

Original Research

Received 30 June 2026

Revised 24 July 2026

Accepted 07 Aug 2026

Natural Resources for Human Health 2026, Vol. 6 (16s): 1173–1189

KEYWORDS:

wireless sensor networks; black-hole attack; selective forwarding; Merkle hash tree; intrusion detection; network security

A Lightweight Merkle Hash Tree-Based Framework for Black-Hole and Selective-Forwarding Attack Identification in Wireless Sensor Networks

M.Ruba¹, S.Kanchana², Anish.T.P³, G Vinoth Chakkaravarthy⁴, Muthumayil K⁵, Karthick P⁶

¹ Department of Computer Science and Engineering, K.Ramakrishnan college of Engineering, Trichy, Tamilnadu

² Department of Computer Science, Faculty of Liberal Arts and Business Studies, SRM Institute of Science and Technology, Kattankulathur.

³ Department of Computer Science and Engineering, R.M.K. College of Engineering and Technology, Puduvoyal, Tamilnadu

⁴ Department of Computer Science and Engineering, Velammal College of Engineering and Technology, Madurai, Tamil Nadu. India.

⁵ Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai, India;

⁶ Vel Tech Multi Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Chennai, India

ABSTRACT: Wireless sensor networks (WSNs) forward data over multiple hops through resource-constrained and often unattended nodes, which makes black-hole and selective-forwarding attacks easy to mount and difficult to distinguish from ordinary packet loss. Existing detectors rely on neighbour overhearing, end-to-end acknowledgements or learned traffic models, and they either spend energy on promiscuous listening, localise attackers poorly, or provide no verifiable evidence. This paper proposes a Merkle hash tree (MHT) framework in which every forwarding node commits, once per epoch, to the sets of packets it received and forwarded using two constant-size authenticated roots (44 bytes in total). The sink audits these commitments with random leaf challenges, verifies the answers through logarithmic-size Merkle proofs against the next hop's committed root, and converts the resulting mismatch counts into node-level decisions with one-sided binomial tests that account for natural channel loss. In a simulation of 50 to 200 node networks with 5% to 25% compromised forwarders, half of which also falsified their own reports, the framework identified 100% of attackers with a false-positive rate of at most 0.18%, needed 3.2 epochs on average, and raised the packet delivery ratio from 44.8% to 82.4% after isolation. Control traffic is the main cost (57.2% of data bytes at the most responsive setting and 7.3% when epochs are lengthened and audits are sampled); this trade-off is quantified.

1. INTRODUCTION

Wireless sensor networks (WSNs) collect measurements from large numbers of small, battery-powered nodes and deliver them over multiple wireless hops to a base station, or sink [1]. Because the nodes are cheap, unattended and often physically accessible, an adversary can capture one, extract its keys and re-program it, after which the node continues to take part in routing while behaving maliciously. Routing is therefore one of the most attractive targets in a WSN, and a long line of work has catalogued the resulting attacks and their consequences for data availability [2,3].

Two of these attacks are especially damaging and especially difficult to detect. In a black-hole attack, a compromised node attracts traffic, typically by advertising a favourable route, and then discards every packet it is asked to forward. In selective forwarding, the compromised node drops only a fraction of the traffic or only packets of a chosen type, so that the network appears to function and the attack stays below the loss level that operators normally tolerate [2]. The difficulty is that a wireless link loses packets for innocent reasons such as fading, collisions and queue overflow, so a missing packet is only weak evidence of misbehaviour. A detector must separate a small, persistent excess of loss at one node from the ambient loss that every node experiences.

Existing defences each carry a cost. Watchdog-style monitoring asks a node to overhear whether its next hop retransmits a packet [4], which keeps the radio in receive mode and depends on overhearing that is itself unreliable. Acknowledgement-based schemes return evidence from the destination or from selected checkpoint nodes [5,6], but they usually localise a fault to a stretch of the path rather than to a single node. Learning-based detectors can recognise abnormal traffic patterns [7], yet they need training data and do not produce evidence that a third party can verify. A further limitation common to many schemes is that a compromised node can lie about what it has done, and nothing in the protocol lets the sink check the claim.

This paper takes a different route. Each forwarding node keeps a compact, cryptographic record of the packets it received and the packets it forwarded in an epoch, in the form of two Merkle hash tree roots [8]. Merkle trees have been used in sensor networks to commit to readings during secure aggregation [9,10], and outside sensor networks to build tamper-evident logs and audit proofs [11,12]. Here we apply them to auditing the forwarding behaviour of individual nodes. Committing to a set of packets with a single hash lets the sink later demand a few randomly chosen entries, check each one against the next hop's commitment and obtain evidence that the node cannot fabricate without breaking the hash function. The remainder of the paper is organised as follows. Section 1.1 reviews related work, Section 2 presents the system model, the framework and its analysis, Section 3 reports and discusses the simulation results, and Section 4 concludes.

1.1 RELATED WORK

Overhearing and local monitoring. The watchdog mechanism of Marti et al. [4] lets a node verify that its next hop forwards a packet by listening to the channel, and the cooperative intrusion-detection architecture of Zhang and Lee [13] extends local monitoring to ad hoc networks; surveys of the field are given in [14]. The original watchdog work itself notes that ambiguous collisions, limited transmission power and collusion can cause missed detections and false accusations, and monitoring requires nodes to receive packets that are not addressed to them. Reputation systems such as the Bayesian framework of Ganeriwal et al. [15] combine direct observations with second-hand reports, which improves robustness but inherits the quality of the underlying observations.

Acknowledgement and checkpoint schemes. Yu and Xiao [5] detect selective forwarding with a multi-hop acknowledgement scheme in which nodes on the path return acknowledgements, so that the loss of a packet can be attributed to a part of the route. CHEMAS [6] reduces the cost of this idea by choosing checkpoint nodes at random and identifying suspect nodes from the acknowledgements they produce. These schemes avoid promiscuous listening but, as the experiments of Section 3 confirm, path-level evidence makes it hard to separate an attacker from honest nodes that lie on the same paths.

Statistical and learning-based detection. Kaplantzis et al. [7] train a support vector machine on traffic features to recognise black-hole and selective-forwarding behaviour. Such models can detect patterns that threshold rules miss, but they need representative training traffic and tuning for each deployment, and their decisions are not accompanied by checkable evidence.

Secure routing and authenticated structures. INSENS [16] makes routing intrusion-tolerant by computing routing tables at the base station and using redundant paths, SPINS and μ TESLA [17,18] provide efficient authenticated communication and broadcast, and statistical en-route filtering [19] drops injected false data inside the network. These mechanisms protect the integrity and authenticity of messages, but none of them lets the sink check whether a node actually forwarded the packets it received. Merkle trees [8] supply the missing building block: SIA and SHIA use them to commit to aggregated data [9,10], history trees give tamper-evident logs [11], and Certificate Transparency uses audit proofs to let clients check that an entry is present in a log [12]. Our framework applies the same commit-then-audit pattern to the forwarding behaviour of sensor nodes.

1.2 Research Contribution

- A commitment protocol for forwarding behaviour: each node sends the sink one authenticated report per epoch whose size (44 bytes) does not depend on how many packets it handled, and answers audit challenges with Merkle proofs whose size grows only logarithmically with the number of packets.
- A two-part detection rule, consisting of a count test and a challenge-proof test, that uses exact one-sided binomial statistics calibrated to the channel loss. The count test exposes nodes that report their drops honestly, and the challenge test exposes nodes that falsify their reports.
- A security and complexity analysis covering forgery resistance, the per-epoch probability of catching a falsified claim, and the computation, storage and communication cost per node.
- A simulation study over 50 to 200 nodes and 5% to 25% compromised forwarders that compares the framework with Watchdog-style monitoring and end-to-end acknowledgement inference, and reports detection rate, false-positive rate, latency, control traffic, energy, memory and post-isolation delivery ratio, including the trade-off between overhead and responsiveness.

