Simulation of the availability and initialization speed of a layer 2 mesh network*

Oleksii Dzhus[1,*,†] and Mykhaylo Lobur[1,†]

[1] Lviv Polytechnic National University, 12 S. Bandery str., 79000 Lviv, Ukraine

Abstract

This paper presents a comprehensive simulation-based study of availability and initialization speed in Layer 2 (L2) mesh networks, with a particular focus on the BATMAN-adv protocol. A high-fidelity software framework was developed to emulate physical-layer characteristics, collision-aware MAC behavior, and realistic OGM exchange dynamics. The model supports controlled cold-start experiments, traffic visualization, and scalability analysis across networks ranging from 2×2 up to 15×15 nodes. Results show that initialization time grows approximately with network diameter and becomes increasingly sensitive to control-plane contention as the network scales. For a 15×15 grid, typical convergence times ranged from 20–45 seconds depending on OGM intervals, link quality, and packet load. Visualization of traffic matrices revealed the transient instability characteristic of early initialization, while curve-fitting analysis using a three-parameter asymmetric model captured the statistical relationship between error rate and initialization time. The fitted coefficients showed consistent behavior across packet loads, demonstrating that traffic intensity primarily shifts convergence curves without altering their shape. Overall, the study provides a quantitative basis for predicting L2 mesh initialization dynamics and offers a validated simulation toolset for future research and real-world testing.

Keywords

wireless, mesh network, BATMAN, BATMAN-adv, originator message, network initialization, convergence speed, availability, simulation, wireless mesh routing, scalability, performance modeling

1. Introduction

Mesh networks have emerged as a critical infrastructure solution for modern communication systems, offering resilience, scalability, and decentralized operation across a wide range of applications, from Internet of Things sensor networks to emergency response systems [1]. Unlike traditional hierarchical network architectures, mesh networks enable nodes to communicate dynamically, creating self-organizing topologies that can adapt to changing conditions [2]. However, the initialization phase—during which nodes discover neighbors, establish connections, and form the network topology—represents a crucial performance bottleneck that directly impacts network availability and responsiveness. For L2 mesh networks operating at the data link layer, initialization speed becomes particularly significant as it determines how quickly the network can achieve operational readiness without relying on higher-layer routing protocols [3].

Despite extensive research on mesh network protocols and topology formation algorithms, limited attention has been paid to systematically modeling and simulating the initialization dynamics of L2 mesh networks under realistic conditions. Most existing studies focus on steady-state performance metrics or assume instantaneous network formation, overlooking the transient initialization period where network behavior is most unpredictable [4]. The initialization speed depends on complex interactions between multiple factors, including node density, radio propagation characteristics, medium access control protocols, and discovery mechanisms. Understanding these initialization dynamics through accurate simulation is essential for designing networks that can rapidly converge to stable operation, particularly in time-critical applications such as disaster recovery networks or tactical military communications.

This paper presents a comprehensive simulation framework for analyzing the speed of L2 mesh network initialization, incorporating realistic physical layer models, collision-aware MAC protocols, and various discovery strategies. Our approach provides novel insights into the temporal

* CPITS-II 2025: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2025, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

oleksii.p.dzhus@lpnu.ua (O. Dzhus); mykhaylo.v.lobur@lpnu.ua (M. Lobur)

0009-0004-0030-6162 (O. Dzhus); 0000-0001-7516-1093 (M. Lobur)

evolution of network connectivity during the critical initialization phase, revealing non-intuitive relationships between network parameters and convergence time. By examining initialization under different node distributions, discovery intervals, and interference patterns, we identify key optimization opportunities that can reduce initialization latency compared to standard approaches [5]. These findings have immediate practical implications for network designers seeking to minimize downtime and improve the reliability of mesh network deployments in dynamic environments.

2. Source Review

BATMAN-adv operates at L2, embedding routing information in data packets and eliminating the need for each node to maintain a full network map. This design reduces processing overhead and supports protocol-agnostic data forwarding, enhancing throughput and flexibility [6–8]. The protocol is valued for its portability, dynamic organization, and security, making it suitable for challenging environments such as disaster relief and remote areas [9–11].

BATMAN-adv demonstrates stable performance in many scenarios but faces challenges under heavy traffic, where route instability and increased resource consumption can occur [8, 9, 12, 13]. Comparative studies show that while BATMAN-adv is efficient in certain conditions, protocols such as OLSR and Babel may outperform it in high-mobility or high-traffic environments due to faster route adaptation and lower latency [14, 15]. Enhancements such as signal strength-based routing and packet aggregation have been proposed to improve stability and channel efficiency [10].

Recent work focuses on optimizing BATMAN-adv for faster reconnection after disruptions, improving routing criteria, and integrating with delay-tolerant networking for better packet delivery in intermittent conditions [10, 16, 17]. Hybrid approaches with SDN and new emulation tools are also being explored to enhance scalability and management [15, 18, 19].

3. Materials and Methods

3.1. Initialization Speed and Synchronization

In industrial wireless sensor networks with a mesh-star architecture, the Fast and Low-Overhead Time Synchronization algorithm achieves rapid initialization and synchronization. Simulations show that logical clock skew in the mesh layer converges within approximately 10 s, and full synchronization is achieved in about 20 s—significantly faster than traditional distributed algorithms like ATS and GTSP. The convergence speed is mathematically linked to the network's average node degree and size, with larger, well-connected networks converging more quickly. The iterative synchronization process in mesh networks can be mapped to a Markov chain, where the convergence rate depends on the second-largest eigenvalue of the transition probability matrix. This provides a theoretical basis for predicting initialization speed as a function of network parameters [20].

