Low-power Tree-based Any-to-any Routing

Damiano Salvaterra
Department of Information Engineering and Computer Science
Univeristy of Trento
Trento, Italy
damiano.salvaterra@studenti.unitn.it

***Abstract*—This report presents the design, implementation, and evaluation of a tree-based any-to-any routing protocol for low-power wireless networks. The protocol leverages a collection tree to enable point-to-point communication across arbitrary nodes, using periodic beacons and topology reports to maintain routing tables with expiration policies. Parent selection is based on a cumulative ETX metric with hysteresis, supporting both ContikiMAC and NullRDC MAC layers. To reduce control overhead, the protocol supports piggybacking and adaptive timing strategies proportional to node depth. Simulations in Cooja demonstrate high reliability and low latency under NullRDC, while ContikiMAC shows lower delivery rates due to duty cycling. Parametrization is shown to impact performance trade-offs between energy consumption and protocol responsiveness. The protocol is also validated on the CLOVES testbed, confirming its applicability in real-world constrained IoT deployments.**

I. Introduction

This report presents the design, implementation, and evaluation of a tree-based any-to-any routing protocol for low-power wireless networks. Building upon an existing data collection tree, the protocol enables point-to-point communication by reusing upward links to the sink, downward routes to children, direct neighbor paths, and multi-hop forwarding via common ancestors. Each node maintains a local routing table populated through periodic and event-driven topology reports, with expiration mechanisms to remove stale entries. The protocol supports both NullRDC and ContikiMAC, minimizing control traffic and beacon collisions, and leveraging link-quality estimation techniques to cope with dynamic channel conditions.

The remainder of this report is organized as follows. Section II introduces the tree construction mechanism, including beacon flooding and parent selection based on a cumulative ETX metric with hysteresis and exponential smoothing. Section III describes the topology management strategy, detailing how nodes react to subtree changes and parent disconnections. Section IV defines the metric computation model and the link quality estimation based on packet statistics and RSSI. Section V presents implementation details, including packet formats, routing table design, and the piggybacking strategy for topology reports. Section VI discusses the parameterization of protocol timers and the implications for ContikiMAC compatibility. Section VII reports the results of extensive simulations in Cooja, followed by testbed experiments on CLOVES. Finally, Section VIII concludes the report and outlines possible directions for future enhancements.

II. Tree Building Procedure

The protocol relies on a three-stage tree construction process, structured as follows:

1) **Beacon Flooding:** The sink node initiates the topology discovery by broadcasting a beacon containing the following fields: a sequence number indicating the current tree epoch, a hop count equal to zero, a null parent, and a routing metric initialized to $\mathcal{M}_{\text{sink}} = 0$. Each receiving node parses the beacon to evaluate whether the advertising node can serve as a better parent.
2) **Beacon Forwarding and Parent Selection:** Upon reception of a beacon from a node s, a node v compares the advertised metric $\mathcal{M}_s$ with its current metric $\mathcal{M}_v$. If v is disconnected, its default metric is set to $\mathcal{M}_v = \infty$ (unreachable). After checking the freshness of the beacon via the sequence number, the node computes the cumulative path cost through s, denoted as $\mathcal{M}_v^{(s)}$. If
$$\mathcal{M}_v^{(s)} < \mathcal{M}_v - B(\mathcal{M}_v, H), \tag{1}$$
then v selects s as its new parent. Otherwise, s is added as a neighbor in the routing table. The hysteresis bias B acts as a stability threshold and is defined as:
$$B(\mathcal{M}_v, H) = \max\left(\Delta\mathcal{M}_v^{\min}, \frac{H}{\mathcal{M}_v}\right), \tag{2}$$
where H is a tunable parameter and $\Delta\mathcal{M}_v^{\min}$ is the minimum non-negligible metric variation. A larger value of H makes the parent selection more conservative, reducing oscillations in dynamic environments. Conversely, a smaller H increases responsiveness, enabling faster adaptation to transient link improvements. The presence of $\mathcal{M}_v$ in the denominator further reduces the bias for poor links, allowing even small relative improvements to be considered.
If v selects s as its parent, it schedules a topology report after a delay $\tau_{r_b}(d_v)$ and broadcasts a new beacon with updated metric and parent fields. Upon reception, s infers that v selected it as parent and adds v as a child in its local routing table.
3) **Subtree Advertisement and Reporting:** To enable coordinated and efficient topology dissemination, each node schedules its first topology report after a delay inversely proportional to its depth d_v in the tree:
$$\tau_{r_b}(d_v) \propto \frac{1}{d_v}. \tag{3}$$

Since leaf nodes have the highest depth, they report earlier, allowing intermediate nodes to piggyback their own information onto upstream reports. This strategy reduces redundant control traffic, especially under the NullRDC layer, where transmission latency is negligible. The exact expression for $\tau_{r_b}(d_v)$ is detailed in Section VI.

