR Merkle tree generation. The leaf seeds a_k are drawn from a stream cipher $prf\,(\cdot,\cdot)$, which, in turn, is seeded by a secret key K_{sd} and a public nonce n.

Merkle Tree Usage. LIDOR utilizes Merkle hash trees for packet labeling, flow authentication, and proof of packet reception. In particular, the idea is to use the input values for the tree's leaf nodes as packet identifiers b_k and to commit to them with the root H which is used as a flow identifier. The packets identifiers, in turn, are computed from a secret a_k as $b_k = hash\,(a_k)$. Since PKTs are end-to-end authenticated, the destination node will only reveal the pre-image a_k of the packet identifier b_k for authentic packets in the form of an acknowledgment (ACK) (Section III-C4). Upon reception of a_k, intermediate nodes can deduce that the destination must have received an authentic packet with b_k.

Nonce-seeded Merkle Tree Construction. We construct the LIDOR Merkle tree as follows: (i) we use a unique and random nonce n and use it together with K_{sd} to seed a cryptographically secure pseudo-random number generator $prf\,(\cdot,\cdot)$, e. g., a stream cipher; and (ii) we "chop" the output of $prf\,(K_{sd}, n)$ into w blocks of size $|hash\,(\cdot)\,|$ to create all a_k for $k = 1, \dots, w$ and construct the Merkle tree as shown in Fig. 3. Our unique approach of seeding the Merkle tree with a nonce n enables the source to share all secret values a_k with the destination by just communicating n. Together with the shared key K_{sd}, the destination is able to repeat the Merkle tree construction process and retrieve all a_k. Without (i), the source would need to communicate all a_k individually in a confidential manner that would waste bandwidth as done in [8].

When creating the Merkle tree from n and K_{sd}, we need to assert that n is never reused for any source–destination pair, otherwise replay attacks are possible. Reasonable candidates for n are timestamps or randomly chosen values drawn, e. g., from a system-provided rng function. The drawback of choosing timestamps as nonces is the additional attack vector on time synchronization services such as NTP [28] or GPS [29]. When choosing n purely at random, n must be large enough to avoid nonce reuse due to the well-known "birthday problem." We choose to implement the second option with a random 192-bit nonce.

The tree size w is optimally chosen such that it is equal to the number of packets a source node wishes to transmit for a certain flow. If this number is known a priori, w can be approximated and fed into LIDOR as an optimization. In all other cases, LIDOR has to rely on a default tree size. However, choosing the default tree size incurs a trade-off: (i) the length of the flow authenticator included in every packet grows logarithmically with the tree size, but (ii) very small trees cause frequent flow restarts, i. e., whenever all b_k have been used, a new tree must be created and the route exploration process restarts. In the Section III-C2, we propose a countermeasure for (i) in the form of an in-network compression mechanism that can reduce the average overhead of the flow authenticator to a constant factor (Section III-C2).

2) Packet Verification: Next, a node filters out the PKTs that either have already been forwarded (i.e., duplicates) or contain an invalid flow authenticator. In addition, a node computes the minimal authenticator length for the next hop node.

Duplicate Detection. Duplicate detection consists of two steps. First, a node calculates a packet digest m using a collision-resistant hash function of the incoming PKT (excluding the variable-length field f_k). This serves for identifying unique PKT copies which might have the same packet identifier b_k. If the node has already seen the pair $\langle m, b_k \rangle$, the PKT is dropped. To prevent replay attacks, each node

keeps per-flow state to memorize which PKTs have already been acknowledged for preventing replay attacks. A very low-complexity and space-efficient implementation of such a data structure is a zero-initialized bit vector. Setting bit k in the bit vector signifies that the kth PKT of a certain flow is acknowledged, and, thus, future replays of b_k can be ignored. Specifically, a node checks whether PKT k of the indicated flow has already been acknowledged (kth bit set), and if yes, discards it. The effectiveness of our replay protection mechanism in shown Section VII.

Flow Authentication. The Merkle tree assures that all b_k can be authenticated to a single value, that is, the root H which serves as a flow identifier. Intermediate nodes can validate that b_k belongs to H by traversing the tree from b_k to the root H' using intermediate tree nodes f_k and checking that $H' = H$. The flow authentication procedure has been described in [8] and assures that only PKTs belonging to the flow will be forwarded.

3) Packet Forwarding: Here, we describe the forwarding decision that is based on a reliability metric. We further discuss the in-network Merkle tree compression to reduce network overhead and explain the purpose of the PKT timer.

Reliability Metric. In contrast to the per-destination routing state used in classic MANET protocols, LIDOR keeps the forwarding state per flow. In addition, LIDOR nodes maintain separate per-neighbor reliability metrics for every encountered flow. The reliability metric $\mu^H_{i,j} \in [0,1]$ of a node i for its neighbor j for the flow H is computed as a running average of the PKT delivery rate (i. e., the number of valid ACKs returned from a neighbor). It has been shown in [8], [30] that this approach provides lightweight protection against any type of accidental and deliberate packet loss including hard-to-detect selective packet dropping, i. e., greyhole attacks. In Section V, we discuss the calculation of the reliability metric in detail. We further prove that previous approaches [8] are not secure, i. e., they do not converge towards an attacker-free path if attackers are present in the network. Also, we prove that LIDOR's approach converges.

Forwarding Decision. This decision is made probabilistically based on the reliability metric. A node forwards a PKT with probability $1-r$ using a broadcast transmission or with probability r using a unicast to the most reliable neighbor. Should two or more neighbors have the same reliability metric, we use the average round-trip time to break the tie. The intuition is that we use broadcast for *route exploration* if no reliable path exists, and otherwise unicast for *route exploitation*.

In-Network Merkle Tree Compression. The size of the flow authenticator f_k has a drastic impact on the protocol overhead. f_k grows linearly with the tree height l. In previous works [8], [19], [31], all sibling nodes in the tree from the leaf to the root (tree nodes $x_1, \ldots, x_l$ in Fig. 3) are included in each packet. For large trees, this naïve approach generates significant overhead. For instance, a tree of height $l = 8$ allowing to send $2^8 = 256$ PKTs under that flow requires the header to include 8 hash values for f_k. In absolute terms, these results in $8 \times 16 = 128$ bytes *per PKT* when using a collision-resistant hash function with a 16-byte output.

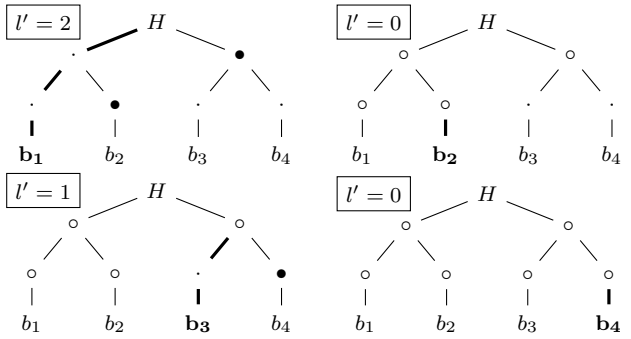


Fig. 4: Exemplary Merkle tree ($w = 4$) visualizing optimal flow authenticator lengths l' for different PKT identifiers b_k. Bullets (•) indicate tree nodes that have to be included in PKT k. Circles (○) are known tree nodes sent in previous PKTs. Dots (·) are unknown nodes but are not required to authenticate b_k. Thick lines indicate the verification path.

LIDOR employs a more efficient method: LIDOR nodes incrementally construct the Merkle tree as they receive new b_k and f_k values (note that intermediate nodes cannot construct the entire tree from n since they do not possess K_{sd}). Starting from the second received PKT, $l' < l$ new tree nodes are required to authenticate the flow. The idea is visualized in Fig. 4. In a stable network, the lower bound average of l' is *constant* with $(2^l - 1)/2^l < 1$ which leads to an 8-fold overhead reduction compared to sending the full authenticator length.