1.3 Motivation

- Evidence produced by forwarding-path detectors is rarely verifiable. A detector that can show a third party why a node was flagged is more useful for revoking keys or replacing hardware than one that only reports a score.
- Overhearing consumes energy on every node, and acknowledgement inference localises faults poorly. A scheme that audits a few randomly chosen packets per node can in principle localise attackers at a cost that the operator can adjust.
- Sensor nodes have kilobytes of memory, so the state needed for auditing must be small and its size must be predictable. Hash trees provide this property because a fixed-size root summarises an arbitrarily large set.

2. MATERIALS AND METHODS

2.1 Network and Threat Model

The network consists of N static sensor nodes placed uniformly at random in a square deployment area, and a trusted sink at its centre (Figure 1). Each node forwards data to the sink along a routing tree built by breadth-first search on the hop count, with ties broken in favour of the nearest neighbour. Time is divided into epochs, and in every epoch each node originates M data packets of 64 bytes. A packet is identified by a 4-byte identifier formed from the originator address and a sequence number. Each wireless transmission is received with probability p_i , and an honest node forwards a packet it has received with probability p_b , the remainder being lost to queue overflow or similar local causes. In all experiments $p_i = 0.95$ and $p_b = 0.99$.

A compromised node can behave in two ways. A black-hole node drops every packet it is asked to forward, which corresponds to a drop probability $q = 1$. A selective-forwarding node drops each forwarded packet independently with probability q , drawn uniformly from 0.2 to 0.6 for each such node. Compromised nodes keep sending their own data, because stopping would reveal them immediately. In addition, a compromised node may falsify its report by claiming that it forwarded every packet it received. In the experiments each compromised node falsifies its report with probability 0.5.

The analysis assumes that the sink is trusted and shares a pairwise secret key K_u with every node u ; that the hash function is collision resistant; and that a compromised node cannot forge the message authentication code of another node. It further assumes that reports and audit messages reach the sink reliably, for example through redundant paths or hop-by-hop acknowledgement, and that compromised nodes do not collude. These assumptions are discussed in Section 3.7.

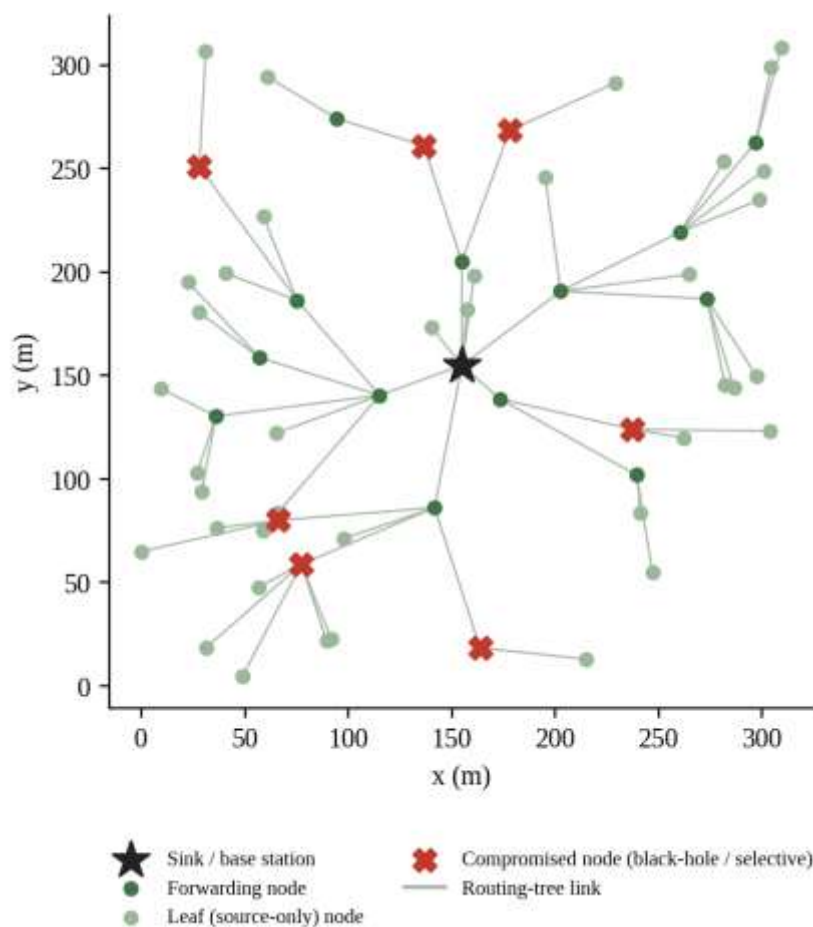


Figure 1. Example deployment of 60 sensor nodes with 7 compromised forwarders and the routing tree.

2.2 Merkle Hash Tree Primer

A Merkle hash tree [8] commits to a list of items with a single hash value. Each item is hashed to form a leaf, and each internal node is the hash of the concatenation of its two children, up to a single root. For a set of n packet identifiers, the leaf for identifier ID is $h = H(0x00 \parallel ID)$ and the internal node over children a and b is $H(0x01 \parallel a \parallel b)$. The prefix bytes provide domain separation, so that a leaf can never be confused with an internal node. The leaves are sorted by identifier before the tree is built. To prove that an identifier belongs to the set, the prover reveals the identifier and the $\lceil \log_2 n \rceil$ sibling hashes on the path from its leaf to the root, as illustrated in Figure 2, and the verifier recomputes the root and compares it with the committed value. Because the leaves are sorted, absence can be proved as well: the prover reveals two adjacent leaves, with their paths, whose identifiers bracket the queried one.

We use SHA-256 [20] truncated to 128 bits as the hash function H , which keeps every hash in a proof to 16 bytes. BLAKE2 [21] is an alternative with a lower software cost, and longer outputs can be used when a larger security margin is needed. Forging a proof for an identifier that is not in the committed set requires finding a second preimage or a collision for H . As a functional check of our implementation, we built 10,000 random trees with 1 to 200 leaves: all 10,000 valid membership proofs were accepted and none of 10,000 forged membership proofs was accepted.

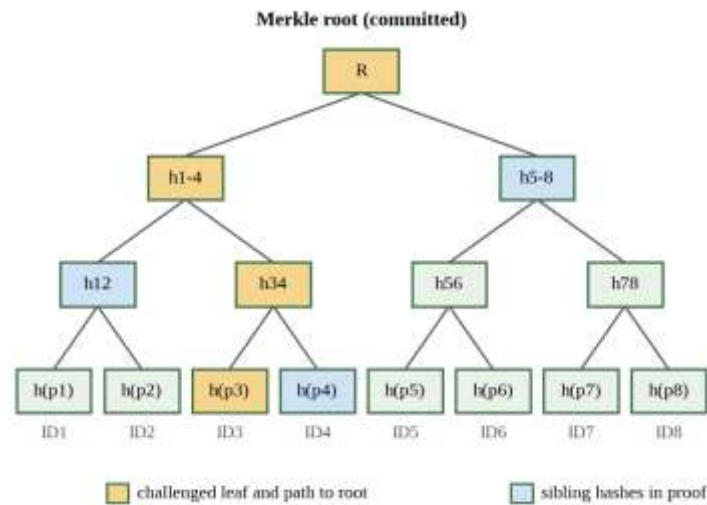


Figure 2. Merkle hash tree over eight sorted packet identifiers and the authentication path for a challenged leaf.