III. Topology Management

In the absence of significant topology changes, each node periodically emits *upstream topology reports* according to a timer $\tau_r(d_v)$, serving as *keep-alive messages*. These reports may be empty but still allow the receiver (i.e. the parent) to refresh the validity of routing table entries related to the sender and its managed subtree $\mathcal{T}_v$. Additionally, beacons are periodically reissued by the sink every T_B seconds to restart the tree construction process described in Section II, enabling global topology resynchronization.

Let v denote a node in the network, $\mathcal{T}_v$ its subtree, and π_v its current parent. There are two categories of topology change that must be propagated through the network:

1) A descendant node $m \in \mathcal{T}_v$ becomes unreachable.
2) The parent π_v becomes unreachable by v, necessitating a parent switch.

Note that changes internal to $\mathcal{T}_v$ (as long as the nodes in $\mathcal{T}_v$ reamain the same) are not directly relevant to π_v or the rest of the tree, as routing is performed hop-by-hop. For completeness, we also acknowledge the case where a direct neighbor n of v becomes unreachable. In such cases, v simply removes the stale entry for n from its routing table and, if needed, routes to n via π_v, assuming the network can still reach n via other branches.

The following subsections detail the two cases of topology dynamics. The format of topology reports and their aggregation via piggybacking is described in Section V.

A. Handling Subtree Changes

Since the protocol does not support end-to-end acknowledgments, a node v can only assess the reachability of its immediate neighbors (parents and children). In particular, if a node $m \in \mathcal{T}_v$ becomes unreachable, this can only be detected by its parent π_m, typically after a series of failed transmissions or missing ACKs.

One option would be to propagate such failures immediately upstream toward the sink r, but this would incur unnecessary control overhead: packets destined to m would eventually be dropped at r anyway, and if m is only temporarily unavailable, this could result in oscillatory and energy-expensive behavior.

To avoid such flapping, the protocol opts for a more stable mechanism: when π_m detects the failure of its child m, it marks m and all its known descendants $\mathcal{T}_m$ as *expired*. This expiration is then included in the next scheduled topology report from π_m, which propagates upstream via the parent chain. Each intermediate node updates its own routing table by removing expired entries associated with $\mathcal{T}_m$, ensuring consistent topology state over time without generating excessive traffic.

B. Reactive Parent Switching

Since the original collection tree is designed for data collection to a sink, the upward path to the sink is more critical than downward routes to descendants, and is therefore monitored more aggressively. When a node v experiences repeated transmission failures to its parent π_v, it reactively initiates a parent change procedure.

Node v begins by marking π_v as expired, then purges all stale entries from its routing table to retain only valid routes when notifying the new parent. Then node then selects a new parent π'_v such that:

$$\pi'_v = \underset{n \in \text{Neigh}(v)}{\operatorname{argmin}} \ \mathcal{M}(n), \tag{4}$$

and updates its own metric accordingly.

To restore subtree connectivity, v advertises its entire subtree $\mathcal{T}_v$ to the new parent π'_v via a topology report. This report is forwarded upstream by π'_v and subsequent nodes until it reaches the sink. As a result, all nodes along the path to r install or refresh routes to nodes in $\mathcal{T}_v$, maintaining end-to-end reachability despite the local reconfiguration at v.

IV. Routing Metric and Link Quality Estimation

The routing metric employed by the protocol is a cumulative cost to the sink, where each hop contributes a local estimate of link quality. This local cost is based on an Exponentially Weighted Moving Average (EWMA) of the **Expected Transmission Count (ETX)**. Specifically, the total metric of a node v at time t is defined recursively as:

$$\mathcal{M}(v) = \mathcal{M}(\pi_v) + \text{ETX}_t^{v \to \pi_v}, \tag{5}$$

where π_v denotes the current parent of node v.

The local link ETX is estimated as:

$$\text{ETX}_t^{v \to \pi_v} = \begin{cases} f_{\text{RSSI}}(\text{RSSI}) & \text{if } N_{\text{ACK},t}^{\pi_v} = 0, \\ \alpha \cdot \text{ETX}_{t-1}^{v \to \pi_v} + (1-\alpha) \cdot \dfrac{N_{\text{TX},t}^{\pi_v}}{N_{\text{ACK},t}^{\pi_v}} & \text{if } N_{\text{ACK},t}^{\pi_v} > 0, \end{cases} \tag{6}$$

where $N_{\text{TX},t}^{\pi_v}$ and $N_{\text{ACK},t}^{\pi_v}$ denote the number of transmissions and acknowledgments recorded on the link $v \to \pi_v$ at time t ($t = 0$ is the system bootstrap), and $\alpha \in [0,1]$ is a tunable smoothing parameter. A higher α results in a more stable but less responsive estimate, while a lower α increases sensitivity to recent variations.