In order to devise a practical distributed algorithm to calculate l', nodes need to keep track of the current Merkle tree state of their neighbors. LIDOR nodes do this by leveraging the ACKs received from their neighbors: when receiving a_k from neighbor h, a node knows that h has the kth leaf of the Merkle tree, as well as the authenticated path from this leaf to the root. Otherwise, h would have been unable to authenticate and forward the kth PKT in the first place. To determine minimal l', i. e., the shortest possible flow authenticator length for which the next-hop node will still be able to authenticate b_k, we use Algorithm 1. For broadcast PKTs, we set l' to the maximum among all neighbors, i. e., $l' = \max_h l'_h$ with l'_h being the minimal flow authenticator length for neighbor h.

This scheme assures that (i) a node can always authenticate any PKT received from another correct node; and (ii) the transmitted flow authenticator does not convey redundant information, i. e., it is exactly as long as it needs to be for minimal PKT overhead. Note that our scheme is agnostic to packet loss and packet reordering.

Algorithm 1 Minimal Flow Authenticator Length

```
function MINAUTH(k, l, h)
    l' ← 0
    while l' < l do
        k_left ← (k ⊕ (1 ≪ l)) ∧ (−1 ≪ l)
        k_right ← k_left + (1 ≪ l) − 1
        for k' ∈ [k_left, k_right] do
            if h has acknowledged k' then
                return l'
        l' ← l' + 1
    return l
```

In the rare case that a node loses state and is unable to authenticate the flow because f_k is too short, it may "bounce" the PKT back to the sender with a unicast to request for a retransmission with the complete f_k. The receiving node then removes the flag and returns the complete f_k of length l to the requester. This retransmission may only be done once per neighbor and PKT to prevent DoS attacks where an attacker would effectively circumvent the duplicate check.

Awaiting Response. When forwarding a PKT, a node starts one timer for each new b_k which expires after T_{ACK}. In addition to starting the timer, the tuple $\langle m, b_k, H\rangle$ together with the forwarding decision is added to a collection of previously seen PKTs. This tuple serves the purpose of authenticating ACKs (as described in Section III-C5) and it is discarded after the PKT timer expires. If the timer expires and no ACK has been received, the reliability metric for the next hop node is decreased. To avoid premature false positives (timer expires before ACK was returned) or late true positives (lost ACK is detected too late), we employ an adaptive TCP-inspired timeout calculation for T_{ACK} following the same approach as in [26].

4) Packet Reception: In addition to verification, the destination node checks the PKT's σ. PKTs with an invalid σ are discarded. For the first PKT of a flow, the destination locally computes the full Merkle tree using the nonce n as described in Section III-C1. For every PKT, the destination selects a_k from the Merkle tree and generates the appropriate ACK (Eq. (2)) which consists of the packet digest m and the ACK authenticator a_k. The ACK is then returned to the sender.

$$ACK = \langle m, a_k\rangle \tag{2}$$

5) ACK Handling: ACKs are primarily meant as secure proofs of delivery used to update the neighbor reliability metrics. Upon ACK reception, a node calculates $b_k = hash\,(a_k)$ and checks whether the ACK belongs to a valid PKT, i. e., whether any PKT with m and b_k has been forwarded before. If not, the ACK is dropped. Otherwise, and if the sending node matches the previous forwarding decision, the reliability metric for the sending node is increased. The ACK is then forwarded to the neighbors from which the node received copies of the corresponding PKT. If we receive multiple ACKs, only the first one is forwarded. In addition, a valid ACK updates the bit vector used for duplicate detection and neighbor Merkle tree state as described in Section III-C2.

D. Key Differences to Benchmark

We have described our LIDOR protocol in detail. In the following, we discuss the distinct differences to the benchmark protocol [8]. In particular, LIDOR differs from the benchmark in the following key points:

- We employ an effective construction and in-network compression of the Merkle tree. Especially with the in-network compression, we are able to reduce the average packet overhead from a logarithmic to a *constant* factor (with respect to the tree size) in a static setting. We provide the proofs in the overhead analysis in Section IV.
- We provide proper protection against replay attacks. In particular, we keep a list of seen PKT identifiers even after the ACK timeout in form of a space-efficient bitvector as described in Section III-C2.
- We design a reliability metric that will converge even in the presence of a strong wormhole-supported greyhole attack. We provide a proof for non-convergence of the benchmark and a proof of convergence for LIDOR in Section V for a static network topology.

IV. Overhead Analysis

In this section, we present a comparative analysis between the overhead of the benchmark and LIDOR protocol for a converged scenario. For the purpose of this analysis, we consider the number of hash values in the flow authenticator f_k as the overhead of the system. The overhead is added for each hop. In the converged scenario, a stabilized path exists between the source and the destination. Thus, all the nodes in the stabilized path will unicast the packet to their most reliable neighbor. This implies that the recipient of consecutive packets from a node remains the same.

Let I denote the number of hops between the source and destination nodes. Let l be the height of Merkle Tree. Therefore, the number of packets transmitted for a flow of Merkle tree is given by 2^l. Let $|hash(.)|$ denote the size of one hash value in bytes.

A. Benchmark Protocol

In the benchmark protocol, the source transmits all the hash values for each packet. The total number of hash values required for authenticating the packet is same as the height of the Merkle tree, l. Let O_k^B denote the overhead for the k^{th} packet of the flow. Then,

$$\begin{aligned} O_k^B &= \sum_{i=1}^{I} l|hash(.)| + |e_k| \ , \\ &= I(l|hash(.)| + |e_k|) \ , \end{aligned}$$

where $|e_k|$ denotes the size of the hash value of an encrypted ACK e_k in bytesas used in [8]. Let O_B denote the total overhead of the benchmark protocol. Then,

$$O_B = \sum_{k=1}^{2^l} O_k^B = (2^l I)(l|hash(.)| + |e_k|) \ . \tag{3}$$

B. LIDOR Protocol

In LIDOR, the first packet of the flow carries a nonce which is used by the destination node to re-construct the Merkle tree. The intermediary nodes require l hash values of the Merkle tree to verify if the packet belongs to the same flow. Thus, the overhead for the first packet, denoted by O_1^L, is given by

$$O_1^L = Il|hash(.)| + I|n| \ , \tag{4}$$

where, $|n|$ is the size of nonce in bytes. The number of hash values required for authentication may reduce if the recipient of the current packet had previously received one or more

packets belonging to the same flow. Since the network is converged which implies each node sends all the packets to the same node, it can save upon the number of hash values required for transmission as shown in Algorithm 1.

Let η denote the number of hash values required to be transmitted when all the packets are sent to the same node, where η is a non-negative integer in the range $[0, l]$. Then,

$$\sum_{k=1}^{2^l} \eta = 2^l - 1 \ . \tag{5}$$

This implies that the total number of hash values to be transmitted when all the packets are sent to the same node is given by $2^l - 1$.

The nonce is retransmitted in future packets if $t_k + RTT > t_{k+1}$ where t_k is the sending time of the k^{th} packet and RTT denotes the round-trip-time. The best case is if the nonce is transmitted only for the first packet. Whereas, the worst case is if the nonce is transmitted for each packet of the flow. Therefore, we assume that nonce is transmitted for $2 \leq k \leq \lambda_n$ packets, where $\lambda_n (\leq 2^l - 1)$. Thus, the probability of retransmission of nonce, denoted by p_n, is given by

$$p_n = \frac{\lambda_n}{2^l - 1} \ .$$

Then, O_k^L for $2 \leq k \leq 2^l - 1$ is given by

$$O_k^L = I(\eta |hash(.)| + p_n |n|) \ . \tag{6}$$

Let O_L denote the overhead of LIDOR protocol for a flow. Then, from (4), (5) and (6), we have

$$\begin{aligned} O_L &= \left[\sum_{i=1}^{2^l} \eta \right] I|hash(.)| + (1 + (2^l - 1)p_n) I|n| \ , \\ &= I((2^l - 1)|hash(.)| + (1 + (2^l - 1)p_n)|n|) \ . \end{aligned} \tag{7}$$

Let ΔO denote the difference of the benchmark and LIDOR protocol. From (3) and (7), ΔO is given by

$$\begin{aligned} \Delta O &= I|hash(.)| \left(2^l(l-1) + 1\right) + \\ &\quad \left(2^l(|e_k|) - \left(1 + (2^l - 1)p_n\right)|n|\right) \ . \end{aligned}$$

V. Convergence Analysis

In this section, we present the lower and upper bounds on the number of packets required to achieve convergence in the benchmark and LIDOR protocol under a greyhole attack.[2] We assume reliable wireless transmissions, i. e., there is no loss on the channel. The network consists of attacker and non-attacker nodes. Unlike a non-attacker node, the attacker node drops the packet if the packet has been unicast to it. However, both non-attacker and attacker node forward the packet and provide ACK in case of broadcast.