2.3 Framework Overview

Figure 3 shows the workflow of the proposed framework. After deployment, every node shares a key with the sink. During an epoch, each node records the identifiers of the packets it receives from its children and of the packets it forwards to its parent. At the end of the epoch it builds one Merkle tree over each of the two sets and sends the sink an authenticated report that contains the two roots and the two set sizes. The sink then verifies the reports by two tests. The count test compares the number of received and forwarded packets in each report. The challenge-proof test asks the node for a few randomly chosen entries of its forwarded set and asks the next hop to prove, against its own committed root, that it received them. Evidence accumulates across epochs and a one-sided binomial test converts it into a decision. Flagged nodes are excluded and the routing tree is recomputed.

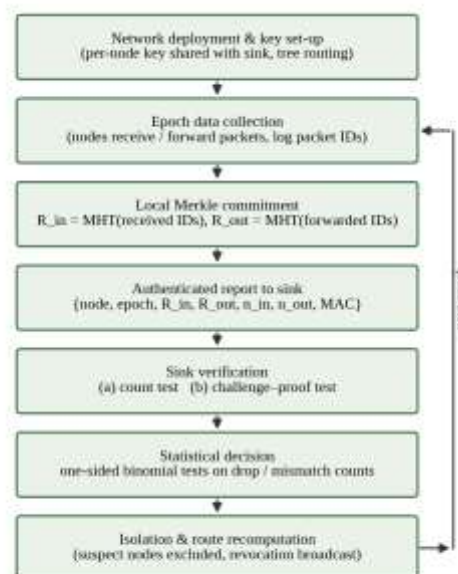


Figure 3. Workflow of the proposed Merkle hash tree-based detection framework.

2.4 Epoch Commitment

Let $L_{in}(u)$ and $L_{out}(u)$ be the sets of identifiers that node u received and forwarded in the current epoch, with sizes $n_{in}(u)$ and $n_{out}(u)$. The node computes

$$R_{in}(u) = \text{MHT}(\text{sort}(L_{in}(u))), \quad R_{out}(u) = \text{MHT}(\text{sort}(L_{out}(u))) \quad (1)$$

and reports

$$\text{Report}(u) = \langle R_{in}(u), R_{out}(u), n_{in}(u), n_{out}(u), \text{MAC}_{Ku}(u \parallel \text{epoch} \parallel R_{in} \parallel R_{out} \parallel n_{in} \parallel n_{out}) \rangle \quad (2)$$

where MAC is an 8-byte truncated HMAC [22] under the pairwise key. The link-layer header already carries the node address and the epoch counter, so the payload is $16 + 16 + 2 + 2 + 8 = 44$ bytes regardless of n_{in} and n_{out} (Table 1). Because the epoch counter is covered by the MAC, a captured report cannot be replayed in a later epoch. The node keeps the sorted identifier lists, and optionally the trees, until the sink's verification window for that epoch has closed. Algorithm 1 summarises the procedure.

Algorithm 1 – Epoch Commitment at Node u

1. $L_{in} \leftarrow \emptyset; L_{out} \leftarrow \emptyset$ // start of epoch
2. For each packet p received from a child {
 3. Append $ID(p)$ to L_{in} ;
 4. If p is forwarded to the parent {
 5. Append $ID(p)$ to L_{out} ;
 6. }
7. }
8. At the end of the epoch, sort L_{in} and L_{out} ;
9. Build MHT over $L_{in} \rightarrow R_{in}$; build MHT over $L_{out} \rightarrow R_{out}$;
10. Report $\leftarrow \langle R_{in}, R_{out}, n_{in}, n_{out}, \text{MAC} \rangle$; send Report to the sink;
11. Retain L_{in}, L_{out} (and the trees) until the audit window closes;
12. End process;

Table 1. Message formats and sizes (identifier 4 B, hash 16 B, MAC 8 B).

Message	Contents	Size (bytes)
Commitment report (node $\rightarrow$ sink)	$R_{in}, R_{out}, n_{in}, n_{out}, \text{MAC}$	$16 + 16 + 2 + 2 + 8 = 44$
Challenge (sink $\rightarrow$ node u)	k leaf indices in $R_{out}(u)$, MAC	$2k + 8$
Response (node $u \rightarrow$ sink)	k identifiers with proofs against $R_{out}(u)$, MAC	$k(4 + 16\lceil \log_2 n_{out} \rceil) + 8$
Query (sink $\rightarrow$ next hop p)	the k identifiers, MAC	$4k + 8$
Proof (next hop $p \rightarrow$ sink)	membership proof (or absence proof, about twice the size) per identifier, MAC	$k(4 + 16\lceil \log_2 n_{in}(p) \rceil) + 8$ per proof type

2.5 Challenge–Response Verification

The sink cannot choose identifiers to challenge, because it knows only the roots. It therefore picks k random leaf positions in the forwarded set of node u , with $k = 4$ in the default configuration, and asks u for the identifiers at those positions together with their proofs against $R_{out}(u)$ (Figure 4). Node u cannot return an identifier that is not in the tree it committed to, so the

sample is a genuine sample of what u claims to have forwarded. The sink then asks the next hop $p = \text{parent}(u)$ to prove, against $R_{in}(p)$, that each of those identifiers was received. Next hop p can answer with a membership proof if it received the packet, or with an absence proof if it did not, and it cannot give a valid proof of the wrong kind. When p is the sink itself, the check needs no network traffic.

The outcome of each sampled identifier is therefore either present or absent. An absent identifier means that u claimed to have forwarded a packet that p never received. Under honest behaviour this happens only when the transmission was lost on the link, which occurs with probability $1 - p_l$. A node that falsifies its report to hide drops produces absent identifiers at the higher rate

$$\delta_f = 1 - (1 - q) \cdot p_b \cdot p_l \quad (3)$$

which is 1 for a black hole and 0.25 for a selective forwarder with $q = 0.2$ in our setting, compared with 0.05 for an honest node. A forger that is audited with k samples in an epoch is therefore exposed in that epoch with probability $1 - (1 - \delta_f)^k$, ignoring the link-loss baseline, and repeated audits accumulate evidence. A node that reports its drops honestly is not exposed by the challenge test, because it never claims to have forwarded what it dropped, but the count test catches it.