At system bootstrap, when no packets have yet been exchanged, ETX cannot be directly estimated. In this case, a heuristic function based on the measured RSSI is used:

$$f_{\text{RSSI}}(\text{RSSI}) = \begin{cases} 1 & \text{if RSSI} > r_{\text{high}}, \\ 10 & \text{if RSSI} < r_{\text{low}}, \\ 1 + \dfrac{r_{\text{high}} - \text{RSSI}}{r_{\text{high}} - r_{\text{low}}} \cdot 9 & \text{otherwise}, \end{cases} \tag{7}$$

where r_{high} and r_{low} define empirical thresholds to map signal strength into a default ETX value.

This hybrid approach ensures that parent selection can occur even before sufficient data traffic is available to compute empirical delivery ratios. Notably: - when $\alpha = 1$, the metric relies entirely on RSSI-based heuristics; - when $\alpha = 0$, it becomes a pure ETX estimator; - intermediate values enable smooth adaptation between responsiveness and stability.

Figures 1 and 2 illustrate the behavior of $f_{\text{RSSI}}(\cdot)$ and the evolution of $\text{ETX}_t^{v \to \pi_v}$ under different link conditions, respectively. Figure 3 shows the accumulation of $\mathcal{M}(v)$ along a branch of the tree as depth increases, under a simulated packet reception model.

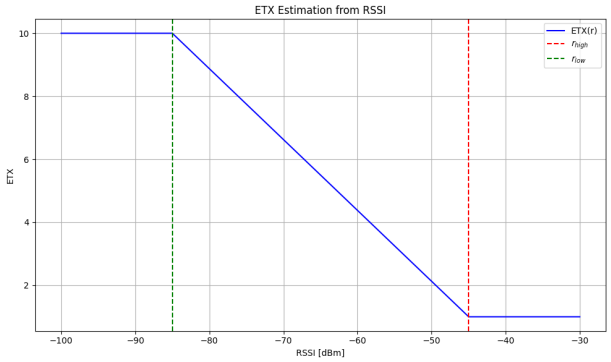


Fig. 1: $f_{\text{RSSI}}(\text{RSSI})$: heuristic estimation of ETX from received signal strength.

V. Notes on the Implementation

This section outlines the key implementation details of the protocol, including the structure and encoding of control packets, the piggybacking mechanism for topology dissemination, and the organization of the routing table. All components are designed to operate within the strict memory and packet size constraints of IEEE 802.15.4 networks, and conform to Contiki-OS conventions for modularity and resource efficiency.

A. Topology Reports

To minimize control traffic overhead, the protocol aggregates topology changes across the subtree rooted at each node. Topology reports are designed to be *piggybacked* as they propagate upstream. Each node maintains a buffer of pending changes in a vector of type `tpl_vec_t`, defined in `rp_types.h`, which stores an array of entries of type `stat_addr_t`—a compact (3-byte) structure containing a node address and a status flag (`STATUS_ADD` or `STATUS_REMOVE`).

When a topology report is sent or forwarded, the node appends its local topology changes to the report and flushes the buffer. This piggybacking mechanism allows reports generated by leaf nodes to be forwarded with additional information appended at each hop, significantly reducing redundant traffic.

Expired or unreachable descendants are explicitly marked for removal using the `STATUS_REMOVE` flag. New descendants (e.g., after a parent switch) are announced with `STATUS_ADD`. Intermediate nodes scan the received report and update their routing table entries accordingly.

B. Packet Formats and Header Structures

The protocol defines two message types: *beacons* and *topology reports*. Unicast packets are encapsulated with a compact custom header, while beacon messages are transmitted directly with a broadcast payload:

a) Packet Header (`uc_hdr`):

- `type` (1 byte): Message type (`UC_TYPE_DATA` or `UC_TYPE_REPORT`).
- `s_addr` (2 bytes): Source address.
- `d_addr` (2 bytes): Destination address.
- `hops` (1 byte): Hop count from source.

b) Beacon Payload (`bc_msg`):

- `seqn` (2 bytes): Sequence number identifying the tree epoch.
- `metric` (2 bytes, Q12.4 format): Cumulative path metric to the sink.
- `hops` (1 byte): Number of hops to the sink.
- `parent` (2 bytes): Parent node address.

c) Topology Report Payload: The report payload starts with a 1-byte field indicating the number of entries, followed by a packed array of `stat_addr_t` elements (3 bytes each). Each entry specifies:

- `addr` (2 bytes): Address of the child or descendant.
- `status` (1 byte): Either `STATUS_ADD` or `STATUS_REMOVE`.