3.2. Impact of Communication Delays and Network Size

High-fidelity simulators (e.g., AirSim) have been used to model robot mesh networks, incorporating realistic robot-to-robot communication speeds (e.g., 0.25 Mb/s). These simulations demonstrate that decision and initialization times scale linearly with network size, and that sparser networks can improve scalability without severely impacting performance [20]. Efficient initialization schemes, such as two-level approaches for mesh partitioning, can reduce initialization time by up to fivefold in large-scale simulations (e.g., 24,576 CPU cores), with total initialization times under 6 min. for huge meshes [22].

Simulations of mesh-based P2P video streaming systems show that startup delays and buffering times are within industry standards, and that mesh architectures can efficiently handle peer churn and dynamic initialization [23, 24].

3.3. Rationale for Choosing Layer 2 Protocol

The relative stability of the BATMAN protocol when implemented at L2, compared with mesh routing protocols operating at Layer 3 (L3), can be understood by examining the architectural and functional distinctions between link-layer forwarding and network-layer routing in dynamic wireless environments. BATMAN-adv, the L2 incarnation of the protocol, conceptualizes the entire mesh network as a single Ethernet broadcast domain. By doing so, it avoids the intrinsic complexity associated with maintaining global network-layer state and instead employs a forwarding model that relies solely on local link information. This design fundamentally alters how the network responds to topology changes, making the system less sensitive to volatility in link quality and node availability.

The instability frequently observed in L3 mesh routing protocols arises from their need to construct, maintain, and periodically update global routing tables. Protocols such as OLSR, RIP, or similar IP-based mechanisms perform continuous dissemination of routing information and require convergence on consistent end-to-end path calculations. In wireless mesh networks, where nodes are mobile or subject to fluctuating radio conditions, the volume and frequency of necessary routing updates increase significantly. Each change in topology triggers a sequence of recalculations, control-plane exchanges, and propagation delays that can result in route flapping, packet loss, and periods of inconsistent routing information. These characteristics impose substantial overhead and prolong the network's convergence time, thereby reducing its operational stability. In contrast, BATMAN at L2 does not attempt to construct a global view of the network. Instead, it limits its perspective to the immediate neighbors of each node and relies on a metric derived from the periodic exchange of lightweight Originator Messages. Each node determines only the best next hop toward a destination, rather than computing the entire end-to-end path. This localized decision model enables topology changes to be absorbed at the point of occurrence, without triggering system-wide recomputation or the dissemination of complex routing updates. The absence of a global state significantly reduces the protocol's susceptibility to instability induced by transient or localized link variations. As a result, the protocol exhibits smoother adaptation to dynamic wireless conditions, because each node responds only to changes it can directly observe.

Operating at L2 also preserves the operational semantics of a conventional Ethernet network, enabling BATMAN-adv to handle broadcast and multicast traffic natively. Many L3 mesh protocols struggle with such traffic because IP routing is inherently designed for unicast forwarding, which often requires additional mechanisms or encapsulation strategies to support address resolution or service discovery protocols, such as ARP, DHCP, and IPv6 Neighbor Discovery. BATMAN-adv's ability to propagate these frames transparently across the mesh avoids the inconsistencies that can arise when L3 protocols attempt to emulate or approximate link-layer broadcast behavior. This property contributes further to stability, since standard network functions continue to operate as expected without imposing extra signaling or translation layers.

4. Results

4.1. Features of the BATMAN Protocol

BATMAN-adv is a L2 mesh routing protocol designed for wireless mesh networks, offering decentralized, self-healing, and flexible connectivity. Its open-source nature and adaptability make it popular for scenarios like disaster response, community networks, and mobile ad-hoc environments. BATMAN-adv is specifically designed not to depend on synchronized clocks of any kind. All of its routing decisions rely on relative sequence numbers, not timestamps. Therefore, it does not require global or local time synchronization mechanisms such as FLOTS, FTSP, TPSN, or any other fast/low-overhead time-sync system.

BATMAN-adv operates at L2 and evaluates link quality by exchanging Originator Messages (OGMs). These OGMs contain sequence numbers that are monotonically increasing, allowing each node to determine whether a message is new, old, or a duplicate. Because sequence numbers are relative counters generated independently by each node, BATMAN-adv never needs nodes to share a unified notion of clock time. Time-synchronization protocols create processing and communication overhead, and in unstable wireless mesh networks, synchronized time can drift quickly or become inconsistent. BATMAN-adv avoids this entire class of problems by relying on

sequence-based freshness, ensuring that routing remains robust even when nodes have no synchronized clocks, reboot frequently, or move rapidly.

4.2. Originator Message Initialization Speed

Every node periodically broadcasts an OGM (default interval $\Delta t = 1$ s) containing: O is a originator MAC address, S is a sender MAC address, $seqno_O$ is a sequence number, and Transmission Quality $TQ \in [0, 255]$, where 255 represents perfect transmission for a 15 by 15 matrix. When node **B** receives an OGM from neighbor **A**, the local reception quality (sliding window from 50 to 500 expected packets):

$$ERX_B(A) = \frac{received_{OGM}|^{\mathbf{A}}}{expected_{OGM}} \in [0,1]. \tag{1}$$

Reverse quality carried in the packet. The OGM contains the reverse link quality $TQ_{A \to B}$, as measured by **A**. Bidirectional link quality (**B**'s computation):

$$TQ_{B \to A} = ERX_B(A) \times TQ_{A \to B} \tag{2}$$

Thus, the end-to-end path TQ after h hops is approximately:

$$TQ_{\text{path}} \approx n^2 \prod_{i=1}^{h} \left(p_{\text{forward}}(i)\, p_{\text{preverse}}(i) \right), \tag{3}$$

where n is a matrix size.