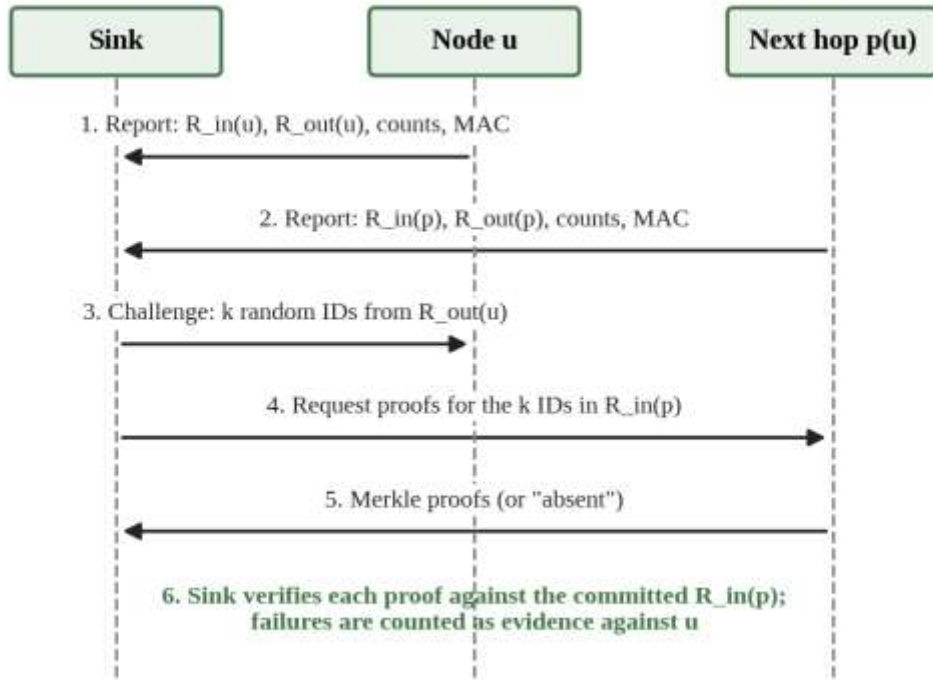


Figure 4. Challenge–response exchange between the sink, the audited node and its next hop.

Algorithm 2 – Sink-Side Verification and Decision

- For each epoch and each node u with a valid report (MAC and epoch counter verified) {
- $N_u \leftarrow N_u + n_{in}(u)$; $D_u \leftarrow D_u + (n_{in}(u) - n_{out}(u))$; // count evidence
- With probability p_{audit} {
- Draw k random leaf positions in the forwarded set of u ;
- Obtain the identifiers and proofs from u ; verify them against $R_{out}(u)$;
- Ask $p = \text{parent}(u)$ for membership/absence proofs against $R_{in}(p)$;
- $S_u \leftarrow S_u + k$; $F_u \leftarrow F_u + (\text{number of absence proofs})$;

8. }

9. If $N_u \geq N_{min}$ and $P[\text{Bin}(N_u, 1 - p_b) \geq D_u] < \alpha \rightarrow \text{flag } u; \quad // \text{ count test}$

10. If $S_u \geq S_{min}$ and $P[\text{Bin}(S_u, 1 - p_l) \geq F_u] < \alpha \rightarrow \text{flag } u; \quad // \text{ challenge test}$

11. }

12. Revoke flagged nodes, recompute the routing tree, and continue;

13. End process;

2.6 Detection Rules and Isolation

Under honest behaviour, the number of packets that a node receives but does not forward, D_u , follows a binomial distribution with N_u trials and success probability $1 - p_b$, and the number of absent proofs, F_u , follows a binomial distribution with S_u trials and probability $1 - p_l$. The sink flags a node when the observed counts are significantly larger than these baselines, namely when

$$P[\text{Bin}(N_u, 1 - p_b) \geq D_u] < \alpha \quad \text{or} \quad P[\text{Bin}(S_u, 1 - p_l) \geq F_u] < \alpha \quad (4)$$

with $\alpha = 10^{-4}$, a minimum of $N_{min} = 10$ observed packets for the count test and $S_{min} = 12$ audited samples for the challenge test. Using an exact tail probability rather than a fixed loss threshold makes the decision adapt automatically to the number of observations: a node with little traffic needs stronger evidence, and a busy node is judged on a larger sample. The loss baselines p_l and p_b are assumed known here; in a deployment they would be estimated from link statistics gathered before any attack. Once a node is flagged, the sink revokes it by an authenticated broadcast, for example one protected with μTESLA [18], and recomputes the routing tree without it.

2.7 Complexity Analysis

For an epoch in which a node handles n packets, building one tree costs n leaf hashes and $n - 1$ internal hashes. The full tree occupies about $(2n - 1) \times 16$ bytes; storing only the sorted identifiers occupies $4n$ bytes, in which case a proof is produced by rehashing the needed part of the list, at a cost of $O(n)$ hashes per challenge. The report is 44 bytes whatever n is, and a proof contains $\lceil \log_2 n \rceil$ hashes, so a 200-leaf proof is 128 bytes plus the identifier. The sink verifies each proof with $\lceil \log_2 n \rceil$ hash evaluations. These costs are what makes the scheme suitable for constrained nodes: computation is linear in the traffic that the node already handles, state is linear in the same quantity with a small constant, and the only network cost that does not depend on the audit rate is the constant-size report.

2.8 Simulation Setup

The framework was evaluated with a custom epoch-level simulator written in Python. The simulator generates the traffic of each epoch packet by packet, applies the per-hop loss and the behaviour of the compromised nodes, records the identifiers handled by each node and feeds the same traffic to all detectors. It does not model the medium-access layer, so collisions, queueing delays and energy depletion appear only through the loss probabilities p_l and p_b . The parameters are listed in Table 2. Compromised nodes are drawn at random from the nodes that forward traffic for at least one child, since a node without children has nothing to drop. Half of them are black holes and half are selective forwarders. Each point is the mean of 40 independent random topologies, and error bars are 95% confidence intervals.

The proposed framework is compared with two baselines. The Watchdog baseline [4] lets each node observe whether its next hop retransmits packets that it sent; an observation succeeds with probability 0.85 when the next hop does forward, to account for imperfect overhearing, and a node is flagged by a one-sided binomial test on the observed success rate against the honest baseline. The E2E-ACK baseline uses only source-to-sink delivery statistics, as in acknowledgement-based inference [5,6]: a source is declared lossy when its delivery ratio is significantly below the honest value for its hop count, and a node is flagged when at least 80% of the sources in its subtree are lossy and its parent does not itself meet this condition. All three detectors use the same significance level $\alpha = 10^{-4}$ and are given the true honest baselines, which favours the baselines because it gives the Watchdog perfect knowledge of the overhearing probability. In attack-free runs, the false-positive rate was 0.18% for the proposed framework, 0.07% for Watchdog and 0.00% for E2E-ACK.

Detection rate is the fraction of compromised forwarding nodes flagged within 40 epochs, the false-positive rate is the fraction of honest forwarding nodes flagged, and latency is the number of epochs until a compromised node is flagged. Control traffic is the number of control bytes transmitted hop by hop, as a percentage of the data bytes transmitted hop by hop. Energy follows the first-order radio model [23] with 50 nJ/bit for the transceiver electronics and 100 pJ/bit/m² for the amplifier. The packet delivery ratio is measured over the honest sources.

Table 2. Simulation parameters.

Parameter	Value
Deployment area	square, side = $40\sqrt{N}$ m (400×400 m for $N = 100$)
Number of nodes (N)	50, 100 (default), 150, 200; sink at the centre
Radio range	80 m
Routing	minimum-hop tree, nearest parent among minimum-hop candidates
Data packet size / packets per source per epoch (M)	64 bytes / 3 (default); 12 and 24 in the sensitivity study
Epochs per run / independent runs	40 / 40
Per-hop delivery probability p_l / honest forwarding probability p_b	0.95 / 0.99
Compromised forwarding nodes	5–25% of $N - 1$ (default 15%); half black-hole ($q = 1$), half selective ($q \sim U(0.2, 0.6)$)
Probability that a compromised node falsifies its report	0.5
Hash / MAC / identifier	SHA-256 truncated to 128 bits / 8-byte HMAC / 4 bytes
Challenge samples per audit (k) / audit probability (p_{audit})	4 / 1 (default)
Significance level α / minimum samples ($N_{\text{min}}, S_{\text{min}}$)	10^{-4} / 10, 12
Watchdog overhearing probability	0.85
Radio energy model	$E_{\text{elec}} = 50$ nJ/bit, $\epsilon_{\text{amp}} = 100$ pJ/bit/m ²

3. RESULTS AND DISCUSSION

3.1 Detection Accuracy

Figure 5 shows the detection rate as the proportion of compromised forwarders grows from 5% to 25% of the network. The proposed framework identified every compromised node in every setting (100%), and it did so for both attack types: at 15% compromise it flagged all black-hole nodes and all selective-forwarding nodes, including those that falsified their reports. The Watchdog baseline reached 99.5% to 99.8% under the idealised overhearing used here, so the two curves overlap in the figure. The E2E-ACK baseline detected 78.0% of the attackers at 5% compromise but only 40.7% at 25%, because attackers close to the sink make every source behind them look lossy, which hides the compromised nodes further upstream. At 15% compromise, E2E-ACK detected 62.8% of the black-hole nodes and 59.3% of the selective forwarders.

