Proc. of International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA 2025)
7-9 August 2025, Antalya-Türkiye

Wireless Mesh Networks Routing Optimization using Machine Learning and Deep Learning

Eric Tran
Institute of Communication Technologies (IICT)
University of Applied Science (HES-SO)
Yverdon, Switzerland
eric.tran@heig-vd.ch

Yann Charbon
Institute of Communication Technologies (IICT)
University of Applied Science (HES-SO)
Yverdon, Switzerland
yann.charbon@heig-vd.ch

Mahboob Karimian
Institute of Communication Technologies (IICT)
University of Applied Science (HES-SO)
Yverdon, Switzerland
mahboob.karimian@heig-vd.ch

Pierre Favrat
Institute of Communication Technologies (IICT)
University of Applied Science (HES-SO)
Yverdon, Switzerland
0000-0002-5735-709X

***Abstract*—Wireless mesh networks (WMNs) are increasingly deployed in large-scale infrastructures, requiring efficient and reliable routing to ensure high performance. Traditional routing protocols such as RPL rely on local decision-making, which often leads to suboptimal global routing structures, particularly in dense or complex topologies. This work investigates whether machine learning (ML) and deep learning (DL) techniques can be used to globally optimize Directed Acyclic Graph (DAG)-based routing in WMNs.**

To enable this exploration, we developed a high-performance simulation framework capable of evaluating the routing efficiency of arbitrary spanning trees using realistic link quality metrics derived from physical parameters such as RSSI. The simulator generates a labelled dataset of optimal and suboptimal routing trees across randomly generated topologies, which we use to train various ML and DL models. Our approach frames the optimization task as a multi-label classification problem, predicting the presence of links in the optimal routing DAG.

Despite extensive experimentation with classical classifiers, MLPs, and RNNs, our results show that standard ML/DL models consistently underperform compared to RPL, particularly as topology size increases. However, the simulator consistently identifies spanning trees that outperform RPL in terms of global routing efficiency, highlighting the potential for further improvement. Our findings suggest that while naive ML/DL approaches are insufficient to solve this NP-hard problem, our methodology and tools lay the groundwork for future research into constraint-aware or graph-based learning techniques for global mesh network optimization.

***Index Terms*—Machine Learning, Deep Learning, Routing, Wireless Mesh Networks, DAG, Optimization, Fast Simulator, Spanning Tree, Classification, MLP, RNN**

I. Introduction

Wireless mesh networks (WMNs) are increasingly adopted across various standards (e.g. Wi-SUN, Thread) to interconnect thousands of nodes. Ensuring the efficient operation of such large-scale networks requires an effective routing algorithm capable of minimising latency, congestion, and bottlenecks.

Although protocols like RPL [1] build a Directed Acyclic Graph (DAG) or spanning tree based on local decision-making, such decentralized approaches can lead to global inefficiencies. Slight local changes often cascade into suboptimal overall topologies, especially in dense or complex WMNs. A more global perspective is required to improve routing performance, yet DAG structural optimization is NP-hard [2].

In this work, we investigate whether ML/DL can learn global DAG structures that improve network-wide performance, thereby overcoming the local shortcomings of RPL. We developed a high-performance simulator to generate realistic, physics-based datasets for supervised learning, then applied classical ML classifiers, MLPs, and RNNs to predict globally optimized DAGs. Despite the simulator showing that solutions better than RPL do exist, naive ML models could not outperform RPL's. Our results highlight the difficulty of applying standard ML/DL in a domain with strict acyclicity and parent constraints, suggesting more advanced graph-based or constraint-aware learning approaches are needed.

A. Related Work

Research on WMN routing often focuses on local protocol improvements or specialized link metrics [3], rather than globally optimized DAG structures. Meanwhile, ML for combinatorial graph problems has shown potential in tasks like TSP or subgraph selection [4]. Methods like NOTEARS [2] propose continuous relaxations to learn DAGs but without spanning-tree constraints. Thus, our problem lies at the intersection of classical RPL-based WMN routing and cutting-edge ML approaches for global structural optimization.