Best next-hop selection for each node N maintains:

$$O \mapsto (\text{next-hop MAC}, \text{best TQ}, \text{last seqno}). \tag{4}$$

For an OGM for originator O received via neighbor N_i:

$$TQ_{\text{global}}(O \text{ via } N_i) = TQ_{\text{incoming}}. \tag{5}$$

Next-hop choice:

$$\text{next-hop}(O) = \arg\max_{N_i} TQ_{\text{global}}(O \text{ via } N_i), \tag{6}$$

where tie-breaking uses MAC address or sequence number [25].

A node rebroadcasts an OGM of originator O only if:

1. It was received from the currently selected best next-hop toward O, *or*
2. The new candidate TQ is significantly higher than the current best (hysteresis threshold).

This changes flooding complexity from $O(n^2)$ to $O(n)$ while keeping single-path loop-free routing [13].

Convergence and initialization speed used at cold start, all nodes begin broadcasting OGMs with $TQ = n^2$ and $seqno = 0$. Ideal-case convergence time (low-loss, symmetric links)

$$T_c \approx D \times \Delta t, \tag{7}$$

where D is a network diameter in hops, Δt is the OGM interval (default 1 s).

In a regular 15×15 grid with 4-connectivity (Manhattan layout) $D = 2 \times (15 - 1) = 28$ hops. Ideal convergence is $T_c \approx 28\,s$. Real-world measured convergence (accounting for EWMA windows of 100 OGMs, hysteresis, and minor losses) $T_c \approx 35 - 60\,s$ for 15×15 grids. Reducing Δt to 200–500 ms can bring 15×15 convergence below 15 s, at the cost of significantly higher control overhead [13, 26].

4.3. Development of a Software Model

The meshnet-lab package [27] was chosen for modeling, but other packages can also be used [28–30]. The square matrix is chosen for its greater connectivity, which corresponds to the best results that a mesh network can show in simulation.

Single-run measurement tool to determine the minimum time required for a batman-adv mesh network of given topology to reach full convergence (100% packet delivery) starting from a cold start:

- Performs guaranteed cold reset before the test.

- Iteratively increases delay after network start until 100% PDR is achieved.
- Prints classic line: 45s ⟶ 500 ok | 0 lost | 100.0%.
- Outputs convergence time in seconds.

```
Pythonwhile delay <= 180:
    full_cold_reset()
    setup_target_grid(rows, cols)
    time.sleep(delay)
    success = subprocess.run(["sudo", "./ping.py", "--pings", "500", "--timeout", "4"],
                             capture_output=True, text=True).stdout.lower().count("success")
    print(f"   {delay:>3}s ⟶ {success:>3} ok | {500-success:>3} lost | {success/5:6.1f}%")
    if success == 500:
        print(f"FULL CONVERGENCE ACHIEVED in {delay} seconds")
        break
    delay += step
```

Generates a high-quality vector-ready visualization of real traffic matrix in a batman-adv mesh after a directed ping flood. Successfully delivered and lost packets are shown as green arrows and red dashed arrows respectively:

Accepts --matrix R C --packets N --delay D --restart yes/no.

Draws physical links, successful flows (thickness ∝ packet count), lost flows (red dashed).

Displays exact packet counts on each directed link.

Uses NetworkX + Matplotlib with publication-ready style.

```
Pythonfor (u, v), cnt in success_count.items():
    width = min(6 + cnt * 2.8, 40)
    nx.draw_networkx_edges(G, pos, edgelist=[(u,v)], width=width,
                           edge_color="#006400", arrowsize=40,
                           connectionstyle=f"arc3,rad=0.25")
    mx, my = (pos[u][0]+pos[v][0])/2, (pos[u][1]+pos[v][1])/2 + 0.22
    plt.text(mx, my, str(cnt), fontsize=17, fontweight="bold",
             color="darkgreen", ha="center",
             bbox=dict(facecolor="white", alpha=0.9))
```

Studies how traffic intensity affects batman-adv convergence time. For a fixed topology, the number of simultaneously sent ping packets is increased (from ping_step to max_pings). For each load level a full convergence curve (PDR vs time) is built:

- Cold reset before every single measurement.
- Starts from user-defined step (e.g. 100, 200, … instead of 50).
- Produces family of curves “PDR vs time” with different colours per load.
- Prints complete raw data table before the plot.

```
Pythonping_values = list(range(ping_step, max_pings + 1, ping_step))
for pings in ping_values:
    delay = time_step
    while delay <= 180:
        full_cold_reset()
        setup_target_grid(rows, cols)
        time.sleep(delay)
        success = count_success(pings)
        print(f"   {delay:>3}s ⟶ {success:>4} ok | {pings-success:>4} lost | {success/pings*100:6.1f}%")
        save_measurement(pings, delay, success/pings*100)
        if success == pings: break
        delay += time_step
```