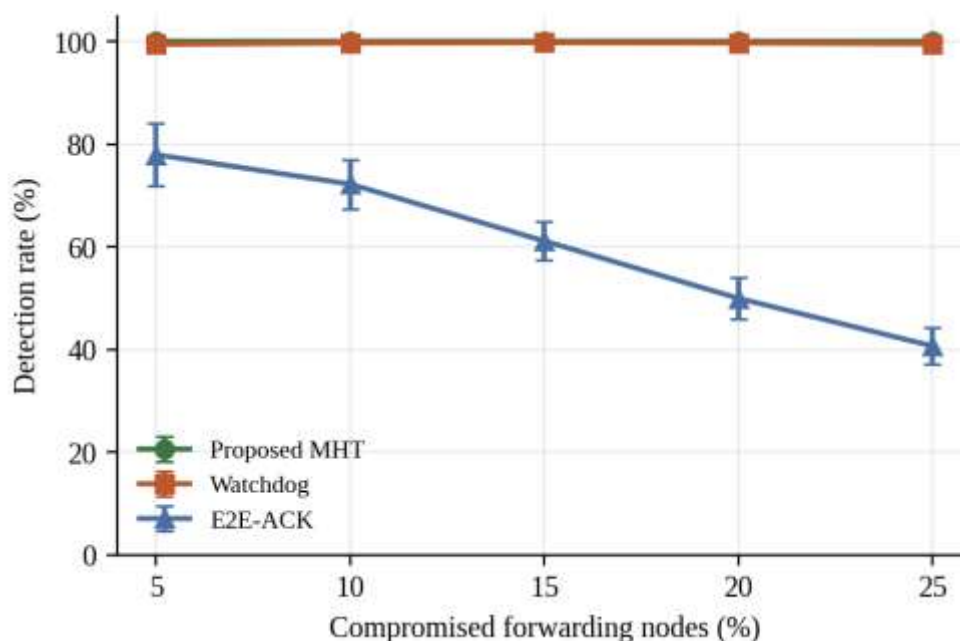


Figure 5. Compromised forwarding nodes vs detection rate (N = 100; error bars show 95% confidence intervals).

Figure 6 shows the same comparison as the network grows from 50 to 200 nodes with 15% compromised. The proposed framework and Watchdog both remain at or above 99.8%, whereas the detection rate of E2E-ACK falls from 80.4% to 48.3%, since longer paths give more honest nodes the same path-level evidence as an attacker.

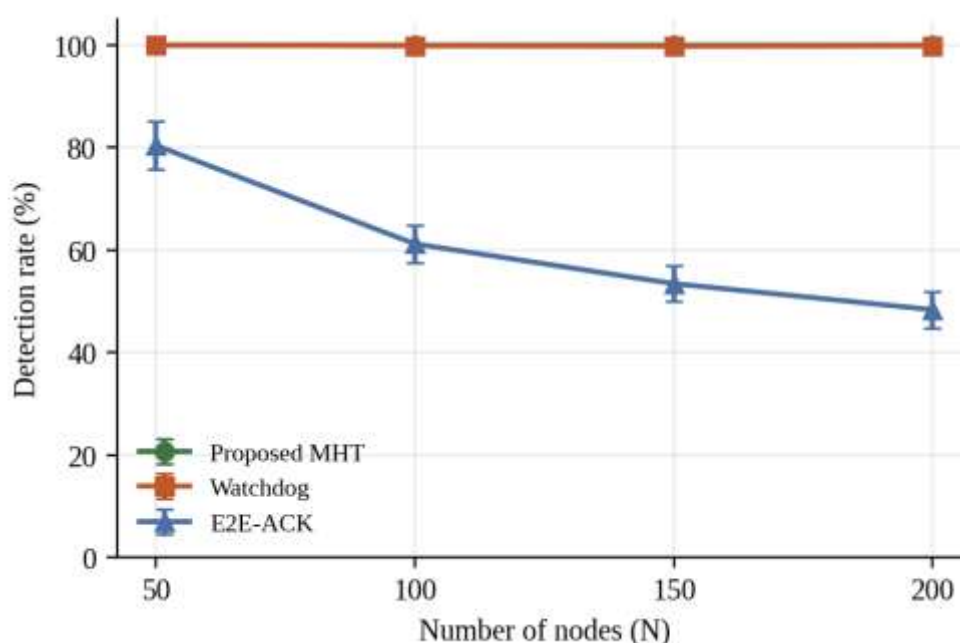


Figure 6. Number of nodes vs detection rate (15% compromised nodes).

3.2 False Positives

Figure 7 reports the share of honest forwarding nodes that were flagged. The proposed framework flagged at most 0.18% of them and Watchdog at most 0.47%, in line with the target set by α . E2E-ACK flagged 3.6% of honest forwarders at 5%

compromise and 11.4% at 25%. These errors arise from the structure of path-level inference: when a node drops traffic, every honest node upstream of it on the same paths shares its symptoms, and the baseline cannot always tell which of them is responsible. The count and challenge tests avoid this ambiguity because each is evaluated per node, against the node's own commitment and its next hop's.

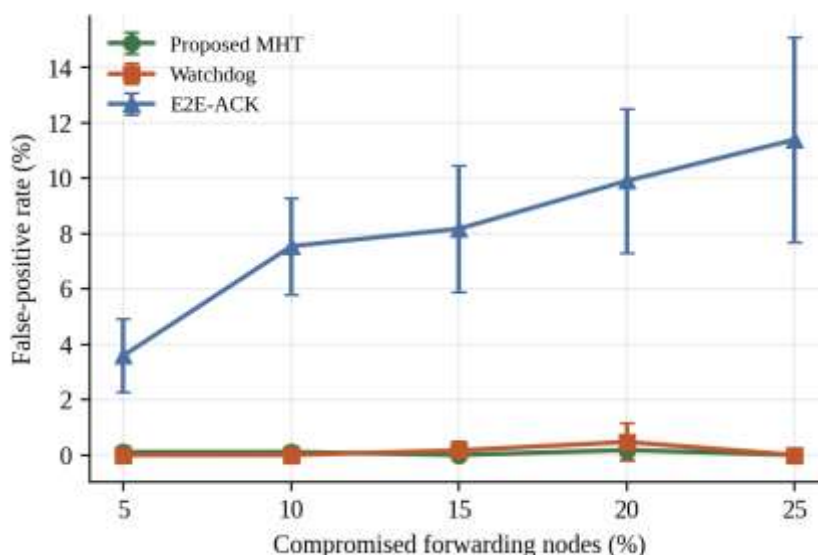


Figure 7. Compromised forwarding nodes vs false-positive rate (N = 100).