Funded by the Swiss Innovation Agency under grant 104.251 IP-ICT

II. Methodology

We aim to replace or augment local routing decisions with an ML-driven global topology constructor. In practice, the local routing protocol (e.g., RPL) can form an initial tree; then all nodes report RSSI and link-quality data to a central service, which trains or applies an ML model to infer a better DAG. The new DAG constrains the parent-child relationships in the network, although local overrides can occur for drastic link changes.

We developed a *fast simulator* to generate a labeled dataset of network topologies and corresponding spanning trees, and an *emulator* to replicate real-world behavior for cross-validating both the predicted topologies and the simulator's outputs.

The dataset is used as training data for different ML/DL models. Different models are compared against each other, but also cross-validated against the performance of RPL (emulator) and the simulator.

Both the simulator [5] and the emulator [6] are openly available on GitHub for further experimentation and validation.

A. Cross-validation

A key aspect of our methodology is to evaluate ML/DL model predictions in an environment that closely approximates real-world conditions. Because we do not have access to multiple large-scale WMNs, we developed an emulator capable of modelling networks with hundreds or even thousands of nodes (subject to computational resources). This emulator runs a complete stack for each node, including PHY-level noise and collision modelling, MAC, and RPL.

III. Fast Simulator for Dataset Generation

A. Core Requirements and Workflow

The simulator must execute quickly, accurately estimate performance, and handle topologies of various sizes. Its pipeline, shown in Fig. 1, creates random topologies, enumerates subsets of possible spanning trees, and simulates each tree's end-to-end performance under a simple CSMA/CA-like model. The best performing tree is identified and stored alongside randomly sampled trees to create a rich training dataset.

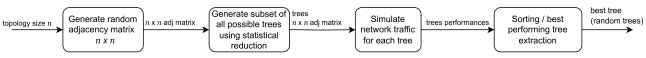

Fig. 1. High-level view of generation workflow

B. Topology Creation and Link Quality

We represent the network as an $n \times n$ adjacency matrix A, with entries $A_{ij} = 0$ for no link or $[0.85, 1.0]$ for viable links. We start with random uniform draws plus a density factor, ensure connectivity with DFS checks, and limit each node's neighbour count k to curb the exponential growth in possible spanning trees. Physical realism is maintained by mapping an RSSI threshold near $-96\,[\mathrm{dBm}]$ to an ≈ 0.85 probability of successful transmission (2-FSK assumption aligned with Wi-SUN standard).

1) Enumerating Spanning Trees: Naively, to form a spanning tree in a topology of n nodes, we must select exactly $n-1$ edges from the available set E. However, the total number of such combinations grows super-exponentially with n and the average node degree, making exhaustive search quickly infeasible.

To manage this, we apply a *skip factor* ξ, skipping large blocks of combinations. If i_S is our current combination index in the sorted list of $C = \binom{|E|}{n-1}$ subsets, we jump ahead by

$$i_S \leftarrow i_S + \tau, \quad \tau = U(0,\, (n-1)\cdot|E|\cdot\xi)$$

where $U(a,b)$ is a uniform integer in $[a,b]$. By performing this statistical analysis on various topologies of different sizes we define:

$$\xi(C) \;=\; \max\!\Big(1,\; \big\lfloor \tfrac{C}{30{,}000{,}000} \big\rfloor\Big) \tag{1}$$

This function is conservative and could potentially be further optimized.

Fig. 2 shows how larger ξ can discard the best 5% of trees in a topology. Once candidate subsets are formed, we validate each for connectivity using DFS; valid trees are simulated (Section III), and their best performers become references, alongside randomly selected suboptimal examples to broaden the dataset.

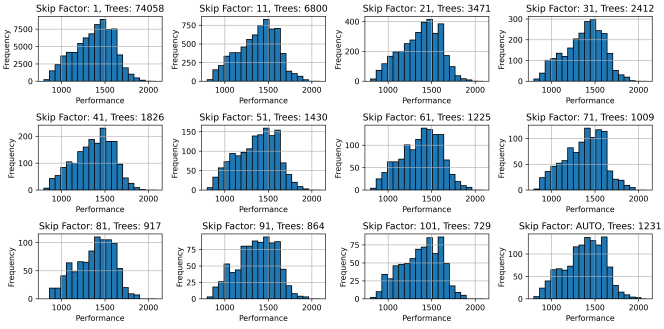

Fig. 2. Occurrence of tree performance values grouped into 5% bins, example topology of $n = 20$, density factor of $d = 0.55$ (Total combinations $C = 1,855,967,520$). Lower performance values correspond to better-performing trees.