Let s and d be the source and destination nodes respectively. Let μ_s^H denote the maximum reliability value among all the one-hop neighbors of s for the flow H. We say that the network has achieved convergence when (i) the reliability of a non-attacker node is maximum, i.e., μ_s^H corresponds to a non-attacker node (ii) μ_s^H does not decrease, (iii) $\mu_s^H \geq 1 - \epsilon$, where $\epsilon (\approx 0)$ is a suitable threshold on the reliability of the network. Let B_x^M denote the successive broadcast of M packets by the node x and B^M denotes the successive broadcast of M packets by all the nodes. Let U_x^y denote the unicast of a packet from the node x to the node y in the system.

[2]Note that LIDOR and the benchmark are resilient to blackhole attacks. An attacker that drops all packets would effectively remove itself from the network.

A. Non-Convergence of Benchmark Protocol

In this section, we present the convergence analysis for the benchmark protocol. In the benchmark protocol, each node computes a reliability metric for a flow H for j^{th} neighbour. Let $\mu_{x,j}^H$ denote the reliability metric computed by the node x for its j^{th} neighbor for the flow H. Then,

$$\mu_x^H = \max_j \mu_{x,j}^H \ .$$

Whereas, $\mu_{x,j}^H$ is given by

$$\mu_{x,j}^H = \frac{\mu_{x,j}^{\text{all},H} + \mu_{x,j}^{\text{first},H}}{2} \ , \tag{8}$$

where $\mu_{x,j}^{\text{all},H}$ and $\mu_{x,j}^{\text{first},H}$ denote the reliability of 'all ACK' and 'first ACK' respectively for the j^{th} neighbor of the node x and flow H. Then, $\mu_{x,j}^{\text{all},H}$ is computed as

$$\mu_{x,j}^{\text{all},H} = \frac{\alpha_{x,j}^{\text{all},H}}{\alpha_{x,j}^{\text{all},H} + \beta_{x,j}^{\text{all},H}} \ , \tag{9}$$

where $\alpha_{x,j}^{\text{all},H}$ and $\beta_{x,j}^{\text{all},H}$ is the proportion of the packets delivered successfully and unsuccessfully respectively for the j^{th} neighbor of the node x and flow H. The $\alpha_{x,j}^{\text{all},H}$ and $\beta_{x,j}^{\text{all},H}$ update for each unsuccessful transmission as

$$\begin{aligned} \alpha_{x,j}^{\text{all},H} &\leftarrow \delta \alpha_{x,j}^{\text{all},H} \ , \\ \beta_{x,j}^{\text{all},H} &\leftarrow \delta \beta_{x,j}^{\text{all},H} + 1 \ . \end{aligned} \tag{10}$$

Whereas, $\alpha_{x,j}^{\text{all},H}$ and $\beta_{x,j}^{\text{all},H}$ update for each successful delivery as

$$\begin{aligned} \alpha_{x,j}^{\text{all},H} &\leftarrow \delta \alpha_{x,j}^{\text{all},H} + 1 \ , \\ \beta_{x,j}^{\text{all},H} &\leftarrow \delta \beta_{x,j}^{\text{all},H} \ . \end{aligned} \tag{11}$$

The parameter δ controls the adaptability of the network and $0 < \delta < 1$. Similarly, $\mu_{x,j}^{\text{first},H}$ can be computed as

$$\mu_{x,j}^{\text{first},H} = \frac{\alpha_{x,j}^{\text{first},H}}{\alpha_{x,j}^{\text{first},H} + \beta_{x,j}^{\text{first},H}} \ . \tag{12}$$

The $\alpha_{x,j}^{\text{first},H}$ and $\beta_{x,j}^{\text{first},H}$ update in the same manner as $\alpha_{x,j}^{\text{all},H}$ and $\beta_{x,j}^{\text{all},H}$ for both successful and unsuccessful transmission. Initially, $\alpha_{x,j}^{\text{all},H} = \alpha_{x,j}^{\text{first},H} = 0 \ \forall x, j$ and $\beta_{x,j}^{\text{all},H} = \beta_{x,j}^{\text{all},H} = 1 \ \forall x, j$. Let M denote the number of packets broadcast by the source node. If we assume that all the packets are received successfully, the reliability of the neighbors of the source increase for all the packets. Let Δ_M denote the reliability

of the neighbors after the transmission of M packets. Then, using (9) and (11), Δ_M is given by

$$\Delta_M = \left(\frac{\sum_{i=0}^{M-1} \delta^i}{\sum_{i=0}^{M} \delta^i} \right) \ . \tag{13}$$

Let us consider that the nodes x_1 and x_2 connect s and d via two hops. Let x_1 be a non-attacker node and x_2 be an attacker node. Consider $B^1,\ U_s^{x_2},\ B^1$ as a packet transmission scenario. We assume that x_2 provides first ACK for both broadcast. Then, using (8) – (13), we have

$$\mu_{s,x_1}^H = \frac{1}{2}\Delta_2 \ ,$$
$$\mu_{s,x_2}^H = \frac{\delta^2+1}{\delta^3+\delta^2+\delta+1} = \frac{1}{1+\delta} \ .$$

Clearly, $\mu_{s,x_1}^H < \mu_{s,x_2}^H$. Therefore, if s selects to unicast, it will unicast only to x_2. Consider the case when s broadcast after $N-1$ successive $B^1,\ U_s^y$ transmissions and x_2 always provides the first ACK. Then, μ_{s,x_1}^H and μ_{s,x_2}^H after the N^{th} broadcast,

$$\mu_{s,x_1}^H = \frac{1}{2}\Delta_N \ ,$$
$$\mu_{s,x_2}^H = \frac{\sum_{i=0}^{N-1} \delta^{2i}}{\sum_{i=0}^{2N-1} \delta^i} = \frac{1}{1+\delta} \ . \tag{14}$$

Let Z denote the difference of μ_{s,x_1}^H and μ_{s,x_2}^H. Using (14), we have

$$\begin{aligned} Z &= \frac{1}{2}\Delta_N - \frac{1}{1+\delta} \ , \\ &= \frac{1-\delta^N}{2(1-\delta^{N+1})} - \frac{1}{1+\delta} \ , \\ &= \frac{(\delta-1)(1+\delta^N)}{2(1+\delta)(1-\delta^{N+1})} \ . \end{aligned} \tag{15}$$

Since $\delta < 1$, $Z < 0$ for any N, which implies $\mu_{s,x_1}^H < \mu_{s,x_2}^H$. Thus, there is a possibility that the network gets stuck in the loop of $B^1,\ U_s^{x_2}$ when x_2 only provides the first ACK. This implies that there exists a possibility that s never converges to the non-attacker neighbor, i.e. x_1. Next, we describe the convergence in the LIDOR protocol wherein each node will converge to a non-attacker neighbor.