3.3 Detection Latency

Figure 8 shows the number of epochs needed to flag a compromised node. The proposed framework needed 3.1 to 3.4 epochs, Watchdog 4.5 to 5.4 epochs and E2E-ACK 6.8 to 8.7 epochs. The latency of the proposed framework did not change with network size, remaining between 3.23 and 3.28 epochs for 50 to 200 nodes. In this default configuration an epoch contains $M = 3$ packets per source, so 3.2 epochs correspond to about 9.7 packet intervals, against 15.5 for Watchdog. This advantage depends on the short epoch and on auditing every node in every epoch, and Section 3.5 shows how it changes when the audit load is reduced.

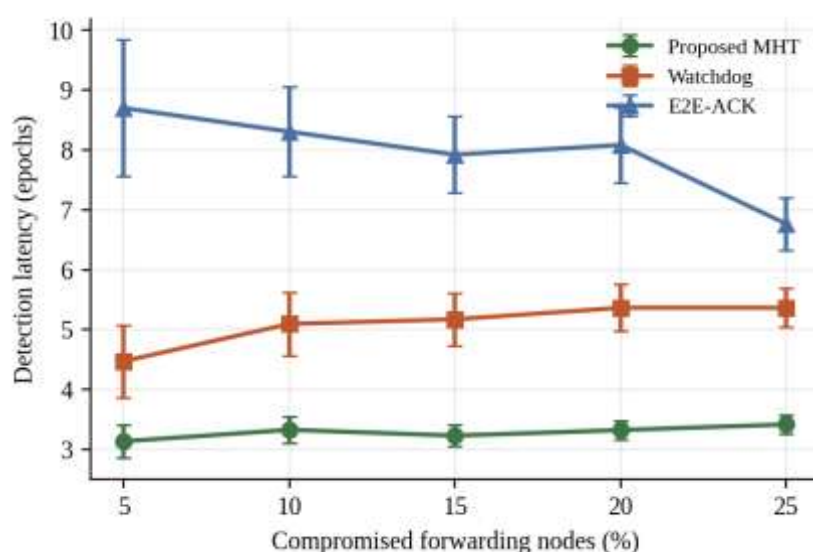


Figure 8. Compromised forwarding nodes vs detection latency (N = 100).

Table 3. Detection performance at $N = 100$ with 15% compromised forwarders (mean of 40 runs).

Metric	Watchdog	E2E-ACK	Proposed MHT
Detection rate (%)	99.8	61.2	100.0
Black-hole nodes detected (%)	100.0	62.8	100.0
Selective forwarders detected (%)	99.6	59.3	100.0
False-positive rate (%)	0.17	8.17	0.00
Detection latency (epochs)	5.17	7.92	3.23

3.4 Overhead and Resource Use

Figure 9 shows the control traffic of the three schemes at the default configuration ($M = 3$, every node audited every epoch, $k = 4$). The proposed framework generates 57.2% of the data bytes at $N = 100$, rising from 32.7% at $N = 50$ to 83.4% at $N = 200$. E2E-ACK needs 8.4% to 10.8%, and Watchdog sends almost no control traffic (0.14%). This is the main cost of the framework and it is considerably higher than that of the baselines. It has three causes: every node sends a commitment each epoch, the commitment is 44 bytes while an epoch carries only 3×64 bytes of data per source, and the audit exchanges of Table 1 traverse the same number of hops as data does. The cost grows with N because the average path lengthens from 2.1 to 3.9 hops, and every report and audit message travels that far.

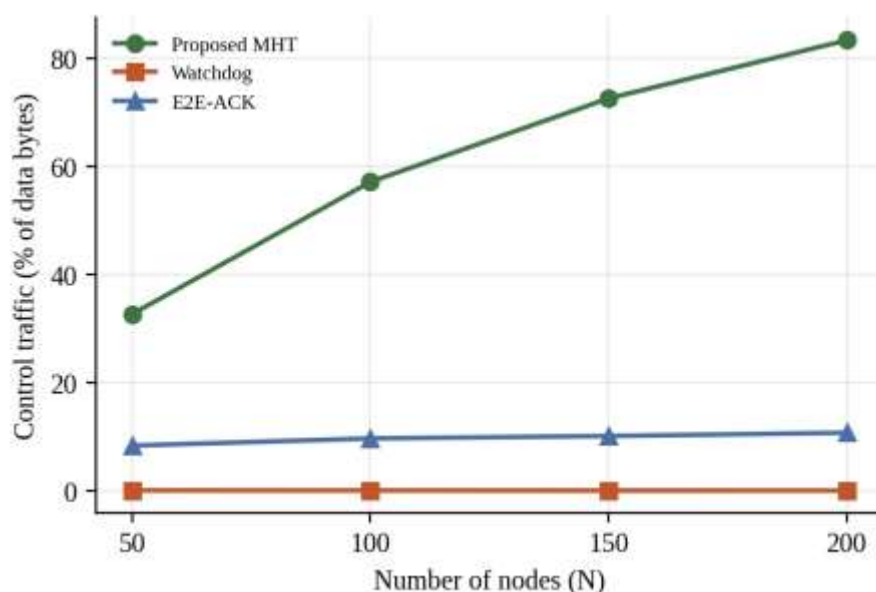


Figure 9. Number of nodes vs control traffic (default configuration).

Figure 10 converts control traffic and overhearing into radio energy. At $N = 100$ the extra radio energy is 57.2% of the data energy for the proposed framework, 6.5% for Watchdog, where the cost comes from receiving every overheard retransmission, and 9.7% for E2E-ACK. Hashing adds computation, and its energy depends strongly on the platform. The framework performs about 49,819 hash evaluations per network over 40 epochs (12.6 per node per epoch). Assuming $2 \mu\text{J}$ per hash (hardware-assisted) the extra energy is 2.5% of the data energy, assuming $20 \mu\text{J}$ it is 24.8%, and assuming $100 \mu\text{J}$ (a slow software implementation) it is 124.1%. These per-hash costs are assumptions and not measurements, and the choice of hash function and hardware should be validated on the target platform.

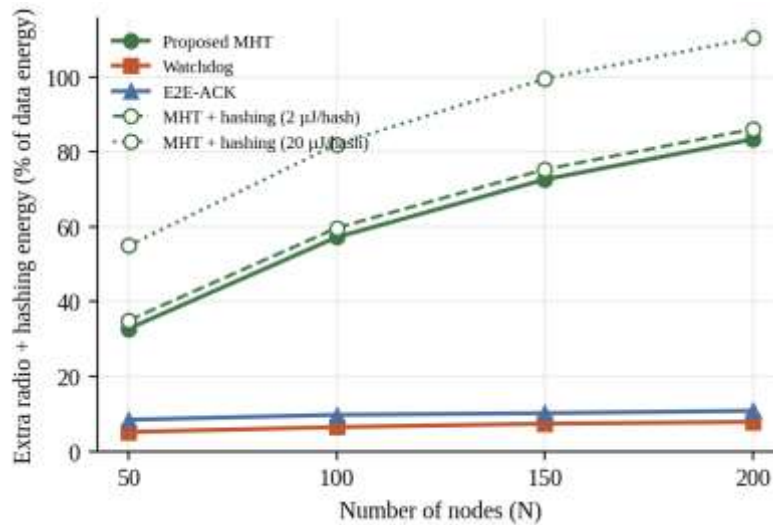


Figure 10. Number of nodes vs extra energy overhead; dashed and dotted lines add assumed hashing energy of 2 μJ and 20 μJ per hash.