Investigates scalability of batman-adv: how convergence time grows with network size (from 2×2 up to $n \times n$ nodes) at constant traffic intensity:

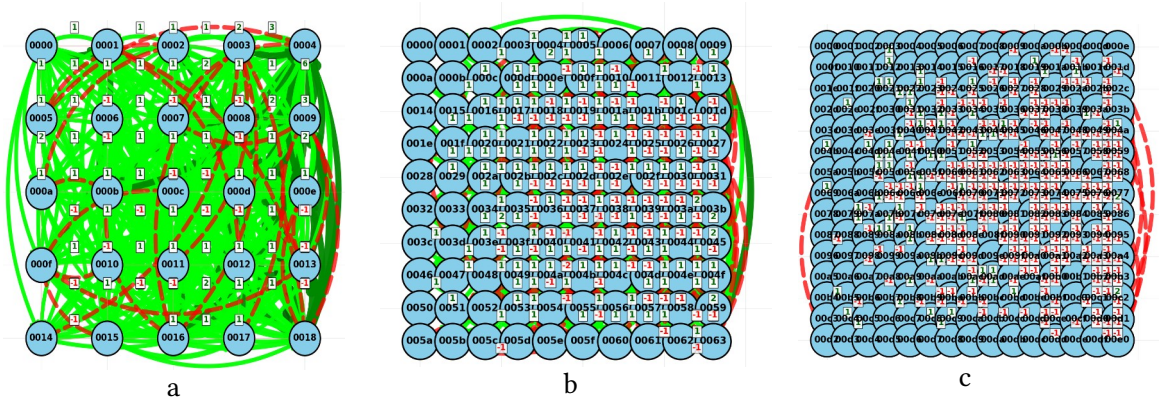
- Full cold reset before every measurement of every size.
- One curve per grid size, viridis colour map.
- Square marker indicates the exact moment of 100% convergence.
- Complete table with every single data point printed before the figure.

```
Pythonfor size in range(2, max_size + 1):
    delay = step_seconds
    while delay <= 180:
        full_cold_reset()
        print(f"   Creating {size}×{size} grid and waiting {delay}s...")
        setup_target_grid(size, size)
        time.sleep(delay)
        success = count_success(num_pings)
        print(f"   {delay:>3}s ⟶ {success:>3} ok | {num_pings-success:>3} lost |
{success/num_pings*100:6.1f}%")
        save_point(size, delay, success/num_pings*100)
        if success == num_pings:
            plt.plot(delay, 100, 's', markersize=14, markeredgecolor='black')
            break
        delay += step_seconds
```

4.4. Simulation Results for BATMAN-adv

The experiment consisted of first initializing a virtual network using the actual L2 mesh interface. Inside each namespace, a *bat0* interface is created and all veth ends that represent radio links are added to it with batctl if add. BATMAN-adv then builds the mesh routing table over these virtual "radios."

To display the network and connections between nodes, a visualization is used that shows the results of ICMP packet transmission. Figure 1 shows an example of different networks. Nodes are selected randomly and the number of successful transmissions is counted. For each network at the same time, initialization accounts for a different number of errors: 24% for 5×5 network, 28 for 10×10, and 56% for 15×15. The numbers on the lines indicate the number of successful and unsuccessful transmissions, which are initialized in random order between different nodes of the network.