B. Convergence of LIDOR Protocol

LIDOR does not differentiate between $\mu_{x,j}^{\text{all},H}$ and $\mu_{x,j}^{\text{first},H}$, i.e. $\mu_{x,j}^H = \mu_{x,j}^{\text{all},H} = \mu_{x,j}^{\text{first},H}$ and updates $\mu_{x,j}^H$ for all received ACKs. However, the procedure of updating $\mu_{x,j}^{\text{all},H}$ and the procedure of selecting whether to broadcast or unicast a packet is the same as in the benchmark. In case that two or more neighbors have same reliability value, a round-trip time estimation from past transmissions similar to TCP is used to break the tie. Next, we present the convergence analysis with and without attackers.

No Attackers. In this scenario, we assume that all nodes are non-attacker. Initially, s broadcast the packet to all its neighbors. Since, there is no loss, the reliability will increase for all the neighbors of s. After the broadcast of a few packets, s will perform unicast to the node with least round-trip-time. Let x be the neighbor of s which has least round trip time. Thus, once s performs unicast to x, the reliability of x becomes more than the reliability of any other neighbor of s. Since the reliability of x can only increase, s will perform unicast to x with high probability and hence converge to x. This implies the node which connects s and d in the least number of hops and has the lowest round-trip time will be chosen as a unicast forwarder.

Considering B_s^{M-1} and U_s^x, we have $\mu_s^H = \Delta_M$. Then, from the definition of convergence, we have

$$\mu_s^H = \Delta_M \geq 1-\epsilon, \tag{16}$$

Substituting (13) into (16), we have

$$\left(\frac{\sum_{i=0}^{M-1} \delta^i}{\sum_{i=0}^{M} \delta^i} \right) \geq 1-\epsilon \ ,$$
$$\delta^M \leq \epsilon \frac{1-\delta^{M+1}}{1-\delta} \ ,$$
$$M \geq \frac{1}{ln(\delta)} ln\left(\frac{\epsilon}{\epsilon\delta+1-\delta} \right) \ .$$

Let M_{min} denote the minimum number of packets to attain convergence for the source node in the absence of attackers. Then,

$$M_{\text{min}} = \frac{1}{ln(\delta)} \left[ln\left(\frac{\epsilon}{\epsilon\delta+1-\delta} \right) \right] \ . \tag{17}$$

Attackers with 1 hop. In this scenario, we consider that s and d has one layer of relay nodes between them, i.e. s and d are connected in two hops via relay nodes. The layer of relay nodes consists of N non-attacker and A attacker nodes. Let x_i denote the i^{th} non-attacker node for $i \in \{1,2,...,N\}$. Similarly, let x_i denote the i^{th} attacker node for $i \in \{N+1, N+2, ..., N+A\}$. The reliability increases on the unicast for each x_i for $i \in \{1,2,...,N\}$ whereas the reliability decreases on the unicast for each x_i for $i \in \{N+1, N+2, ..., N+A\}$. Therefore, once x_i for $i \in \{N+1, N+2, ..., N+A\}$ receives a unicast, its reliability decreases and hence it will not receive any unicast in future.

We consider that s has transmitted $M(>A)$ packets. Considering all possible combinations of broadcast and unicast, the reliability of the most reliable non-attacker node lies between $[\Delta_{M-A}, \Delta_M]$. The reliability of Δ_M corresponds to the best case path wherein the most reliable node has received the first unicast. It also corresponds to the sequence of M broadcasts, which is less probable. The reliability of Δ_{M-A} corresponds to the worst case path wherein there had been a unicast to each of the A attacker nodes. Then, the upper bound on the number of packets required for convergence is computed by considering the worst case reliability of the non-attacker node, i.e., Δ_{M-A}. Therefore, from the definition of convergence, we have

$$\Delta_{M-A} \geq 1-\epsilon \ . \tag{18}$$

From (13), we have

$$\Delta_{M-A} = \left(\frac{\sum_{i=0}^{M-A-1} \delta^i}{\sum_{i=0}^{M-A} \delta^i} \right) \ . \tag{19}$$

Let $M_{\max}$ denote the number of packets required for convergence in the worst case. Substituting (19) into (18), we have

$$\frac{\sum_{i=0}^{M-A-1}\delta^i}{\sum_{i=0}^{M-A}\delta^i} \geq 1-\epsilon \ ,$$
$$\delta^M \leq \frac{\epsilon\delta^A}{\epsilon\delta+(1-\delta)} \ ,$$
$$M_{\max} = \frac{1}{ln(\delta)}\left[ln\left(\frac{\epsilon\delta^A}{\epsilon\delta+(1-\delta)}\right)\right] \ . \quad (20)$$

The lower bound on the number of packets required for convergence is obtained by considering the best case reliability, i.e. Δ_M. Therefore, the lower bound on the number of packets required for convergence, denoted by $M_{\min}$, is as given by (17). From (17) and (20), we have

$$\begin{aligned} M_{\max} - M_{\min} &= \frac{1}{ln(\delta)}\left(ln(\epsilon\delta^A) - ln(\epsilon)\right) \ , \\ &= \frac{1}{ln(\delta)} ln(\delta^A) \ = A \ . \end{aligned}$$

Thus, $M_{\max} = M_{\min} + A$. Since there are A attacker nodes to which unicast can happen only once, the convergence gets delayed by A packets if s happens to choose the worst case path, i.e., s performs unicast to each attacker node.

Attackers with I hops. In this section, we present the analysis for the network containing N non-attacker and A attacker nodes for $I-1$ layers of relay nodes between s and d. Let us consider the network with $I = 3$. Let x_i and y_i , where $i \in \{1, ..., N\}$, denote the i^{th} non-attacker node in the first and second layer of relay nodes respectively. Let x_i and y_i , where $i \in \{N+1, ..., N+A\}$, denote the i^{th} attacker node in the first and second layer of relay nodes respectively. The μ^H_{s,x_i} for any non-attacker node (i.e. $i \in \{1, ..., N\}$) decreases if x_i unicast to any attacker node y_j where $j \in \{N+1, ..., N+A\}$. Therefore, each non-attacker node can have A unsuccessful unicast attempts. Therefore, the worst case path for the network with $I = 3$ consists of A iterative cycles of successive broadcast followed by a unicast to each attacker node by the source node and an unsuccessful attempt of each non-attacker node being unicast by the source node. A unicast to each attacker node and a series of successive follows the end of the iterative cycle. Let us consider $N = 2$ and $A = 3$. The worst case path for this network is $B^{M_{\min}}$, $U_s^{x_3}$, $U_s^{x_4}$, $U_s^{x_5}$, $U_s^{x_1}U_{x_1}^{y_3}$, $U_s^{x_2}U_{x_2}^{y_3}$, B^{G_1}, $U_s^{x_3}$, $U_s^{x_4}$, $U_s^{x_5}$, $U_s^{x_1}U_{x_1}^{y_4}$, $U_s^{x_2}U_{x_2}^{y_4}$, B^{G_2}, $U_s^{x_3}$, $U_s^{x_4}$, $U_s^{x_5}$, $U_s^{x_1}U_{x_1}^{y_5}$, $U_s^{x_2}U_{x_2}^{y_5}$, B^{G_3}, $U_s^{x_3}$, $U_s^{x_4}$, $U_s^{x_5}$.

