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.

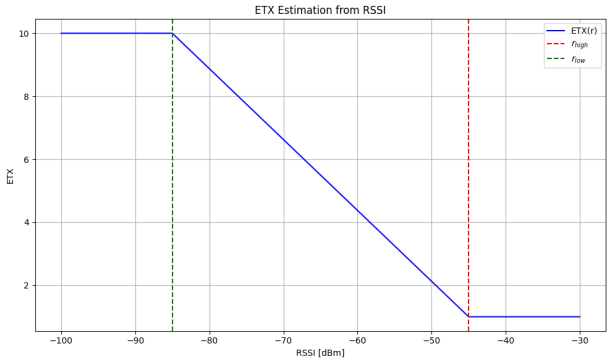

Fig. 1. $f_{\text{RSSI}}(\text{RSSI})$: heuristic estimation of ETX from received signal strength.

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.

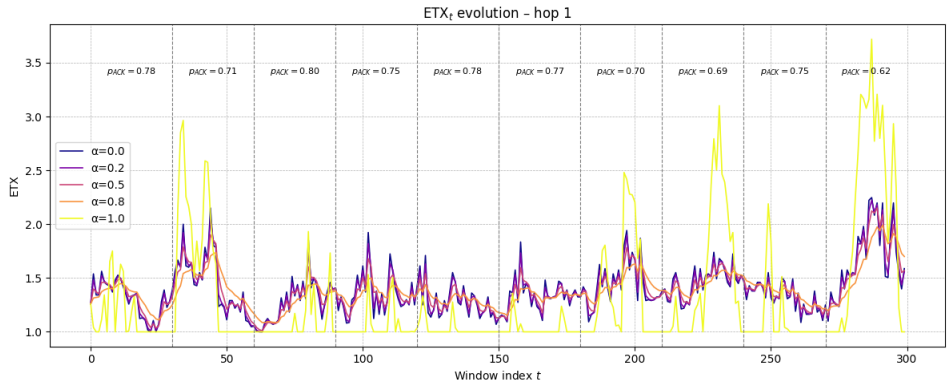

Fig. 2. Temporal evolution of $\mathrm{ETX}_t^{v\to\pi_v}$ for different values of ACK probability (p_{ACK}), modeled as a Bernoulli random variable. This simplified model is not meant to capture realistic channel behavior, but to highlight the smoothing effects of α.

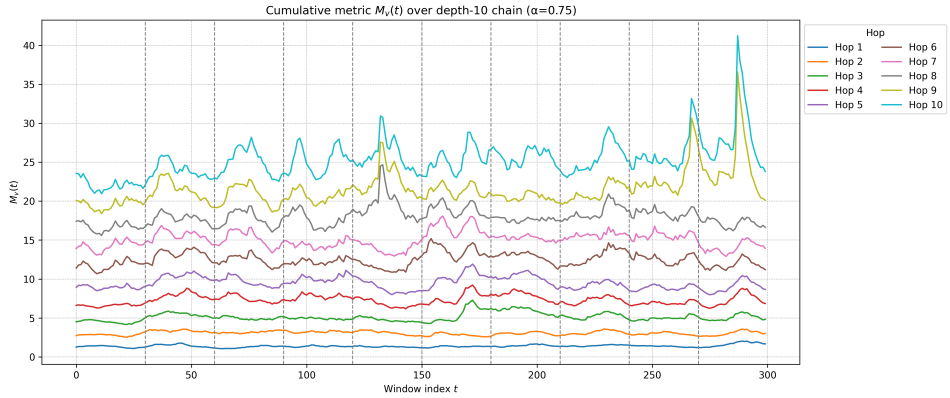

Fig. 3. Cumulative metric $\mathcal{M}(v)$ in a linear branch of depth 10. Each link is assigned a random p_{ACK}, with $\alpha = 0.75$. The RSSI distribution is modeled using the Gudmundson model with Rayleigh fading [1], [2].

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

Experiments are conducted first via simulation and then on the CLOVES testbed. The next two paragraphs reports the simulation and testbed results, respectively.

a) Simulation Results: Simulations has been performed with the Cooja simulator over the file **test_nogui_dc.csc** given by the instructor. In particular, some combination of parameters have been tested in simulation. Specifically, for each combination, 100 simulation of duration 15 minutes (simulated) have been executed. After some tests, the parameters that came out to be worth to test in batch simulation were essentially three (excluding the MAC layer): the hysteresis H, the Channel Check Rate, and α of Equation 6. The six combinations of parameters that are tested are:

- **sim-1**: ContikiMAC, $CCR = 32Hz$, $H = 100$, $\alpha = 0.9$: high CCR, conservative in selecting the parent;
- **sim-2**: ContikiMAC, $CCR = 16Hz$, $H = 100$, $\alpha = 0.9$: lower CCR, conservative in selecting the parent;
- **sim-3**: ContikiMAC, $CCR = 32Hz$, $H = 50$, $\alpha = 0.5$: high CCR, responsive (in time and in absolute value of the metric) in selecting the parent;
- **sim-4**: ContikiMAC, $CCR = 16Hz$, $H = 50$, $\alpha = 0.5$: lower CCR, responsive (in time and in absolute value of the metric) in selecting the parent;
- **sim-5**: ContikiMAC, $CCR = 32Hz$, $H = 75$, $\alpha = 0.75$: high CCR, balanced responsiveness.
- **sim-6**: ContikiMAC, $CCR = 16Hz$, $H = 75$, $\alpha = 0.75$: lower CCR, balanced responsiveness.
- **sim-7**: NullRDC, $H = 100$, $\alpha = 0.9$: conservative configuration using NullRDC, used as the baseline.