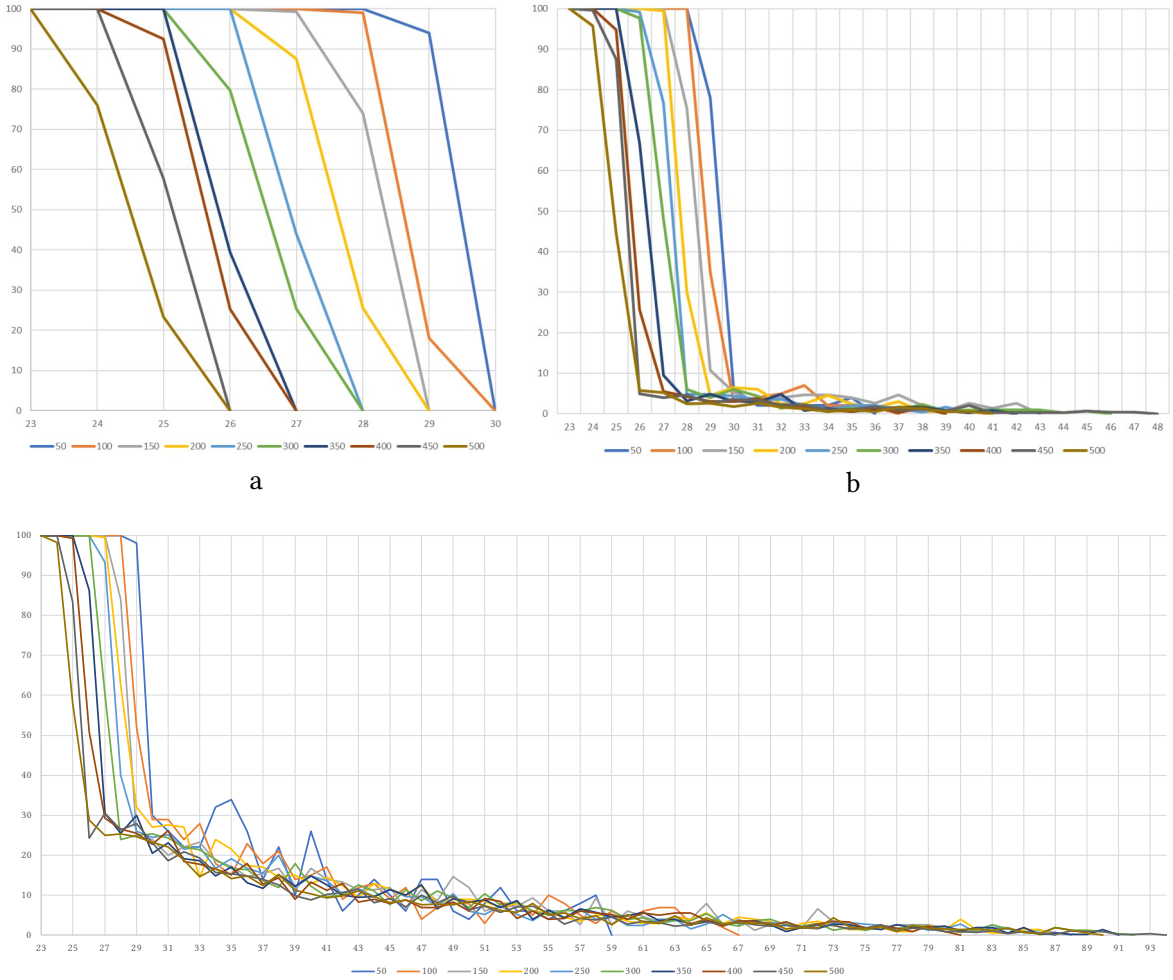
a b c

Figure 1: Examples of networks of different sizes 5×5 (a), 10×10 (b), and 15×15 (c) with a delay of 25 seconds and with the forwarding of 500 test packets

For different networks, the speed of their stabilization is different, which can be seen from the graphs in Figure 2.

Since the network sizes vary, the simulation time also varies and increases exponentially with increasing network size. In the first stage, the previous virtual network is reset to clear node caches and their OGM tables (effectively deleting connections in the network). In the second stage, a new network is initialized, which requires time to populate the OGM tables. However, for the experiment, the network was not allowed to fully initialize; the process of transmitting ICMP packets began. The number of successful packet transmissions is recorded. Afterwards, the final

number of errors is calculated depending on the network size and the number of transmitted text packets.



a b c

Figure 2: Stabilization graphs for networks of dimensions 5×5 (a), 10×10 (b), and 15×15 (c)

An experiment was also conducted on the stabilization speed of networks of different sizes under a load of 500 packets, the results of which are shown in Figure 3.

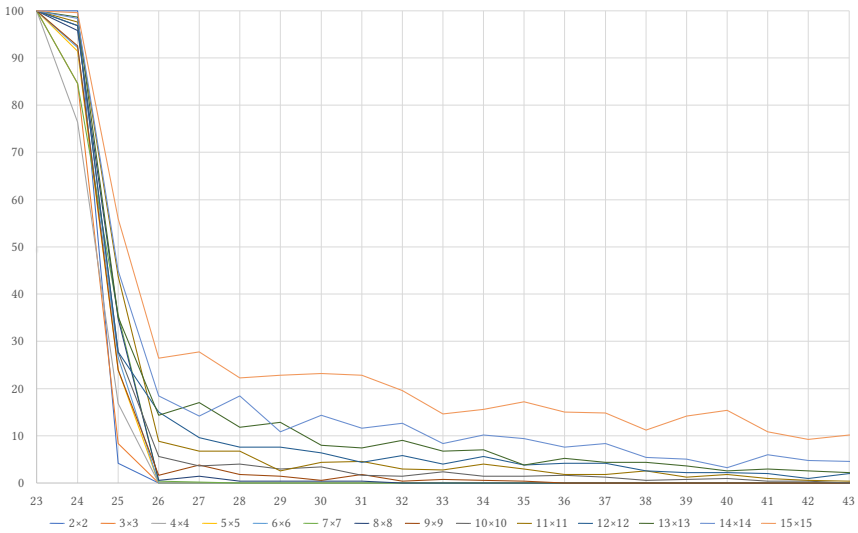


Figure 3: Mesh network stabilization speed depending on network size

The simulation was performed using Linux Ubuntu 24.04.2 LTS on an Intel Core i9-13900H processor with 20 cores and 24 GB of RAM. Figure 4 shows an example of the system load; when messages are sent, the processor load increases.

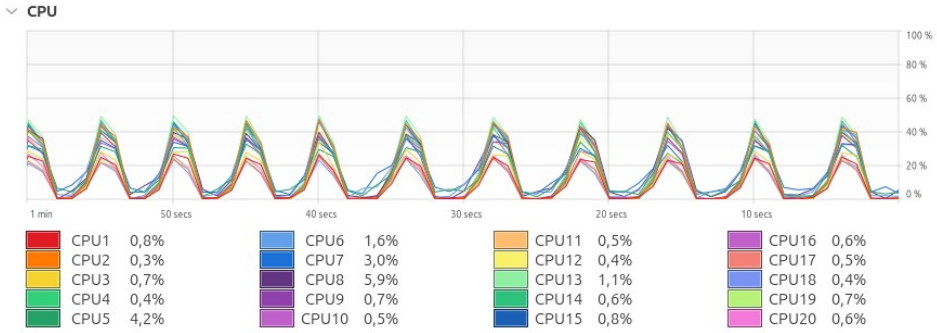


Figure 4: CPU load during simulation

The time diagram shows that in the experiment, the time intervals between processor load peaks increase. Since the processors aren't fully utilized, the only bottleneck affecting the error rate is the network initialization speed. Distributed computing systems can be used to model larger networks.