C. Simulation of Uplink/Downlink Flooding

We assess the routing performance (i.e. the overall network bandwidth) of each spanning tree by simulating network traffic in both uplink ($down \rightarrow up$) and downlink ($up \rightarrow down$) directions *concurrently* at every time step, orchestrated by a simplified CSMA/CA-like mechanism with collisions and retransmissions. No actual data is transferred; instead, each node maintains counters representing the number of packets to transmit, effectively emulating packet buffers.

In the simulation, each non-root node is initially assigned m uplink packets to deliver to the root, while the root progressively injects downlink packets aimed at filling each node's buffer with m packets. At each simulation step, a random permutation of nodes is processed, where each node attempts either an uplink or downlink transmission. If both

phases remain incomplete, the direction is randomly chosen. Transmission success is determined probabilistically based on the link quality A_{ij}, while collisions or busy recipients result in deferred retries.

The simulation proceeds until all uplink packets have reached the root and each node has received its full quota of downlink packets, or until a user-defined maximum number of steps is reached. The final performance metric is the total number of simulation steps required, with fewer steps indicating more efficient routing.

The simulation algorithm is detailed by Algorithm 1.

Explanation of variables and conditions:

- E as the set of edges in the tree; for each node i, let $\mathcal{C}(i)$ be the set of all children nodes of node i, i.e.,

$$\mathcal{C}(i) = \{j \in N \mid (i,j) \in E\}$$

- A as the adjacency matrix describing the network topology, where A_{ij} denotes the probability of a successful transmission along the edge (i,j).
- $p_{up}(i)$ as the number of packets destined for the root node (uplink packets) held by node i.
- $p_{dwn}(i)$ as the number of packets destined for each node (downlink packets) held by node i.
- $\mathcal{I}_{up}(i)$ as a boolean indicating whether node i intends to transmit an uplink packet.

$$\mathcal{I}_{up}(i) = \begin{cases} 1, & \text{if } \mathcal{P}_{up}(i) > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $\mathcal{B}(i)$ as a boolean indicating whether node i is busy being engaged in a transmission (either transmitting or receiving a packet).
- Φ_{up} and Φ_{dwn} as the termination conditions for each simulation directions, formally defined as:

$$\Phi_{up} = \begin{cases} 0, & \text{if } \exists i \; p_{up}(i) = 0 \\ 1, & \text{otherwise} \end{cases} \quad \Phi_{dwn} = \begin{cases} 0, & \text{if } \exists i \; p_{dwn}(i) < m \\ 1, & \text{otherwise} \end{cases}$$

- Φ_{early} as the early termination condition defined by:

$$\Phi_{early} = \begin{cases} 0, & s > s_{max} \\ 1, & \text{otherwise} \end{cases}$$

The algorithm outputs the average (or capped value) of the number of steps required per epoch, which corresponds to the mean performance of the considered spanning tree. A lower number of steps implies a more efficient routing configuration. If the simulation exceeds a maximum step threshold, the process terminates early, ensuring efficiency in computation when it is used to get the best performing tree out of a set. This is performed by dynamically adjusting the threshold to not compute useless steps when the performance is bad.

This simulation is crucial for evaluating the global performance of spanning trees, simulating both uplink and downlink flooding concurrently. The algorithm is described in such way that it takes into account a lot of important features required to assess the routing performance, including overall bandwidth, overall latency, bottlenecks, and more physical feature such as link-quality and retransmissions.

This algorithm is available on GitHub [5].