The report payload is dimensioned to respect the IEEE 802.15.4 packet size limit. The maximum number of entries per report is computed at compile time as defined in `rp.h`.

C. Routing Table Entry Format

Each routing table entry (type `entry_t`) stores detailed per-destination forwarding state. The following fields are maintained:

- **Node type**: Indicates if the entry refers to a `PARENT`, `CHILD`, `NEIGHBOR`, or `DESCENDANT`.
- **Age timestamp**: Used to evaluate expiration.
- **Next-hop address**: Link-layer forwarder toward the destination.
- **Hop count**: Number of hops to the destination.
- **Link ETX**: Smoothed estimate for ETX, used in parent selection.
- **Counters**: Number of transmissions and acknowledgments observed.
- **Advertised metric**: The metric to the sink as advertised by the node corresponding to the entry.

Entries are updated upon reception of packets and purged when expired. When a topology change is detected (e.g., child becomes unreachable), affected entries are marked and reported in the next topology report.

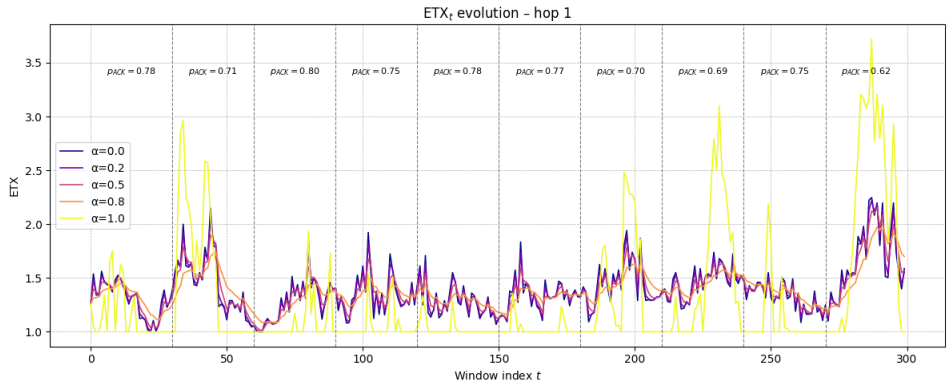


Fig. 2: Temporal evolution of $\mathrm{ETX}_t^{v\to\pi_v}$ for different values of ACK probability (p_{ACK}), modeled as a Bernoulli random variable. This simplified model is not meant to capture realistic channel behavior, but to highlight the smoothing effects of α.

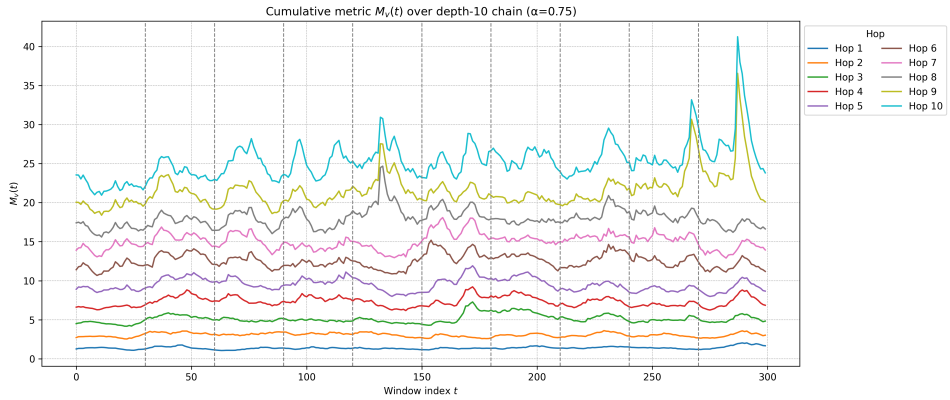


Fig. 3: Cumulative metric $\mathcal{M}(v)$ in a linear branch of depth 10. Each link is assigned a random p_{ACK}, with $\alpha = 0.75$. The RSSI distribution is modeled using the Gudmundson model with Rayleigh fading [1], [2].

VI. Timers and ContikiMAC

Timing configuration plays a central role in the performance and stability of the protocol. Two categories of timing parameters are defined: (i) those governing control-plane activity (e.g., beacon and topology report intervals), and (ii) randomized transmission delays (jitters), essential for collision avoidance and MAC-layer synchronization. These parameters are tuned differently for NullRDC and ContikiMAC to account for their different radio duty-cycling behavior.