Memory use is modest. When each node stores both full trees for the current epoch, the mean is 0.71 KB and the largest value observed is 4.7 KB at $N = 100$, reaching 8.0 KB at $N = 200$ for the nodes closest to the sink. Storing only the sorted identifiers reduces the mean to 93 bytes and the maximum to 0.6 KB at $N = 100$, at the price of rehashing during audits. Table 4 summarises the resource figures.

Table 4. Resource summary at $N = 100$ (default configuration).

Metric	Watchdog	E2E-ACK	Proposed MHT
Control traffic (% of data bytes)	0.14	9.75	57.17
Extra radio energy (% of data energy)	6.5	9.7	57.2
Hash evaluations per run (40 epochs)	–	–	49,819
Peak per-node state, full trees (KB)	–	–	4.7
Peak per-node state, identifiers only (KB)	–	–	0.6
Commitment report size (bytes)	–	–	44

3.5 Trade-off between Overhead and Responsiveness

The default configuration favours fast detection. Two parameters reduce the control traffic: lengthening the epoch, so that the constant-size commitment is amortised over more packets, and auditing each node with probability $p_{\text{audit}} < 1$ in each epoch. Table 5 and Figure 11 show five configurations at $N = 100$ with 15% compromised, with latency expressed in packet intervals so that different epoch lengths can be compared. Lengthening the epoch to $M = 12$ reduces control traffic from 57.2% to 21.9% without losing any detection, but latency increases from 9.7 to 29.2 intervals because reports arrive less often. Auditing a quarter of the nodes in each epoch (configuration C) brings the overhead down to 7.3%, comparable to the energy cost of Watchdog overhearing, but detection falls to 97.7% and latency rises to 86.4 intervals, about four times that of Watchdog (21.2). With only two samples per audit (configurations D and E) the detection rate falls to 92.5% and 91.5%, because a forging node needs more epochs to accumulate the minimum number of audited samples within the 40-epoch window. Varying the number of samples in the default configuration shows the same pattern: $k = 1, 2, 4$ and 8 give 24.4%, 36.5%, 57.2% and 86.5% control traffic with latencies of 7.7, 4.8, 3.2 and 2.8 epochs.

Table 5. Overhead and responsiveness of the proposed framework for different epoch lengths (M), audit probabilities (p_{audit}) and sample sizes (k) at $N = 100$, 15% compromised.

Config	M	p_{audit}	k	Detection (%)	FP (%)	MHT latency (intervals)	Watchdog latency (intervals)	Control traffic (%)
A	3	1	4	100.0	0.00	9.7	15.5	57.2
B	12	1	4	100.0	0.00	29.2	21.1	21.9
C	12	0.25	4	97.7	0.14	86.4	21.2	7.3
D	12	0.25	2	92.5	0.00	132.3	21.1	5.1
E	24	0.25	2	91.5	0.00	264.8	29.6	2.8

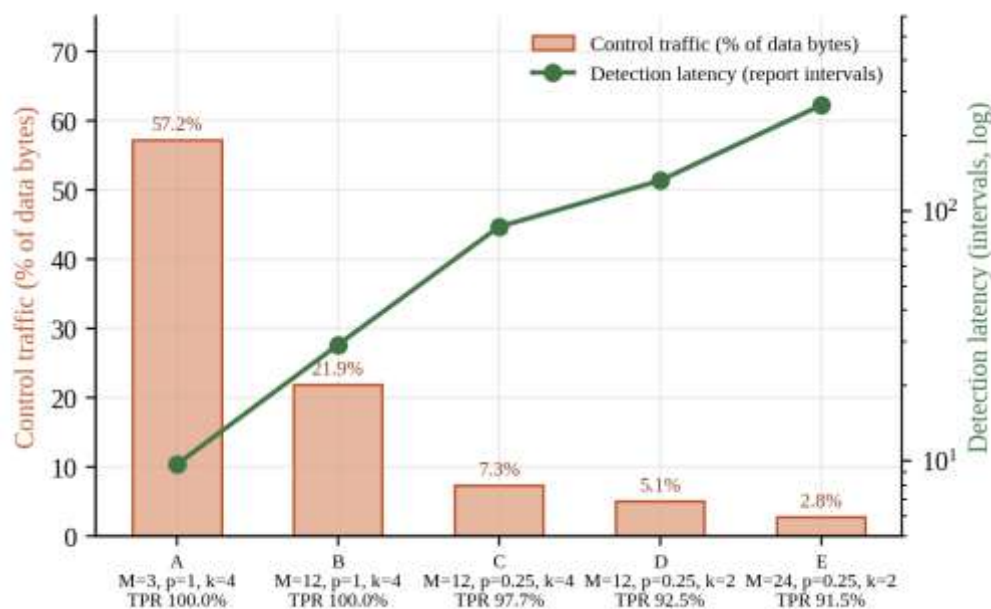


Figure 11. Control traffic and detection latency for configurations A–E of Table 5.

3.6 Network Performance after Isolation

Figure 12 shows the packet delivery ratio of the honest sources when flagged nodes are excluded and the routing tree is rebuilt ($N = 100$, 15% compromised). Without a defence the delivery ratio stays at 44.8%. With the proposed framework it rises to 82.4% (mean of the last ten epochs), compared with 82.2% for Watchdog and 70.9% for E2E-ACK, whose missed detections and false positives leave part of the network cut off or still exposed. The proposed framework and Watchdog both approach the delivery ratio of an attack-free network, which is about 84% for the per-hop success probability of 0.94 and the average path length of this topology, and they recover within about five epochs.

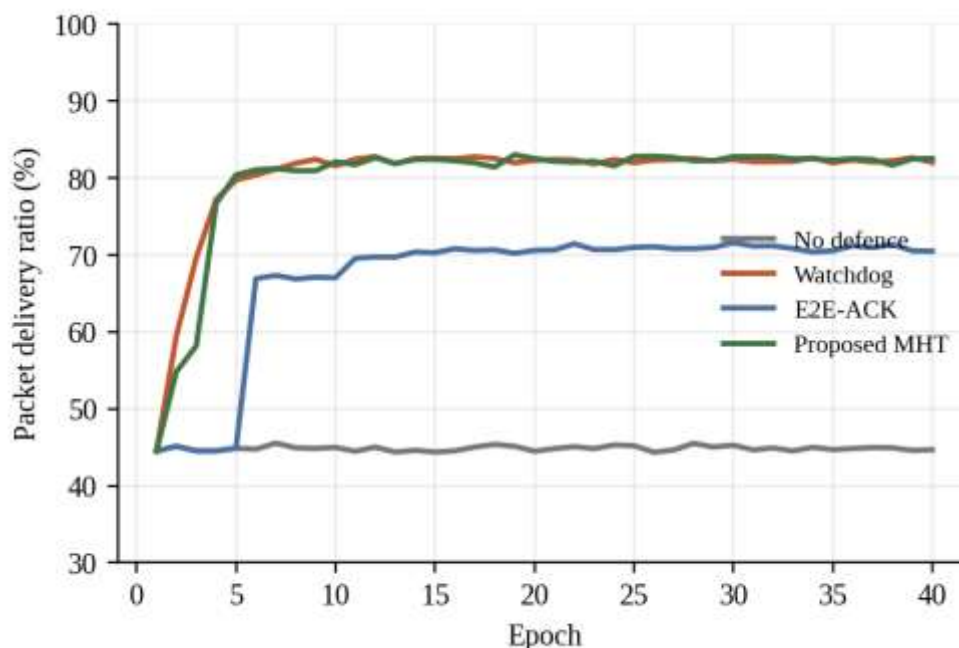


Figure 12. Epoch vs packet delivery ratio after isolating flagged nodes ($N = 100$, 15% compromised).