Algorithm 1 Evaluate spanning-tree performance

Input: E (edges), n (nodes), L_{epochs} (simulation epochs), m (packets per node), A_{ij} (link quality), s_{max} (max steps)
for $l = 1$ **to** L_{epochs} **do**
  Initialize $p_{up}(i) = m,\ p_{dwn}(i) = 0, \mathcal{B}(i) = 0,\ \forall i \in N$
  Reset the simulation step counter $s \leftarrow 0$
  while $\Phi_{up} = 0$ AND $\Phi_{dwn} = 0$ **do**
    Count a new simulation step $s \leftarrow s + 1$
    Introduce on downlink packet at root:
    $p_{dwn}(0) \leftarrow p_{dwn}(0) + 1$
    for each node i **do**
      Update $\mathcal{I}_{up}(i)$ based on $p_{up}(i)$
    end for
    Uniform random permutation of node order: $\pi(N)$
    for each node $i \in \pi(N)$ **do**
      Randomly decide transmission direction $d \in \{-1, +1\}$
      if $d = +1$ (uplink) **then**
        if $\mathcal{B}(i) = 0$ **then**
          Select a random child j from $\mathcal{C}(i)$ with $\mathcal{I}_{up}(j) = 1$
          if $A_{ij} > U(0,1)$ AND $p_{up}(i) < m$ **then**
            Transmit:
            $p_{up}(i) \leftarrow p_{up}(i) + 1,\ p_{up}(j) \leftarrow p_{up}(j) - 1$
            Parent is busy while receiving: $\mathcal{B}(i) \leftarrow 1$
            Reset intent: $\mathcal{I}_{up}(j) \leftarrow 0$
          end if
          Child is busy both on TX success/failure: $\mathcal{B}(j) \leftarrow 1$
        end if
      else if $d = -1$ (downlink) **then**
        if $\mathcal{B}(i) = 0$ AND $p_{dwn}(i) > 0$ **then**
          Select a random child j from $\mathcal{C}(i)$
          if $\mathcal{B}(j) = 0$ AND $A_{ij} > U(0,1)$ AND
            $p_{dwn}(j) < m$ **then** ▷ Not reached target packets
            Transmit:
            $p_{dwn}(i) \leftarrow p_{dwn}(i) - 1,\ p_{dwn}(j) \leftarrow p_{dwn}(j) + 1$
            Child is busy while receiving: $\mathcal{B}(j) \leftarrow 1$
          end if
          Parent is busy both on TX success/failure: $\mathcal{B}(i) \leftarrow 1$
        end if
      end if
    end for
    Reset $\mathcal{B}(i) \leftarrow 0,\ \forall i$
    $p_{up}(0) \leftarrow 0$ ▷ Root does not store uplink packets
    Compute $\Phi_{up}, \Phi_{dwn}, \Phi_{early}$
    if $\Phi_{early} = 1$ **then**
      Break loop
    end if
  end while
  Accumulate sim. steps across epochs: $s_{cumul} \leftarrow s_{cumul} + s$
  if $\Phi_{early} = 1$ **then**
    Exit epoch loop
  end if
end for
Output:

$$\bar{s} = \begin{cases} \frac{s_{cumul}}{L_{epochs}}, & \text{if } \Phi_{early} = 0 \\ s_{max}, & \text{otherwise} \end{cases}$$

D. Dataset generation

To generate a reference dataset suitable for ML/DL, multiple executions of the simulation workflow are performed. This process computes the best tree(s) for each topology. While the raw adjacency matrix and best tree data can be used directly for training, introducing additional telemetry data enhances

the model's diversity. An extra simulation step is employed to compute metrics for each tree, including:

- Node ranks
- Transmission success/failure rates
- Retransmission rates

These metrics provide valuable context for training ML/DL algorithms, potentially improving the model's understanding of network dynamics and performance.

IV. Machine Learning Approaches

A. Problem Formulation and Data Post-Processing

We treat our problem as *multi-label classification*: for a topology with n nodes, we create a vector of length $n(n-1)/2$ representing the upper-triangular entries of the adjacency matrix. Each entry is assigned a label of 1 if the corresponding edge is used in the final tree, or 0 otherwise. We may also add "telemetry" features (e.g., rank or success rates) to enrich model inputs.