The μ^H_{s,x_i} $\forall i$ decreases and becomes equal before $B_s^{G_\nu}$, where $\nu = \{1, 2, 3\}$. G_ν represents the minimum number of broadcast to be performed by all the nodes such that $\mu^H_{s,x_i} \geq 1-\epsilon$ $\forall i$. Then, G_1 is given by

$$G_1 = \frac{1}{ln(\delta)} ln\left(\frac{\frac{\epsilon}{1-\delta}}{\delta^{M_{\min}+1}\left(1+\frac{\epsilon\delta}{1-\delta}\right)+1}\right) \ . \quad (21)$$

Whereas, G_ν for any $\nu > 1$ is given by

$$\Gamma_\nu = \sum_{m=1}^{\nu-1}\left(\delta^{\left(\sum_{l=\nu-m}^{\nu-1} G_l\right)+m}\right) \ ,$$
$$\Upsilon_\nu = \delta^{M_{\min}+\nu+\left(\sum_{l=1}^{\nu-1} G_l\right)}\left(1+\frac{\epsilon\delta}{1-\delta}\right)+1+\Gamma_\nu \ ,$$
$$G_\nu = \frac{1}{ln(\delta)} ln\left(\frac{\frac{\epsilon}{1-\delta}}{\Upsilon_\nu}\right) \ . \quad (22)$$

Then, using (21) and (22), the upper bound on the number of packets required for convergence for a network with $I = 3$ hops is given by

$$M_{\max} = M_{\min} + (A+1)(A) + NA + \left(\sum_{\nu=1}^{A} G_\nu\right) \ .$$

In general, for a network with I hops, (i) The number of unsuccessful unicast attempts for each non-attacker nodes is given by the number of unicast packets for the network with $I-1$ hops. (ii) The number of unicast performed by the source node to each attacker node is one additional the number of unicast performed in the network with $I-1$ hops. (iii) The number of successive broadcasts is given by the number of unicast for the network with $I-1$ hops.

Let ξ_I denote the number of unicast packets in the network with I hops. Then, $\xi_2 = A$, $\xi_3 = A(A+1) + NA$, and $\xi_I = (N+A)\xi_{I-1} + A$.

On solving further, we obtain

$$\begin{aligned} \xi_I &= (N+A)^{I-1}\xi_2 + \left(\sum_{i=0}^{I-2}(N+A)\right)A \ , \\ &= (N+A)^{I-1}A + \left(\sum_{i=0}^{I-2}(N+A)\right)A \ , \\ &= \left(\sum_{i=0}^{I-1}(N+A)^i\right)A \ , \\ &= \left(\frac{(N+A)^I - 1}{N+A-1}\right)A \ . \end{aligned} \quad (23)$$

Then, using (21), (22) and (23), the upper bound on the number of packets required for network with I hops is given by

$$M_{\max} = M_{\min} + \xi_I + \left(\sum_{\nu=1}^{\xi_{I-1}} G_\nu\right) \ . \quad (24)$$

VI. Implementation

We choose the Click modular router [32] for LIDOR implementation to allow for a realistic evaluation on both real hardware and simulation. In this section, we discuss suitable candidate functions for LIDOR's crypto primitives and devise a practical link-local broadcast authentication scheme based on symmetric cryptography.

TABLE III: Computation time in µs of several cryptographic algorithms on various platforms for 1024-byte strings averaged over 10000 runs. Ed25519 is a state-of-the-art elliptic-curve signature scheme and included as a reference.

CLASS	ALGORITHM	ALIX	APU	NEXUS	MAC
$hash(\cdot)$	SHA-256	184	36	18	6
	Blake2b	167	8	29	3
$prf(\cdot,\cdot)$	XSalsa20/20	97	12	12	5
$tag(\cdot,\cdot)$	SipHash-2-4	66	4	8	1
	HMAC-SHA-512	417	35	92	6
	Ed25519 (verify)	8761	1479	815	168

A. Reference Platforms

We evaluate LIDOR on heterogeneous platforms with different CPU architectures, processing capabilities, memory configurations (256 MB to 16 GB RAM), and operating systems (Debian Linux, Android 6, macOS 10.11). In particular, these are: *ALIX* [33], *APU* [34], LG *Nexus* 5, and *MacBook* Pro (early 2015).

B. Cryptographic Primitives

The choice of efficient cryptographic primitives is imperative for any practical communication protocol. In LIDOR, cryptographic operations consume the longest processing time during packet forwarding and constitute the major portion of the communication overhead. Our implementation relies on primitives provided by the lightweight, cross-platform libsodium (v1.0.11) [35]. A performance comparison between different candidate algorithms on our reference platforms is shown in Table III. The table also gives an intuition on why public key crypto is unsuitable to be used on a per-PKT basis: the cumulated forwarding delay would be unacceptably large. We select LIDOR's cryptographic primitives as follows:

- $hash(\cdot)$ is implemented as Blake2b with an output size of 16 bytes. Note that the hash function used to construct the Merkle tree does not need to be collision resistant but only preimage and second-preimage resistant.[3] Hence, a 128-bit security margin is sufficiently large.
- $prf(K_{sd}, n)$ is implemented as XSalsa20/20, a stream cipher using 256-bit keys and 192-bit nonces.
- $tag(K_{sd}, \cdot)$ is implemented as SipHash-2-4 [36], which generates small 8-byte authentication tags for short-input (order of kilobytes) packets using a shared secret K_{sd}.

C. Practical One-hop Broadcast Authentication

LIDOR requires neighbor-to-neighbor communication to be authenticated to prevent blackmailing and Sybil attacks. Cryptographic methods to authenticate broadcast communication are either based on digital signatures or on TESLA [37], which is based on symmetric-key cryptography and delayed key disclosure to achieve asymmetry. We deem both approaches impractical since digital signatures are computationally expensive; and TESLA requires time synchronization between all nodes, and introduces authentication delay which would impede LIDOR's reactiveness to path changes. The small output size of SipHash enables us to implement a one-hop broadcast authentication scheme based on symmetric-key cryptography without TESLA's deficiencies: a forwarding node computes authentication tags for each neighbor $h \in \mathbb{F}$ (excluding the sender) and appends all of them to the PKT. A receiving node then tries to authenticate every tag. If any of them succeeds, the PKT is processed, and otherwise discarded. The expected number of SipHash calculations at a receiving node is $|\mathbb{F}|/2$, the communication overhead is $|\mathbb{F}| \times |tag(\cdot,\cdot)|$. We argue that this scheme is practical since: (i) the number of neighbors is typically low compared to the total number of nodes in the network (which is what TESLA was designed for), so the communication and computational overhead for transmitting and verifying all tags remains low on average; and (ii) broadcasts are used for route exploration which rarely occurs in established communication flows.

[3] *Collision resistance*: given $hash(\cdot)$, it is hard to find x and x' such that $hash(x) = hash(x')$. *Preimage resistance*: given y, it is hard to find x such that $hash(x) = y$. *Second-preimage resistance*: given x, it is hard to find $x' \neq x$ such that $hash(x) = hash(x')$.

VII. EXPERIMENTS

Previous works [8], [27] have already shown that LIDOR's approach successfully thwarts several blackhole and greyhole attack variants. Therefore, we focus on two specific variants that have not been addressed so far. In this section, we first describe our experiment and testbed setup, and then evaluate the impact of a replay-supported and a wormhole-supported greyhole attack. Finally, we include simulation-based experiments to demonstrate scalability of LIDOR in a larger node setup and with a variable number of attackers in the network.

A. Testbed and Setup

Our testbed consists of 10 APU-based nodes [34] which are distributed in an office environment. Figure 11 shows the layout. For each of the following experiments, we use Wi-Fi channel 14 to minimize interference with production networks. Before each experiment, we synchronize all nodes to a local NTP server via the nodes' Ethernet interfaces and bound the synchronization error to 0.1 ms resulting in a maximum error in the end-to-end delay measurements of 0.2 ms. In addition, each node filters its neighbors by RSSI with a threshold of -70 to avoid spurious links. We select source and destination nodes to be at a maximum distance such that they are connected via at a path of five hops. In all experiments, the source injects 128-byte packets at a rate of 10 packets per second for an entire flow of 256 packets. We repeat each experiment 100 times. For the *wormhole* scenario, the attacker nodes use their wired Ethernet interface as a direct connection to tunnel traffic between the nodes. We use the *TPy* framework [38] to orchestrate our experiments.