3.7 Discussion

The results support three conclusions. First, committing to forwarded and received packet sets with Merkle trees gives a detector that is accurate and precise in the simulated setting: all compromised nodes were found, including those that falsified their reports, with a false-positive rate below 0.2%. Second, in this idealised setting the accuracy is not better than that of Watchdog monitoring, which also found nearly all attackers. The benefit of the proposed framework is therefore not a higher detection rate in simulation but different properties: its evidence is cryptographically checkable, it does not depend on overhearing, and a node cannot hide drops by misreporting. These properties are by construction and are not quantified by the simulation, which, for instance, does not model false accusations by compromised watchdogs. Third, the cost is mostly communication. At the most responsive setting the control traffic is large, and it can be reduced to a level comparable with Watchdog's overhearing energy only by accepting a slower detection, as Table 5 shows.

The study has several limitations. The simulator is epoch-level and does not model medium-access effects, congestion or energy depletion; the loss baselines are assumed known; the topology is static with a single sink; and the attacks are limited to dropping, with no delaying, modification, wormholes or Sybil identities. The framework assumes reliable delivery of reports and audit messages, although a black-hole node on the path could drop them, and it does not address collusion, such as a next hop that falsely denies receiving a packet in order to frame its child. In that case the sink would see an inconsistency on a link but could not tell which end is at fault, and cross-checking against additional neighbours would be needed. The hash energy figures are assumptions. Validation in a packet-level simulator or on a testbed is the necessary next step before any deployment claim.

4. CONCLUSION

This paper presented a lightweight Merkle hash tree framework for identifying black-hole and selective-forwarding nodes in wireless sensor networks. Each node commits to the packets it received and forwarded with two roots in a 44-byte authenticated report, the sink audits the commitments with random leaf challenges verified by logarithmic-size Merkle proofs, and one-sided binomial tests turn the evidence into per-node decisions. In simulations with 50 to 200 nodes and up to 25% compromised forwarders, the framework detected all attackers, including those that falsified their reports, with false positives below 0.2%, in 3.2 epochs on average, and restored the delivery ratio from 44.8% to 82.4%. Per-node computation and storage are small, but control traffic is substantial at the most responsive setting (57.2% of data bytes) and can be lowered to 7.3% by lengthening epochs and sampling audits, at the price of slower and slightly less complete detection. Future work will use adaptive audits that

concentrate on subtrees with a delivery deficit, aggregate commitments along the routing tree, add cross-checking to resist colluding neighbours and framing, and validate the design in a packet-level simulator and on sensor hardware.

DATA AVAILABILITY STATEMENT

All simulation parameters are listed in Table 2. The Python source code of the simulator and the raw result files used to produce the figures and tables are provided as supplementary material.

REFERENCES

- [1] Akyildiz, I.F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. Wireless sensor networks: A survey. *Comput. Netw.* 2002, 38, 393–422.
- [2] Karlof, C.; Wagner, D. Secure routing in wireless sensor networks: Attacks and countermeasures. *Ad Hoc Netw.* 2003, 1, 293–315.
- [3] Wood, A.D.; Stankovic, J.A. Denial of service in sensor networks. *Computer* 2002, 35, 54–62.
- [4] Marti, S.; Giuli, T.J.; Lai, K.; Baker, M. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, Boston, MA, USA, 6–11 August 2000; pp. 255–265.
- [5] Yu, B.; Xiao, B. Detecting selective forwarding attacks in wireless sensor networks. In *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Rhodes Island, Greece, 25–29 April 2006.
- [6] Xiao, B.; Yu, B.; Gao, C. CHEMAS: Identify suspect nodes in selective forwarding attacks. *J. Parallel Distrib. Comput.* 2007, 67, 1218–1230.
- [7] Kaplantzis, S.; Shilton, A.; Mani, N.; Sekercioglu, Y.A. Detecting selective forwarding attacks in wireless sensor networks using support vector machines. In *Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP)*, Melbourne, Australia, 3–6 December 2007; pp. 335–340.
- [8] Merkle, R.C. A certified digital signature. In *Advances in Cryptology—CRYPTO ’89; Lecture Notes in Computer Science*, Vol. 435; Springer: New York, NY, USA, 1990; pp. 218–238.
- [9] Przydatek, B.; Song, D.; Perrig, A. SIA: Secure information aggregation in sensor networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, USA, 5–7 November 2003; pp. 255–265.
- [10] Chan, H.; Perrig, A.; Song, D. Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, USA, 30 October–3 November 2006; pp. 278–287.
- [11] Crosby, S.A.; Wallach, D.S. Efficient data structures for tamper-evident logging. In *Proceedings of the 18th USENIX Security Symposium*, Montreal, QC, Canada, 10–14 August 2009; pp. 317–334.
- [12] Laurie, B.; Langley, A.; Kasper, E. Certificate Transparency; RFC 6962; Internet Engineering Task Force: Fremont, CA, USA, 2013.
- [13] Zhang, Y.; Lee, W. Intrusion detection in wireless ad-hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, Boston, MA, USA, 6–11 August 2000; pp. 275–283.
- [14] Sun, B.; Osborne, L.; Xiao, Y.; Guizani, S. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. *IEEE Wirel. Commun.* 2007, 14, 56–63.
- [15] Ganeriwal, S.; Balzano, L.K.; Srivastava, M.B. Reputation-based framework for high integrity sensor networks. *ACM Trans. Sens. Netw.* 2008, 4, 15.
- [16] Deng, J.; Han, R.; Mishra, S. INSENS: Intrusion-tolerant routing for wireless sensor networks. *Comput. Commun.* 2006, 29, 216–230.
- [17] Perrig, A.; Szewczyk, R.; Tygar, J.D.; Wen, V.; Culler, D.E. SPINS: Security protocols for sensor networks. *Wirel. Netw.* 2002, 8, 521–534.
- [18] Liu, D.; Ning, P. Multilevel μ TESLA: Broadcast authentication for distributed sensor networks. *ACM Trans. Embed. Comput. Syst.* 2004, 3, 800–836.
- [19] Ye, F.; Luo, H.; Lu, S.; Zhang, L. Statistical en-route filtering of injected false data in sensor networks. *IEEE J. Sel. Areas Commun.* 2005, 23, 839–850.

- [20] National Institute of Standards and Technology. Secure Hash Standard (SHS); FIPS PUB 180-4; NIST: Gaithersburg, MD, USA, 2015.
- [21] Aumasson, J.-P.; Neves, S.; Wilcox-O’Hearn, Z.; Winnerlein, C. BLAKE2: Simpler, smaller, fast as MD5. In *Applied Cryptography and Network Security (ACNS 2013)*; Lecture Notes in Computer Science, Vol. 7954; Springer: Berlin/Heidelberg, Germany, 2013; pp. 119–135.
- [22] Bellare, M.; Canetti, R.; Krawczyk, H. Keying hash functions for message authentication. In *Advances in Cryptology—CRYPTO ’96*; Lecture Notes in Computer Science, Vol. 1109; Springer: Berlin/Heidelberg, Germany, 1996; pp. 1–15.
- [23] Heinzelman, W.R.; Chandrakasan, A.; Balakrishnan, H. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, Maui, HI, USA, 4–7 January 2000.