However, the predicted adjacency must still form a valid spanning tree with one parent per node and no cycles. After prediction, we apply a simple post-processing rule to ensure validity. If any node is disconnected, we attach it to the best candidate edge by predicted probability; if a cycle arises, we remove the edge with the lowest probability in that cycle. In practice, this step is crucial for compliance with DAG constraints.

B. Considered Models

We explored classical ML classifiers (logistic regression, SVM variants) and two DL paradigms (MLP, RNN/LSTM). All share the same fundamental classification target: predict 1 if an edge is in the optimal tree, else 0. For the RNN, we constructed a pseudo-time-series of multiple suboptimal trees from the same topology. The aim is to let the LSTM observe incremental variations and approach a best structure.

TABLE I
Summary of Model Architectures

Model	Core Arch.	Config	Loss
Logistic Reg.	-	L2 penalty	Multinomial
SVM (SGD)	-	L2 penalty	Hinge
Linear SVM	-	L2 penalty	Sq. Hinge
MLP	In/Dense/Dense/Out	214-495 ReLU	BCE
RNN (LSTM)	In/LSTM/Out	1000 LSTM	BCE

C. Evaluation Metrics and Cross-Validation

During training, we evaluate model performance using precision and recall, with the F1-score computed on the predicted adjacency vectors. However, these classification metrics alone are insufficient to fully characterize routing effectiveness. To address this, we incorporate a cross-validation step using network simulation. Specifically, the predicted adjacency matrix is fed into the same flooding protocol employed for evaluating RPL. The average number of simulation steps required for successful uplink and downlink transmissions serves as a direct measure of routing efficiency.

V. Results and Discussion

Fig. 3 first compares classical ML: linear SVM typically outperforms logistic regression slightly, while SVM+SGD lags behind.

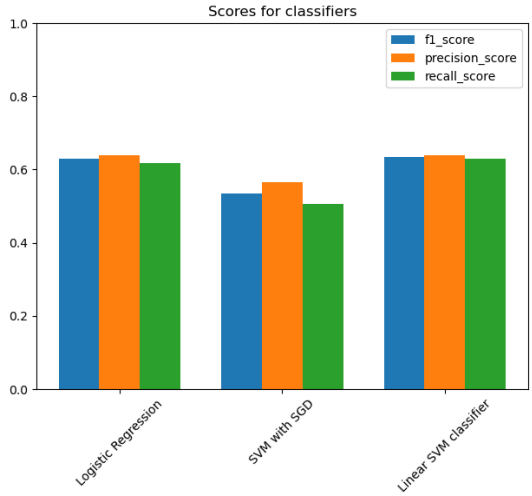

Fig. 3. F1-scores of ML classifiers (test fold). Linear SVM has a slight advantage.

We also tested MLP vs. RNN with various input sets (topology alone, plus telemetry, plus an initial RPL-based DAG). Fig. 4 shows the results, where the RNN has a marginally higher F1 score but is more expensive to train.

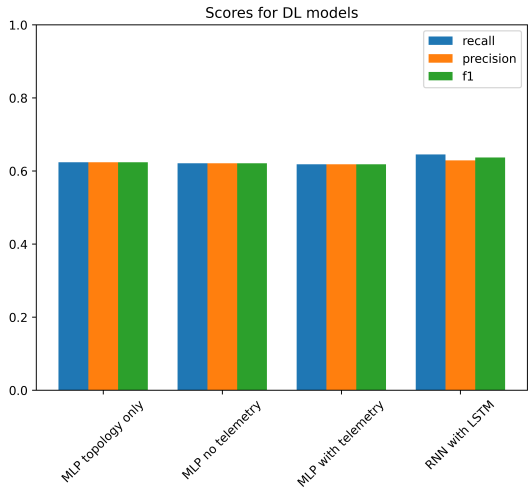

Fig. 4. Comparison of different DL architectures on classification metrics.

Most importantly, cross-validation by re-simulating the ML-predicted trees (Fig. 5) shows that all tested ML/DL solutions still perform *worse* on average than RPL. Meanwhile, the simulator's systematic exploration of candidate trees frequently identifies solutions that *outperform* RPL, demonstrating that improved topologies do exist but remain difficult to capture using standard classification approaches.