B. Summary

We present a summary of our experiment results comparing the performance of LIDOR to the benchmark in three scenarios: 1) without attackers present, 2) with two attackers mounting a replay-supported greyhole attack, and 3) with

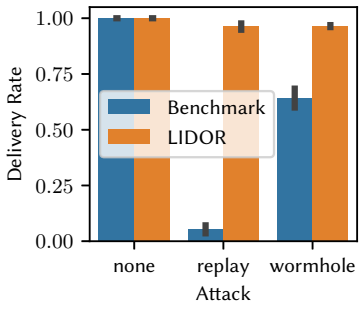


Fig. 5: Packet delivery rate

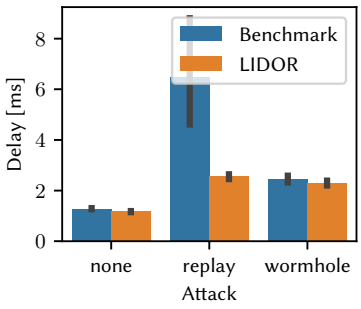


Fig. 6: End-to-end delay

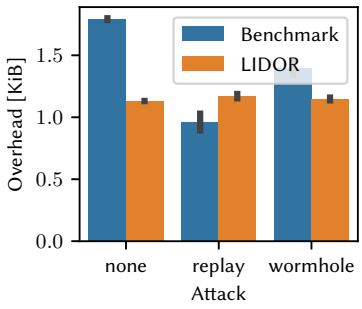


Fig. 7: Per-packet overhead

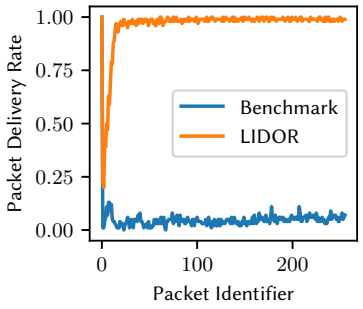


Fig. 8: Packet delivery rate under *replay*-supported greyhole attack.

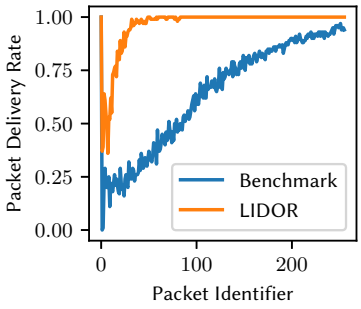


Fig. 9: Packet delivery rate under *wormhole*-supported greyhole attack.

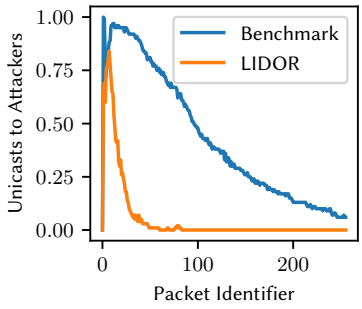


Fig. 10: Attacker selection under *wormhole*-supported greyhole attack.

two attackers mounting a wormhole-supported greyhole attack where the wormhole endpoints are direct neighbors to the source and destination, respectively. We show the packet delivery rate in Fig. 5, the end-to-end delay in Fig. 6, and the overhead Fig. 7. The figures show the mean and standard deviation over the different runs.

In short, we see that LIDOR and the benchmark both achieve perfect delivery rates in the benign case (Fig. 5). When under attack, LIDOR's reliability reduces *only by 3.5 %* for both attacks, while the benchmark breaks down to 5.2 % and 64.1 %, respectively. Thereby, LIDOR achieves improvements over the benchmark of 91 % and 32 %, respectively. In Fig. 6, we see that the end-to-end delay is similar for the benchmark (1.3 ms) and LIDOR (1.2 ms) under no attack which is to be expected since they are both based on the same implementation. Under attack, the end-to-end delay increases. The reason is that the attacker nodes are placed in a favorable position and would allow faster delivery if they would be used as a next hop. Furthermore, Fig. 7 shows the network-wide overhead of a single packet. We see that LIDOR reduces this network overhead by 35 % compared to the benchmark which is in line with our overhead analysis in Section IV. In addition, LIDOR's median overhead does not increase under attack. For the benchmark, the median overhead decreases under attack as the packets are dropped early and do not traverse the entire path from source to destination. In the following sections, we investigate the results for the attack scenarios in more detail.

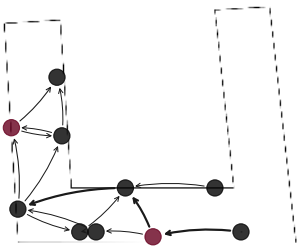
C. Replay-supported Greyhole Attack

In this section, we expose nodes to greyhole attackers that concurrently replay expired PKTs and ACKs to reinforce their appearance as reliable forwarders. We first sketch the attacker's behavior which tries to disrupt communication between s and d. First, the attacker overhears and records any valid PKT–ACK pair of some flow H between s and d. Then, the attacker replays (i. e., broadcasts) PKT and ACK shortly after one another at an interval of T_{rep} (after an initial delay of T_{rep}). The attacker chooses T_{rep} such that it is larger than the ACK timeout, i. e., $T_{\text{rep}} > T_{\text{ACK}}$. To ensure this, the attacker conservatively sets T_{rep} to 200 ms for each pair. We limit the rate of replayed pairs to 10 per second to avoid DoS by flooding.

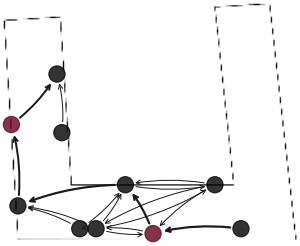
Figure 5 shows the severe impact of lacking replay protection: the benchmark's reliability drops to about 5.2 % which renders the protocol unusable (which is also reflected in the lower overhead of Castor in Fig. 7 as packets are dropped early on the path). On the other hand, LIDOR performs extremely well. There is a small drop in the median reliability which is due to LIDOR having to route around the greyhole attackers. Once the protocol has found a reliable path, it keeps using it. This can be seen in Fig. 9, where only the first few packets of each flow are less likely to be delivered.

D. Wormhole-supported Greyhole Attack

We investigate the impact of a wormhole-supported greyhole attack on both protocols. In Fig. 5, we have already seen that the PDR of LIDOR slightly drops. In fact, only the first

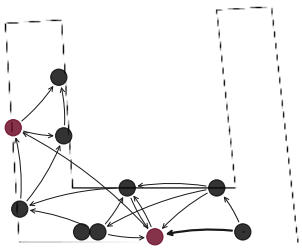


(a) Benchmark, $k \in [0, 256[$

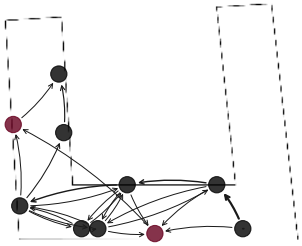


(b) LIDOR, $k \in [0, 256[$

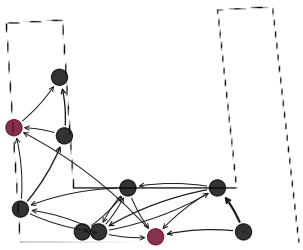
Fig. 11: Path convergence without attack. See Fig. 12 for description.