As expected, results in terms of Packet Delivery Rate (PDR) and Latency vary depending on the MAC layer used. Specifically, we see that with NullRDC the protocol has mean PDR near to 100% and mean latency ¡ 200 ms, while with ContikiMAC the performance worse: amongst the parameter configurations tried, the best one found has an average PDR around 95% and an average latency of 280 ms, and still with a higher variance with respect to the NullRDC one.This is due the the heavy Duty Cycle (DC) introduced by the ContikiMAC layer, which impacts the latency because of the sleep intervals, and also increases the collisions because the nearby nodes leanr the wake up time of the neighbors and so they are likely to send packets at the same time. The impact of the DC on PDR and latency are clear if we compare the simulations performed with different contikiMAC's Channel Ceck Rate (CCR)[1]: With a lower CCR, the DC decreases, but latency increases and the PDR increases, whereas with a higher CCR the opposite holds. It is also interesting to note that a lower CCR also increases the uncertainty (variance) on the performance estimation. Summary statistics of the simulations are reported in Table I, while Figure 4 shows the distribution of PDR, latency and DC with different parameters.

TABLE I
SUMMARY STATISTICS FOR SIMULATION BATCHES

Configuration	Metric	Mean ± Std	Median	95% CI (Mean)
MAC: ContikiMAC CCR=16 Hz T_B=60 s	PDR (%) Latency (ms) Duty Cycle (%)	89.06±4.66 335.14±25.43 2.63±0.20	88.78 333.13 2.62	(88.13,89.98) (330.10,340.19) (2.60,2.67)
MAC: ContikiMAC CCR=16 Hz T_B=120 s	PDR (%) Latency (ms) Duty Cycle (%)	88.22±6.51 328.45±28.98 2.34±0.22	88.78 325.70 2.30	(86.93,89.51) (322.70,334.20) (2.30,2.39)
MAC: ContikiMAC CCR=32 Hz T_B=60 s	PDR (%) Latency (ms) Duty Cycle (%)	94.29±4.03 278.68±22.21 3.56±0.17	95.14 274.98 3.53	(93.49,95.09) (274.28,283.09) (3.53,3.59)
MAC: ContikiMAC CCR=32 Hz T_B=60 s $\alpha = 0.9$	PDR (%) Latency (ms) Duty Cycle (%)	94.06±3.51 276.11±17.36 3.56±0.16	94.33 273.24 3.55	(93.37,94.76) (272.66,279.55) (3.53,3.59)
MAC: ContikiMAC CCR=32 Hz T_B=120 s	PDR (%) Latency (ms) Duty Cycle (%)	94.76±4.33 272.20±17.47 3.38±0.16	95.28 271.36 3.35	(93.90,95.62) (268.73,275.67) (3.35,3.41)
MAC: NullRDC T_B=60 s	PDR (%) Latency (ms) Duty Cycle (%)	98.80±1.59 191.46±8.92 100.00±0.00	99.60 190.75 100.00	(98.49,99.12) (189.69,193.23) (100.00,100.00)

b) Testbed Results: The real scenario experiments are conducted in two portions of the CLOVES testbed. Specifically, the **nodes [11,26]** (referred as topology A) and **[1,36]** (referred as topology B), in both cases with the sink node number 11. Due to the limited time available on the testbed,

[1]CCR i a tunable parameter of Contikimac, that is the frequency with which the node wake up to perform a Clear Channel Assessment, e.g. with CCR = 32Hz, the nodes wakes up every $\frac{1}{CCR} = 31.25$ms

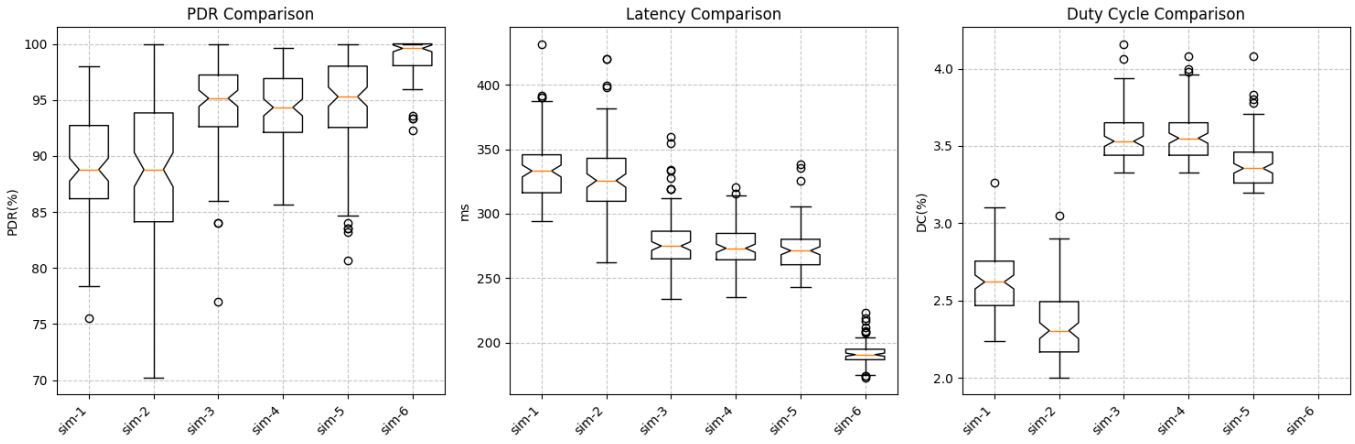

Fig. 4. Comparison of PDR, latency, and duty cycle across the different simulation configurations. sim-6 is the simulation batch ran with the NullRDC layer and can be visually used as baseline. Clearly, with NullRDC the DC is not computed.

only a small number of tests have been executed and so it is not possible to statistically analyze the results.

VIII. Conclusions

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.