In the case of ContikiMAC, particular care must be taken: due to its Low-Power Listening (LPL) design, neighboring nodes may converge on the same estimated wake-up schedule of a common parent, leading to synchronized transmissions and increased risk of collision—especially during beacon forwarding and report propagation. Moreover, because ContikiMAC does not acknowledge broadcast packets, frequent and uncoordinated beaconing may result in widespread packet loss. For this reason, all timers are scaled proportionally to node depth or radio wake-up intervals, and include carefully crafted jitter components.

The key parameters are detailed below.

Network Management Timers

- **Beacon interval** (T_B): The period between two beacon floods initiated by the sink. Defined as:

$$T_B = 60 \text{ [s]}$$

. A higher value reduces control overhead but slows convergence; lower values improve reactivity at the cost of increased energy.

- **Periodic topology report interval** $T_R(v)$: Defines how often a node v sends a keep-alive topology report. Computed as:

$$T_R(v) = \frac{T_B}{3}\left(1 + \frac{1}{d_v}\right) \text{ [s]}$$

.

- **Initial report delay**: After a tree rebuild, a node waits before sending its first topology report. This is also depth-aware and defined as:

$$T_R^{\text{first}}(v) = \frac{5}{d_v} + \delta \text{ [s]}$$

where δ is a jitter term dependent on the RDC layer:

  - For NullRDC: $\delta \sim \text{Uniform}(0, 0.4 \text{ s})$

– For ContikiMAC: $\delta \sim \text{Uniform}(0, 4 \cdot \text{CCI})$

where CCI denotes the CHannel Check Interval, CCI $= \frac{1}{\text{CCR}}$, where CCR denotes the Channel Check Rate, i.e., the frequency at which ContikiMAC wakes up to perform a Clear Channel Assessment (CCA).

Randomized Jitter Delays

- **Beacon forward jitter** (δ_B): Introduced in the broadcast beacon forwarding delay, defined as:

$$\delta_B = \begin{cases} \text{Uniform}\left(0, \frac{1}{8}\right) \text{ [s]}, & \text{NullRDC} \\ \text{Uniform}\left(0, 8 \cdot \text{CCI}\right) \text{ [s]}, & \text{ContikiMAC} \end{cases}$$

  This helps to avoid beacon collisions in dense topologies.
- **Topology report jitter** (δ_R): Applied before sending a unicast topology report. Defined as:

$$\delta_R = \begin{cases} \text{Uniform}(0.1, 0.2) \text{ [s]}, & \text{NullRDC} \\ \text{Uniform}(0.1, 4 \cdot \text{CCI}) \text{ [s]}, & \text{ContikiMAC} \end{cases}$$

,

The parameters above are selected to strike a balance between convergence speed, energy efficiency, and collision mitigation. They are modular and configurable at compile-time via preprocessor definitions in `rp.h`, with RDC-specific variants triggered by the `RDC_MODE` macro defined in `project-conf.h`.

VII. Experimental Results

The evaluation of the protocol is carried out through a two-step approach: large-scale simulations using the Cooja simulator, and limited deployments on the CLOVES testbed. The goal is to assess reliability (PDR), latency, and energy efficiency (duty cycle) under different MAC configurations and parameter sets. Simulations allow for statistically significant results under controlled conditions, while testbed experiments validate the protocol's behavior in real environments.

a) Simulation Results: All simulations are based on the provided scenario `test_nogui_dc.csc`. Each configuration was executed 100 times, with a simulated duration of 15 minutes per run. The three parameters varied were the parent selection hysteresis H, the EWMA smoothing factor α (Equation 6), and the Channel Check Interval (CCR) of ContikiMAC. The MAC layer used was either ContikiMAC (with CCR $\in \{8, 16, 32\}$ Hz) or NullRDC (baseline). The configurations tested are:

- **CM8-con**: ContikiMAC, $CCR = 8$ Hz (default), $H = 100$, $\alpha = 0.9$
- **CM16-con**: ContikiMAC, $CCR = 16$ Hz, $H = 100$, $\alpha = 0.9$
- **CM32-con**: ContikiMAC, $CCR = 32$ Hz, $H = 100$, $\alpha = 0.9$
- **CM8-res**: ContikiMAC, $CCR = 8$ Hz, $H = 50$, $\alpha = 0.5$
- **CM16-res**: ContikiMAC, $CCR = 16$ Hz, $H = 50$, $\alpha = 0.5$
- **CM32-res**: ContikiMAC, $CCR = 32$ Hz, $H = 50$, $\alpha = 0.5$
- **CM8-bal**: ContikiMAC, $CCR = 8$ Hz, $H = 75$, $\alpha = 0.75$
- **CM16-bal**: ContikiMAC, $CCR = 16$ Hz, $H = 75$, $\alpha = 0.75$
- **CM32-bal**: ContikiMAC, $CCR = 32$ Hz, $H = 75$, $\alpha = 0.75$
- **NRDC-cons**: NullRDC, $H = 100$, $\alpha = 0.9$