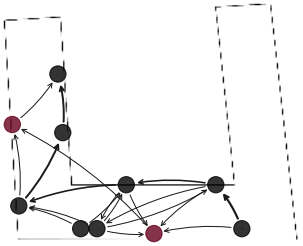
(a) Benchmark, $k \in [0, 32[$



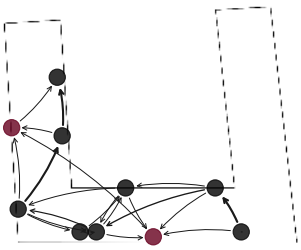
(d) LIDOR, $k \in [0, 32[$



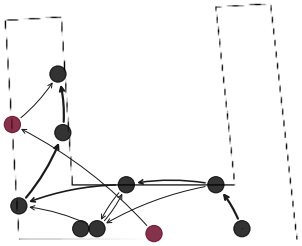
(b) Benchmark, $k \in [32, 128[$



(e) LIDOR, $k \in [32, 128[$



(c) Benchmark, $k \in [128, 256[$



(f) LIDOR, $k \in [128, 256[$

Fig. 12: Path convergence under *wormhole*-supported greyhole attack. Showing unicast transmissions. Flow from bottom right to top left. Red nodes are attackers. Edge thickness idicates link usage frequency.

packets of each flow are dropped as LIDOR first needs to find a valid path, i. e., it needs to *converge*. The delivery rate per packet ID is shown in Fig. 9 where we see that after about 100 packets, the loss rate becomes zero. To verify that this is, in fact, due to the attackers being selected, we depict the relative frequency that an attacker was selected as the sole forwarder (unicast) for a given packet ID in Fig. 10. The figure confirms that attackers are no longer selected as forwarders after about 100 packets. The benchmark protocol does not perform as well. We see that while attacker selection decreases and, thus, PDR increases within a flow (Figs. 9 and 10), the benchmark does not completely reject the wormhole attacker as a viable forwarder. For an even more detailed analysis, we show the network graph including forwarding decisions and the resulting path selection for different portions of a flow in Fig. 12. The figure shows the average over all 100 experiment runs. For comparison, Fig. 11 shows the path selection without an attack. Figures 12a and 12d show that both protocols are "fooled" by the fast link that the wormhole offers for the first packets, i. e., a path including the wormhole has the lowest round-trip time. While the benchmark prefers a non-adversarial path, it still uses the wormhole in a significant number of cases (Fig. 12c). In contrast, LIDOR completely avoids the attackers for packets in the second half of the flow (Fig. 12f).

E. Scalability

In the following, we demonstrate scalability of LIDOR and show the impact of a variable number of attackers on both protocols. We chose a simulator for this purpose to (i) increase the number of nodes, (ii) control hop length via the topology, and (iii) vary the number of attackers. The simulations are performed in ns-3.25 which allows us to integrate our Click integration of the protocols in the simulator environment (Section VI) We use a setup with 102 nodes consisting of a single source–destination pair that is connected via a 10-hop corridor [39], where each hop "layer" consists of 10 nodes. In all experiments, the source injects 128-byte packets at a rate of 2 packets per second for an entire flow of 1024 packets. We repeat each experiment 10 times. We provide results when 0, 10, 20, 30, 40, or 50 of the nodes act as replay-attackers. We exclude the results for the benchmark with more than 20 attackers due to its poor performance.

Figures 13 to 15 show the resulting packet delivery rate, delay, and per-packet overhead. We make several observations. First, we see that the benchmark is unable to provide a reliable service even in the presence of a small number of replay attackers. Second, LIDOR is able to maintain a reliable service even when half of the intermediate nodes (50) act as attackers (see Fig. 13). Third, we observe that the attackers have only a minor effect on the end-to-end delay and overhead of the protocol (see Figs. 14 and 15).

VIII. Discussion

In this section, we elaborate on our analytical and experimental results; draw a conjecture for the applicability in large-scale IoT deployments; discuss the possibility for 100 % reliable communication; and highlight possible other application domains.

A. Convergence: Analysis vs. Experiments

Our analysis in this work shows that LIDOR converges under attack while the benchmark does not. Our experiments confirm that LIDOR converges. However, they do not necessarily show that the benchmark does not converge either. In fact, the benchmark seems to be able to slowly approach a converged state (see Fig. 9). Note that the non-convergent cases for benchmark are statistically rare which explains its overall good performance. However rare, these occurrences can comprise the security of the network.

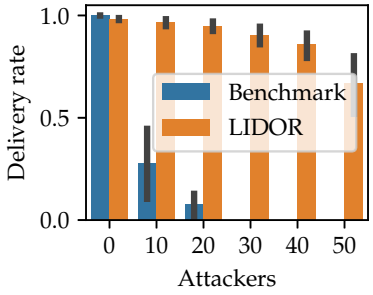


Fig. 13: Packet delivery rate with different numbers of attackers (simulation with 102 nodes)

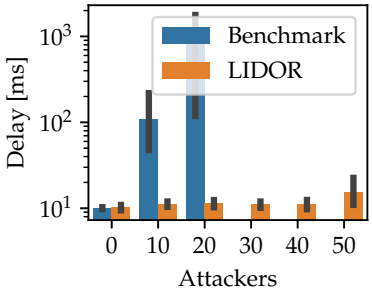


Fig. 14: End-to-end delay with different numbers of attackers (simulation with 102 nodes)

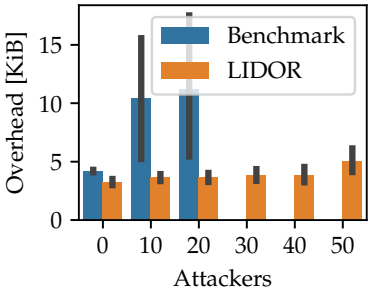


Fig. 15: Per-packet overhead with different numbers of attackers (simulation with 102 nodes)

B. Energy Consumption

Energy consumption is key for the practical deployment of LIDOR in the IoT context. To quantify LIDOR's security overhead, we analyze the number of cryptographic operations (hashing and tagging) involved in transferring an entire flow of size w (tree with $N = 2^l - 1$ nodes) over a converged I-hop path. In particular, we identify the costs for tree generation $E_H = (N+w)E_{hash}$, flow verification $E_f = NE_{hash}$, packet digest calculation $E_m = wE_{hash}$, ACK verification $E_A = wE_{hash}$, and packet tagging or tag verification $E_t = wE_{tag}$. The composite cost N is the sum of the cost for the source $E_s = E_H + E_m + E_A + 2E_t$, destination $E_d = E_H + 2E_t$, and all intermediate nodes $E_i = E_f + E_m + E_A + 2E_t$ on the path. In particular,

$$\begin{aligned} E &= E_s + (I-1)E_i + E_d \\ &= ((I+1)N + (2I+2)w)E_{hash} + (2I+2)wE_{tag} \ . \end{aligned} \tag{25}$$

If the platform- and primitive-dependent values for E_{hash} and E_{tag} are known or can be approximated from the required CPU cycles (e. g., [40]), Eq. (25) allows us to calculate LIDOR's security-related energy overhead.

C. Feasibility for Large-scale IoT Deployments

While LIDOR is not able to exceed the theoretic limits of scalability in wireless multihop networks [41], we attempt our best to keep network overhead as low as possible. In particular, we show that LIDOR's overhead is generally lower than the benchmark which is due to our in-network Merkle tree compression mechanism. In addition, its overhead does not increase under attack which means that attacks (Section II) do not impede scalability. In addition, we show the feasibility of a comprehensively DoS-resilient communication protocol in the IoT context by implementing LIDOR in a computationally efficient manner: in spite of per-packet cryptographic operations, we achieve end-to-end delays in the order of 1 ms in our 5-hop testbed which confirms that the computational overhead is negligible.

D. Towards 100 % Reliability

While LIDOR already performs exceptional under attack, we are aware that we still encounter a certain amount of packet loss. By design, LIDOR tries to be *resilient* to all causes of packet loss but does not employ measures to compensate for loss once it occurred. We are aware that some applications might require a 100 %-reliable transport. We could increase the reliability of a communication by introducing redundancy in form of packet transmissions reactively. Thanks to the end-to-end feedback, the source knows if the destination received a certain packet and could issue a retransmission (using a new packet ID) to the destination. Such a mechanism could be implemented as a LIDOR-aware transport overlay that receives feedback from the network layer and takes care of end-to-end retransmissions.

E. Further Application Domains

While we focus on IoT in this paper, LIDOR's adaptivity to any kind of packet dropping allows for applications in more dynamic scenarios including public safety [27] or highly-dynamic UAV-based networks [42]. However, an experimental evaluation for these types of networks is still missing.