4.5. Approximation of Measurement Results

The mathematical formula for the ratio of the number of errors to the network initialization time used for the approximation is:

$$E(t_{\text{init}}) = \frac{100}{\left(1 + \left(\frac{t_{\text{init}}}{C}\right)^{B}\right)^{E}} \tag{8}$$

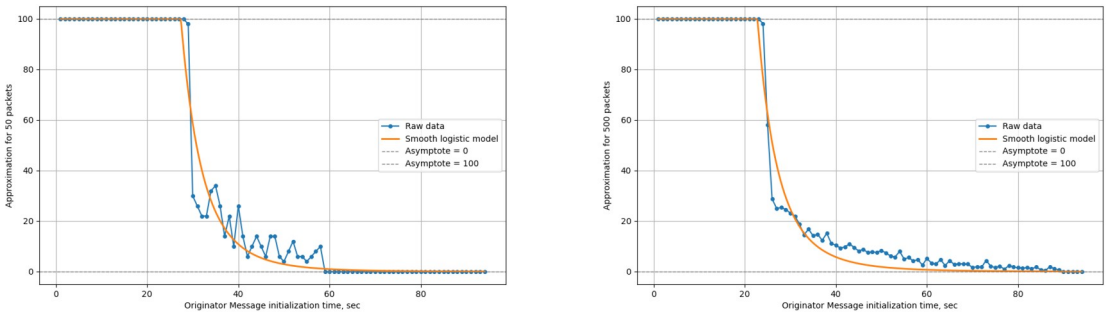
where t_{init} is the input (originator message initialization time in seconds), C, B, and E are fitted parameters that control the shape of the curve (e.g., the inflection point, steepness, and asymmetry). A formula was chosen that does not perfectly repeat the shape of the curve, because when the matrix dimension increases, the second break on the graph degenerates.

For approximation, the *numpy* and *scipy.optimize* packages from Python v. 3.12.3 were used:

```
# --- Improved model with asymmetry to reduce deviations ---
def model(x, C, B, E):
    return 100 / (1 + (x / C)**B)**E

for col in packet_sizes:
    # --- Extract data ---
    y = pd.to_numeric(df[col].astype(str).str.replace(',', '.'), errors='coerce').dropna().values
    x = np.arange(1, len(y) + 1)

    # --- Initial guess ---
    C0 = 28
    B0 = 13
    E0 = 1
    p0 = [C0, B0, E0]

    # --- Fit curve ---
    try:
        popt, pcov = curve_fit(model, x, y, p0=p0, maxfev=100000)
        C_fit, B_fit, E_fit = popt
        y_fit = model(x, *popt)
        mse = np.mean((y - y_fit)**2)
```

```
        results.append({
            'Packets': col,
            'C': C_fit,
            'B': B_fit,
            'E': E_fit,
            'MSE': mse
        })
    except Exception as e:
        results.append({
            'Packets': col,
            'C': np.nan,
            'B': np.nan,
            'E': np.nan,
            'MSE': np.nan
        })
```

After approximation, we obtain the curve shown in Figure 5, which shows that two asymptotes at 100 and 0 are used, as well as two transitions: a sharp drop in the number of errors, and then a smooth decrease to zero. The shape of the curve degenerates into a hyperbola as the dimension increases.



a b

Figure 5: Approximation results for 50 (a) and 500 (b) packets for a 15×15 network

As an example (see Table 1), the values of the approximating coefficients are given for the same network with dimensions 15×15 but with a different number of transmitted packets. The shape of the curves changes little during modulation, therefore the approximating coefficients also remain within approximately the same limits.

Table 1
Determination of approximated parameters for the 15×15 dimension

Packets	C	B	E	Mean Squared Error
50	27.4	521.5	0.0112	**39.6**
100	26.8	786.3	0.0076	25.4
150	26.2	529.7	0.0118	**37.1**
200	25.8	589.7	0.0093	23.7
250	25.5	584.1	0.0100	27.3
300	24.8	545.6	0.0101	29.1
350	24.3	431.0	0.0130	29.7
400	23.7	583.9	0.0091	25.2
450	23.1	509.2	0.0100	27.8
500	22.8	520.5	0.0097	23.9

The mean squared error value for function approximation is also noteworthy, as it is quite high, ranging from 25 to 40 points. As the matrix dimension and the number of transmitted packets increase, this value stabilizes within 23 to 30 points.

5. Discussion

The reduced control overhead at L2 plays an additional role in enhancing protocol stability. BATMAN-adv transmits relatively minor and infrequent control messages, which is advantageous in the context of wireless media that are prone to interference, collisions, and variable throughput. Lower protocol overhead decreases channel contention, leaving more airtime available for data traffic and indirectly improving the predictability and reliability of routing decisions [31, 32]. In highly dynamic or noise-prone environments, minimizing control traffic is essential because excessive signaling can exacerbate congestion and amplify instability within the mesh [33].

Taken together, these characteristics explain why a L2 mesh protocol such as BATMAN-adv tends to produce a more stable operational environment than its L3 counterparts. Its reliance on localized information, the absence of global routing table maintenance, the preservation of native Ethernet behavior, and the reduction of control plane overhead collectively yield a forwarding architecture that is inherently more resilient to the volatility inherent in wireless mesh networks. This stability emerges not from any single design decision, but from the cumulative effect of adopting link-layer principles in place of traditional network-layer routing mechanisms, thereby aligning the protocol more closely with the realities of dynamic, distributed wireless communication.