As expected, the latency results exhibit a strong dependency on the underlying MAC layer. As shown in Figure 4, NullRDC maintains consistently low latency across all configurations due to its continuous radio activity. In contrast, ContikiMAC introduces a latency component that grows with the sleep interval—inversely proportional to the Channel Check Rate (CCR). Thus, configurations with lower CCR values (e.g., 8 Hz) experience significantly higher delays.

A similar trend is observed for Packet Delivery Ratio (PDR). In ContikiMAC-based simulations, increasing the CCR results in higher and more stable PDR values. This behavior can be attributed to the increased likelihood that a receiver is awake when a packet is transmitted (i.e. triggered by the application or transmitted by a neighbor) reducing the number of retransmission attempts and the associated contention on the channel.

This hypothesis is further supported by the observed duty cycle (DC) statistics. While a higher CCR naturally leads to a higher average DC due to more frequent radio wake-ups, the lowest-CCR configuration ($CCR = 8$ Hz) shows a surprisingly high DC variance. This suggests that nodes in this setting often fail to complete packet delivery on the first attempt (either due to collisions or sleeping neighbors) and are forced to keep the radio active for extended periods within a CCR cycle.

Notably, the $CCR = 16$ Hz configuration demonstrates a compelling trade-off: its average duty cycle is similar to the 8 Hz case, but with significantly lower standard deviation, higher reliability ($PDR \sim 95\%$), and reduced latency. This confirms that moderate increases in CCR can yield substantial benefits in performance consistency.

On the other hand, excessively high CCR values (e.g., 32 Hz) further improve PDR (up to $\sim 98\%$) and reduce latency (by roughly 60 ms), but at the cost of an increased duty cycle (from $\sim 2\%$ to $\sim 3.2\%$). Whether such a configuration is preferable depends on application requirements: for applications demanding maximum reliability, a 1% increase in DC may be an acceptable trade-off.

A full summary of simulation statistics across all configurations is provided in Table I.

b) Testbed results: A subset of configurations was tested on the CLOVES testbed under the following setups:

- **Daytime, nodes 11–26**, see Figure 5c
- **Daytime, nodes 1–36**, see Figure 5d
- **Nighttime, nodes 11–26**, see Figure 5a
- **Nighttime, nodes 1–36**, see Figure 5b

Due to time constraints on the shared testbed infrastructure, it was not possible to execute multiple runs per configuration,

preventing statistical analysis of the experimental data. Nevertheless, the results exhibit patterns that are consistent with the trends observed in simulation.

For instance, as shown in Figure 5a, the configuration using ContikiMAC with the default $CCR = 8$ Hz achieves a lower PDR and a higher duty cycle compared to the same protocol with $CCR = 16$ Hz. This behavior aligns with the simulation results and suggests that extended sleep intervals increase the likelihood of collisions and retransmissions, leading to higher energy expenditure and degraded reliability.

Interestingly, daytime experiments on the 11–26 node subset reveal a case where ContikiMAC achieves a slightly higher PDR than NullRDC (by approximately 0.6%). While this deviation is likely due to statistical noise, it also highlights that the protocol remains highly effective even under strong duty-cycling constraints. Moreover, the fact that NullRDC underperforms during daytime, compared to its own nighttime results, indicates that performance degradation may be attributed to external interference from co-located technologies such as Wi-Fi or Bluetooth, which are typically more active during daytime hours.

Despite these limitations, the protocol achieves comparable performance across the two different topologies with NullRDC, demonstrating resilience to variations in network structure. In the node set spanning nodes 1–36, performance degradation with ContikiMAC is significantly more pronounced. This effect is further exacerbated when using parameter configurations that increase the protocol's responsiveness to link quality variations. Such behavior indicates that the wireless channel in this topology is particularly challenging, placing additional stress on the routing mechanism. Nonetheless, the observed upward trend in PDR across configurations suggests that further tuning of protocol parameters could lead to improved reliability.