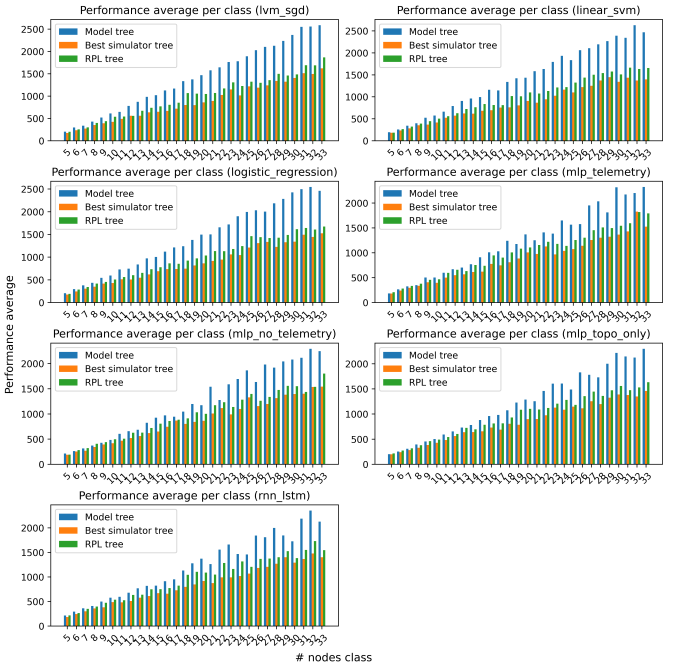

Fig. 5. Cross-validation results, comparing predicted trees (MLP or RNN) vs. RPL. Lower values are better.

A. Analysis

Our results highlight the difficulty of imposing DAG constraints via naive classification. The single-parent-per-node rule and the need to avoid cycles complicate direct ML approaches. Standard classification excels at recognising patterns but not enforcing combinatorial structure. Future advances will likely hinge on *graph-aware* or *constraint-driven* ML methods.

VI. Conclusion and Future Work

We explored global DAG routing optimisation in WMNs via ML/DL. We developed a high-performance simulator to generate realistic data and tested classical and deep classifiers. Although brute force can find spanning trees superior to RPL, our trained models did not surpass RPL performance, with the gap widening in larger network topologies.

Nonetheless, the simulator and dataset creation pipeline form a solid foundation. Future work should investigate:

- **Graph-Constrained Learning:** Graph Neural Networks (GNNs) or reinforcement learning that enforces spanning-tree structure explicitly.
- **Hybrid Approaches:** Leverage RPL's local heuristics but let a central solver refine partial constraints or detect global bottlenecks.
- **Scalability**: Extending the skip-factor sampling to larger n, perhaps with advanced importance-sampling or parallelisation.

In closing, purely data-driven methods for DAG routing remain challenging. With more sophisticated architectures or explicit constraints, a global ML-based optimiser may eventually complement or surpass RPL in real-world WMNs.

References

[1] T Winter et al. *Rpl: Ipv6 routing protocol for low power and lossy networks, RFC 6550, IETF, 2012*. 2011. DOI: 10.17487/RFC6550.

[2] Xun Zheng et al. *DAGs with NO TEARS: Continuous Optimization for Structure Learning*. 2018. arXiv: 1803.01422 [stat.ML]. URL: https://arxiv.org/abs/1803.01422.

[3] Georgios Parissidis et al. "Routing Metrics for Wireless Mesh Networks". In: Feb. 2009, pp. 199–230. ISBN: 978-1-84800-908-0. DOI: 10.1007/978-1-84800-909-7_8.

[4] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. *Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon*. 2020. arXiv: 1811.06128 [cs.LG]. URL: https://arxiv.org/abs/1811.06128.

[5] Eric Tran and Yann Charbon. *DAGFastSimulator*. 2025. URL: https://github.com/YannCharbon/DAGFastSimulator.

[6] Yann Charbon and Mahboob Karimian. *wisun-mbed-simulator*. 2024. URL: https://github.com/mahboobkarimian/wisun-mbed-simulator.