Of course, this simulation only partially reflects the operation of a real wireless network, but it allows us to evaluate the mechanics of such a network. It should also be noted that the number of packets sent does not significantly affect the shape of the dependence, merely “shifting” the graph to the right, which is due solely to a proportional increase in the number of errors. This is evidenced by the shape of the attenuation curve.

Since network operation in real-world conditions is dynamic, the simulation during the first “cold” initialization will represent the most extreme network operation scenario. Even if some network nodes are unavailable at a given time or local loops occur, the network will eventually reconfigure and restore functionality.

6. Conclusions

This thesis investigates how quickly Layer 2 mesh networks become operational after a cold start, focusing on BATMAN-adv as a representative L2 forwarding protocol. While mesh networks are widely used for distributed, fault-tolerant communication, their initialization dynamics are still poorly characterized, especially at the data-link layer where no global routing tables are maintained. BATMAN-adv relies on periodic OGMs with relative sequence numbers to build forwarding paths without requiring time synchronization, reducing sensitivity to mobility and radio variability.

A dedicated simulation framework was developed to evaluate initialization speed under realistic conditions. The environment constructs virtual mesh topologies using Linux namespaces and BATMAN-adv interfaces, performs automated cold resets, measures packet delivery success, and generates high-quality traffic visualizations. The model allows systematic variation of topology size, OGM intervals, packet load, and network density. Experiments conducted on grids from 5×5 to 15×15 demonstrate that initialization time increases with network diameter and becomes more variable under heavy load. Empirically measured convergence times for larger networks typically fall between 20 and 45 seconds, consistent with theoretical predictions derived from multi-hop OGM propagation.

To quantify the relationship between initialization delay and packet error rate, an asymmetric three-parameter fitting function was applied using SciPy optimization tools. The function successfully reproduced the two-phase decay observed in measurements—an initial sharp drop in errors followed by gradual stabilization—while maintaining relatively stable parameter values across packet loads. Although simulation cannot fully reproduce the behavior of real wireless hardware, the results provide meaningful insights into link-layer convergence mechanisms and demonstrate that packet volume primarily shifts error curves rather than changing their structure.

The study concludes that BATMAN-adv offers inherently stable initialization behavior due to low control overhead, localized decision-making, and avoidance of global routing state. Future work includes validating the simulation results on physical wireless devices to assess the influence of real-world radio effects.

Further plans are underway to repeat this experiment on real wireless devices. It is also planned to test how the shape of the network affects its convergence time, for example, rectangular or circular.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] V. Sokolov, P. Skladannyi, H. Hulak, Stability Verification of Self-Organized Wireless Networks with Block Encryption, in: Computer Modeling and Intelligent Systems, vol. 3137 (2022) 227–237. doi:10.32782/cmis/3137-19

[2] I. F. Akyildiz, X. Wang, W. Wang, Wireless Mesh Networks: A Survey. Computer Networks, 47(4) (2005) 445–487. doi:10.1016/j.comnet.2004.12.001

[3] R. Bruno, M. Conti, E. Gregori, Mesh Networks: Commodity Multihop Ad Hoc Networks, IEEE Communications Magazine, 43(3) (2005) 123–131. doi:10.1109/MCOM.2005.1404606

[4] T. Camp, J. Boleng, V. Davies, A Survey of Mobility Models for Ad Hoc Network Research, Wireless Communications and Mobile Computing, 2(5) (2002) 483–502. doi:10.1002/wcm.72

[5] P. H. Pathak, R. Dutta, A Survey of Network Design Problems and Joint Design Approaches in Wireless Mesh Networks, IEEE Communications Surveys & Tutorials, 13(3) (2011) 396–428. doi:10.1109/SURV.2011.060710.00062

[6] M. Singh, V. Talasila, A Practical Evaluation for Routing Performance of BATMAN-ADV and HWMN in a Wireless Mesh Network Test-Bed, in: 2015 Int. Conf. on Smart Sensors and Systems (IC-SSS), 2015, 1–6. doi:10.1109/smartsens.2015.7873617

[7] N. Anas, et al. Performance Analysis of Outdoor Wireless Mesh Network using B.A.T.M.A.N. Advanced, in: 2015 IEEE/ACIS 16th Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015, 1–4. doi:10.1109/snpd.2015.7176189

[8] D. Seither, A. König, M. Hollick, Routing Performance of Wireless Mesh Networks: A Practical Evaluation of BATMAN Advanced, in: 2011 IEEE 36th Conf. on Local Computer Networks, 2011, 897–904. doi:10.1109/lcn.2011.6115569

[9] J. Lin, Z. Liu, J. Dai, Research and Optimization of Wireless Mesh Network Routing Protocol with Multiple Criterions, in: 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conf. (IAEAC), 1, 2019, 72–78. doi:10.1109/iaeac47372.2019.8997932

[10] J. Lin, J. Dai, Improvement of Routing Protocol for Wireless Mesh Network based on Packet Aggregation, in: 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conf. (IMCEC), 2019, 175–179. doi:10.1109/imcec46724.2019.8984063