VIII. Conclusions and possible improvements

The Solution resented demonstrates reliability even with high duty cycling in a realistic scenario, and resilience in network topology. Still, its performance degratates in a harsh topology structure or in a scenario with high interference. To mitigate these, several otpimization may be possibile. First of all, an adaptive beaconign mechanism may be possibile (eg Trickle-like algorithms) that adaptively change the beacon time and the topology report intervals to meet the scenario requirement (e.g, a simple implementation may be to trigger a new becaon flood after the root receives 2 or 3 topology reports that reports changes in the topology). A more interesting optimization may be to make the protoco adapt the parameters in the metric comptation adaptively to the scenario, like H, CCR, r_{high}, r_{low}, α, ecc. this may make possible a "blind" deployme nt of the protocol in a general scenario and let the algorithm optimize and converge to the optimal set of parameters, and also adapt in case of change of the envirmonment (eg, from day to night in a scenario like the CLOVES testbed).

The proposed solution demonstrates robust performance under strong duty-cycling constraints and exhibits resilience across different network topologies on real-world testbeds. However, performance degradation is observed in scenarios characterized by harsh topological structures or significant external interference.

To address these limitations, several optimizations can be envisioned. First, an *adaptive beaconing mechanism* could be introduced, inspired by Trickle-like algorithms, where the beacon and topology report intervals are dynamically adjusted based on observed network dynamics. For instance, a simple rule could trigger a new beacon flood if the sink receives multiple topology reports indicating changes in the network structure (e.g., two or three consecutive reports with subtree modifications).

A more advanced direction would involve *adaptive parameter tuning* for the metric computation. Parameters such as the hysteresis threshold H, the Channel Check Interval (CCR), RSSI thresholds r_{high} and r_{low}, and the EWMA smoothing factor α could be made dynamically adjustable. This would enable a form of scenario-aware or even self-optimizing behavior, where the protocol autonomously converges to an optimal configuration during runtime.

Such adaptive mechanisms would be particularly useful for deployments in dynamic environments(such as diurnal wireless interference patterns observed in the CLOVES testbed) where channel conditions and traffic patterns vary over time. Enabling the protocol to react to these changes would improve long-term reliability, reduce energy consumption, and allow for "blind" deployments without prior scenario-specific tuning.

References

[1] M. Gudmundson, "Correlation model for shadow fading in mobile radio systems," *Electronics Letters*, vol. 27, no. 23, pp. 2145–2146, 1991.

[2] A. Goldsmith, "Wireless communications," pp. 64–98, 2005.

TABLE I: Statistics of PDR, Laatency and Duty Cycle for each simulations batch (mean, standard deviation and 95& confidence interval)

Batch	PDR (%)			Latency (ms)			Duty Cycle (%)		
	Mean	Std	95% CI	Mean	Std	95% CI	Mean	Std	95% CI
batch-CM16-bal	97.64	2.07	[97.22, 98.07]	89.70	6.75	[88.26, 91.14]	3.20	0.03	[3.19, 3.21]
batch-CM16-con	94.90	2.94	[94.30, 95.50]	152.09	20.12	[147.99, 156.19]	2.06	0.09	[2.04, 2.08]
batch-CM16-res	94.44	3.84	[93.66, 95.23]	149.89	18.02	[146.18, 153.60]	2.07	0.09	[2.05, 2.09]
batch-CM32-bal	94.94	3.56	[94.19, 95.69]	148.51	19.35	[144.61, 152.41]	2.03	0.08	[2.01, 2.05]
batch-CM32-con	97.50	2.17	[97.06, 97.95]	88.34	6.87	[86.92, 89.75]	3.20	0.03	[3.19, 3.20]
batch-CM32-res	98.39	1.62	[98.04, 98.73]	88.26	7.36	[86.72, 89.80]	3.20	0.02	[3.19, 3.20]
batch-CM8-bal	83.92	9.43	[81.95, 85.88]	313.73	57.23	[302.07, 325.39]	2.10	0.36	[2.02, 2.17]
batch-CM8-con	83.57	9.52	[81.62, 85.52]	309.31	48.86	[299.41, 319.22]	2.13	0.33	[2.07, 2.20]
batch-CM8-res	82.63	10.15	[80.51, 84.74]	305.11	56.17	[293.67, 316.55]	2.05	0.27	[1.99, 2.11]
batch-sim-NRDC-cons	99.05	1.18	[98.81, 99.30]	18.88	0.47	[18.78, 18.98]	100.00	0.00	[100.00, 100.00]