IX. Conclusion

The provisioning of robust and secure communication is crucial for safety-critical IoT applications. In this article, we proposed LIDOR, which is multi-hop communication protocol with an efficient end-to-end acknowledgment-based feedback mechanism tailored to IoT devices. To the best of our knowledge, LIDOR is the first algorithm of its kind with proven convergence in presence of DoS attacks. Convergence is in fact important since "non-convergent" property of a scheme can itself be used to create DoS. We have performed extensive experiments in our premises. These experiments have confirmed the resilience of LIDOR against replay and wormhole attacks. Specifically, LIDOR outperforms the benchmark scheme by 91 % under replay attack and 32 % under wormhole attack in terms of packet delivery ratio and reduces overhead by 35 % in the benign scenario and *does not increase significantly* under attack. The current proof-of-convergence is valid for *known* packet dropping attack variants, i.e., the attacker always drops unicast packets to cause maximum harm. Our experiments indicated that LIDOR converges even in a wormhole-supported greyhole attack. We intend to generalize the current proof to *arbitrary* packet dropping strategies, which would mean that

LIDOR converges to any *future and still unknown* dropping attacks. Finally, for better reproducibility, we make the source code of our implementation publicly available [43].

ACKNOWLEDGMENT

This work has been co-funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. Also, the authors express their sincere gratitude to the Ministry of Electronics and IT, Govt. of India, for their financial support under the Visvesvaraya Ph.D. Scheme for Electronics and IT.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," vol. 29, no. 7, pp. 1645–1660.

[2] Bluetooth SIG, "Mesh model bluetooth specification v1.0." [Online]. Available: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=429634

[3] ——, "Mesh profile bluetooth specification v1.0." [Online]. Available: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=429633

[4] K. Liu, J. Deng, P. K. Varshney, and K. Balakrishnan, "An Acknowledgment-Based Approach for the Detection of Routing Misbehavior in MANETs," vol. 6, no. 5, pp. 536–550.

[5] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens, "ODSBR: An On-Demand Secure Byzantine Resilient Routing Protocol for Wireless Ad Hoc Networks," vol. 10, no. 4, pp. 1–35.

[6] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy, "Routing amid colluding attackers," pp. 184–193.

[7] Y. Xue and K. Nahrstedt, "Providing fault-tolerant Ad hoc routing service in adversarial environments," vol. 29, no. 3-4 SPEC.ISS., pp. 367–388.

[8] W. Galuba, P. Papadimitratos, M. Poturalski, K. Aberer, Z. Despotovic, and W. Kellerer, "Castor: Scalable secure routing for ad hoc networks," in *IEEE INFOCOM*.

[9] M. Katagi and S. Moriai, "Lightweight cryptography for the internet of things."

[10] N. Schweitzer, A. Stulman, R. D. Margalit, and A. Shabtai, "Contradiction based gray-hole attack minimization for ad-hoc networks," vol. 16, no. 8, pp. 2174–2183.

[11] T. Poongodi and M. Karthikeyan, "Localized secure routing architecture against cooperative black hole attack in mobile ad hoc networks," vol. 90, no. 2, pp. 1039–1050.

[12] M. Hollick, C. Nita-Rotaru, P. Papadimitratos, A. Perrig, and S. Schmid, "Toward a Taxonomy and Attacker Model for Secure Routing Protocols," vol. 47, no. 1, pp. 43–48.

[13] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols," in *Proceedings of the 2nd ACM Workshop on Wireless Security*, ser. WiSe, pp. 30–40.

[14] ——, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks," in *Proceedings of the IEEE Conference on Computer Communications*, ser. INFOCOM, vol. 3, pp. 1976–1986.

[15] J. R. Douceur, "The Sybil Attack," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Springer Berlin Heidelberg, vol. 2429, pp. 251–260.

[16] M. G. Zapata and N. Asokan, "Securing ad hoc routing protocols," in *ACM Workshop on Wireless Security*.

[17] P. Papadimitratos and Z. J. Haas, "Secure routing for mobile ad hoc networks," in *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*.

[18] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, "A Secure Routing Protocol for Ad Hoc Networks," in *IEEE ICNP*.

[19] Y.-C. Hu, D. B. Johnson, and A. Perrig, "SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks."

[20] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks," vol. 11, no. 1-2.

[21] P. Papadimitratos and Z. J. Haas, "Secure message transmission in mobile ad hoc networks," vol. 1, no. 1, pp. 193–209.

[22] K. Balakrishnan, Jing Deng, and P. Varshney, "TWOAK: preventing selfishness in mobile ad hoc networks," in *IEEE Wireless Communications and Networking Conference, 2005*, vol. 4, no. C. IEEE, pp. 2137–2142.

[23] R. H. Jhaveri and N. M. Patel, "A sequence number based bait detection scheme to thwart grayhole attack in mobile ad hoc networks," vol. 21, no. 8, pp. 2781–2798.

[24] S. Abbas, M. Merabti, D. Llewellyn-Jones, and K. Kifayat, "Lightweight sybil attack detection in manets," vol. 7, no. 2, pp. 236–248.

[25] M. Fomichev, F. Álvarez, D. Steinmetzer, P. Gardner-Stephen, and M. Hollick, "Survey and systematization of secure device pairing," vol. 20, no. 1.

[26] M. Schmittner and M. Hollick, "Xcastor: Secure and Scalable Group Communication in Ad Hoc Networks," in *IEEE WoWMoM*.

[27] M. Schmittner, A. Asadi, and M. Hollick, "SEMUD: Secure Multi-hop Device-to-Device Communication for 5G Public Safety Networks," in *IFIP Networking Conference and Workshops*. IEEE.

[28] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the Network Time Protocol," in *USENIX NDSS*.

[29] P. Papadimitratos and A. Jovanovic, "GNSS-based positioning: Attacks and countermeasures," in *IEEE MILCOM*.

[30] P. Papadimitratos and Z. J. Haas, "Secure Data Communication in Mobile Ad Hoc Networks," vol. 24, no. 2.

[31] M. Islam and M. Hamid, "SHWMP: A secure hybrid wireless mesh protocol for ieee 802.11s wireless mesh networks."

[32] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," vol. 18, no. 3.

[33] PC Engines. ALIX platform. [Online]. Available: http://www.pcengines.ch/alix.htm

[34] ——. APU platform. [Online]. Available: http://www.pcengines.ch/apu.htm

[35] F. Denis, "The Sodium crypto library." [Online]. Available: https://github.com/jedisct1/libsodium

[36] J. P. Aumasson and D. J. Bernstein, "SipHash: A fast short-input PRF," in *LNCS*, vol. 7668.

[37] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," in *IEEE S&P*.

[38] D. Steinmetzer, M. Stute, and M. Hollick, "TPy: A Lightweight Framework for Agile Distributed Network Experiments," in *12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH '18)*. ACM.

[39] A. Loch, M. Hollick, A. Kuehne, and A. Klein, "Corridor-based routing: Opening doors to phy-layer advances for wireless multihop networks," in *IEEE WoWMoM*.

[40] D. J. Bernstein and T. Lange. eBACS: ECRYPT benchmarking of cryptographic systems. [Online]. Available: https://bench.cr.yp.to

[41] P. Gupta and P. R. Kumar, "The Capacity of Wireless Networks," vol. 46, no. 2, pp. 388–404.

[42] L. Baumgärtner, S. Kohlbrecher, J. Euler, T. Ritter, M. Stute, C. Meurisch, M. Muhlhäuser, M. Hollick, O. von Stryk, and B. Freisleben, "Emergency Communication in Challenged Environments via Unmanned Ground and Aerial Vehicles," in *IEEE Global Humanitarian Technology Conference (GHTC)*.

[43] M. Stute, "Castor and LIDOR source code." [Online]. Available: https://github.com/seemoo-lab/click-castor