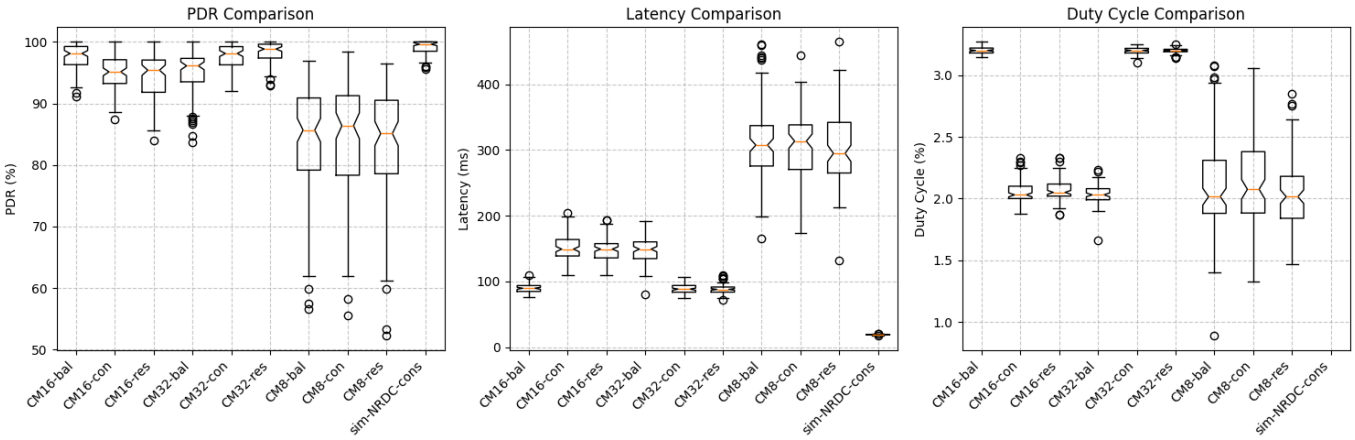
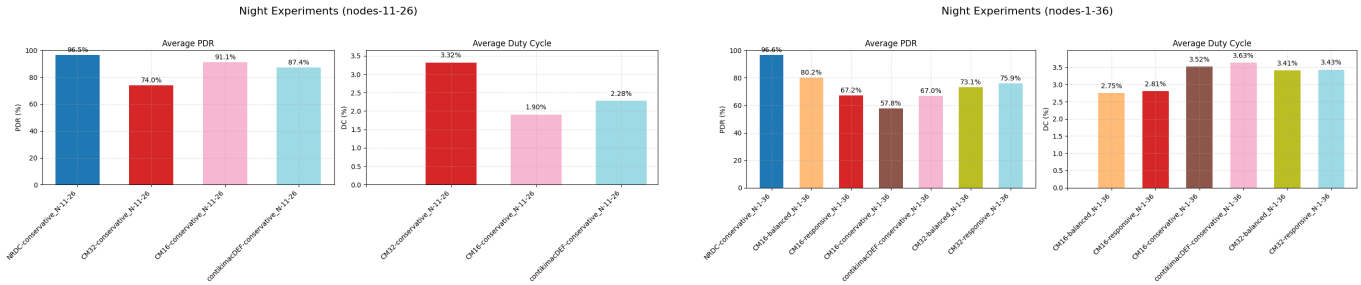
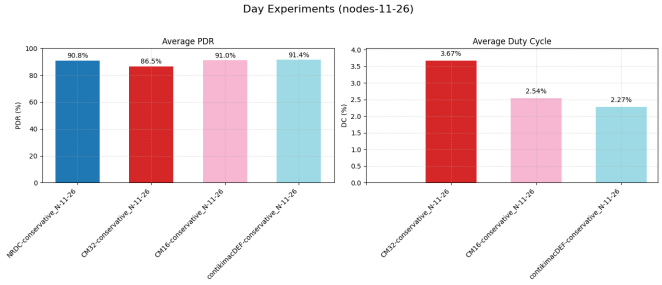


Fig. 4: Boxplots of PDR, Latency and DC of the simulation batches.

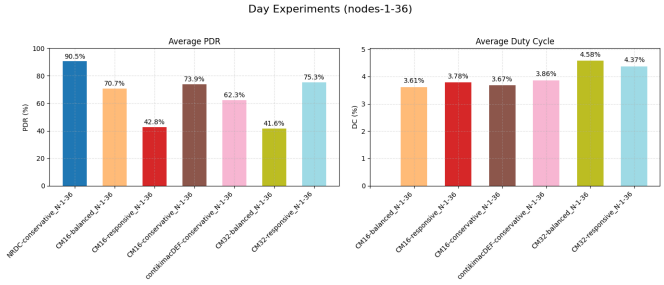


(a) Night (nodes 11–26)

(b) Night (nodes 1–36)



(c) Day (nodes 11–26)



(d) Day (nodes 1–36)

Fig. 5: Comparison of PDR and DC during day and night on the two different node sets.