[11] M. Thomas, M. Tham, Y. Wong, Y. Chang, Performance Evaluation of Video Streaming in Wireless Mesh Networks, in: 2024 IEEE 12th Conf. on Systems, Process & Control (ICSPC), 2024, 390–394. doi:10.1109/icspc63060.2024.10862838

[12] B. Diniz, I. Ferrã, L. Da Silva, K. Branco, Comparative Performance Analysis of Olsr, Batman-Adv, and Babel in UAV Mesh Networks, in: 2025 Int. Conf. on Unmanned Aircraft Systems (ICUAS), 2025, 496–503. doi:10.1109/icuas65942.2025.11007820

[13] L. Liu, J. Liu, H. Qian, J. Zhu, Performance Evaluation of BATMAN-Adv Wireless Mesh Network Routing Algorithms, in: 2018 5th IEEE Int. Conf. on Cyber Security and Cloud Computing (CSCloud) / 2018 4th IEEE Int. Conf. on Edge Computing and Scalable Cloud (EdgeCom), 2018, 122–127. doi:10.1109/cscloud/edgecom.2018.00030

[14] M. Kadadha, et al., A Cluster-based Quality-of-Service Optimized Link State Routing Protocol for Mesh Networks, in: 2022 Int. Wireless Communications and Mobile Computing (IWCMC), 2022, 336–341. doi:10.1109/iwcmc55113.2022.9824333

[15] S. Gilani, A. Qayyum, R. Rais, M. Bano, SDNMesh: An SDN Based Routing Architecture for Wireless Mesh Networks, IEEE Access, 8 (2020) 136769–136781. doi:10.1109/access.2020.3011651
[16] I. Yilmaz, et al., Evaluation of End-to-End Connection Recovery After Traffic Path Disruption in BATMAN Mesh Networks, in: 2025 IEEE Wireless Communications and Networking Conf. (WCNC), 2025, 1–5. doi:10.1109/wcnc61545.2025.10978701
[17] H. Yuliandoko, et al., Performance of Implementation IBR-DTN and Batman-Adv Routing Protocol in Wireless Mesh Networks, Int. J. Eng. Technol., 3 (2016). doi:10.24003/emitter.v3i1.32
[18] M. Bano, A. Qayyum, R. Rais, S. Gilani, Soft-Mesh: A Robust Routing Architecture for Hybrid SDN and Wireless Mesh Networks, IEEE Access, 9 (2021) 87715–87730. doi:10.1109/access.2021.3089020
[19] J. Britos, et al., BATMAN Adv. Mesh Network Emulator, 2015. https://sedici.unlp.edu.ar/handle/10915/50450
[20] Z. Wang, T. Yong, X. Song, Fast and Low-Overhead Time Synchronization for Industrial Wireless Sensor Networks with Mesh-Star Architecture, Sensors, 23 (2023). doi:10.3390/s23083792
[21] Z. Xu, S. Garimella, V. Tzoumas, Communication- and Computation-Efficient Distributed Submodular Optimization in Robot Mesh Networks, IEEE Transactions on Robotics, 41 (2024) 3480–3499. doi:10.1109/tro.2025.3567540
[22] A. Totounferoush, F. Simonis,, B. Uekermann, M. Schulte, Efficient and Scalable Initialization of Partitioned Coupled Simulations with preCICE, Algorithms, 14 (2021) 166. doi:10.3390/a14060166
[23] D. Ghosh, et al., Utilizing Continuous Time Markov Chain for Analyzing Video-on-demand Streaming in Multimedia Systems, Expert Syst. Appl., 223 (2023) 119857. doi:10.1016/j.eswa.2023.119857
[24] N. Barwar, B. Rajesh, Network Performance Analysis of Startup Buffering for Live Streaming in P2P VOD Systems for Mesh-Based Topology, 2016, 271–279. doi:10.1007/978-981-10-0755-2_29
[25] A. Neumann, C. Aichele, M. Lindner, M. Wählisch, Better Approach to Mobile Ad-Hoc Networking (B.A.T.M.A.N.) (Internet-Draft draft-openmesh-b-a-t-m-a-n-00). Internet Engineering Task Force, 2008. https://datatracker.ietf.org/doc/html/draft-openmesh-b-a-t-m-a-n-00
[26] B. Sliwa, S. Falten, C. Wietfeld, Performance Evaluation and Optimization of B.A.T.M.A.N. V Routing for Aerial and Ground-Based Mobile Ad-Hoc Networks, in: 2019 IEEE 89th Vehicular Technology Conf. (VTC2019-Spring), 2019, 1–7. doi:10.1109/vtcspring.2019.8746361
[27] Mesh Network Lab, 2025. https://github.com/mwarning/meshnet-lab
[28] Batman-adv, 2025. http://github.com/open-mesh-mirror/batman-adv
[29] Komondor: An IEEE 802.11ax Simulator, 2025. https://github.com/wn-upf/Komondor
[30] OMNETpp-batmanV, 2020. https://github.com/drblah/OMNETpp-batmanV
[31] V. Sokolov, Y. Kostiuk, P. Skladannyi, N. Korshun, Adaptation of Network Traffic Routing Policy to Information Security and Network Protection Requirements, in: Information Control Systems & Technologies (ICST), vol. 4048, 2025, 397–411.
[32] R. Syrotynskyi, et al., Methodology of Network Infrastructure Analysis as Part of Migration to Zero-Trust Architecture, in: Cyber Security and Data Protection (CSDP), vol. 3800, 2024, 97–105.
[33] P. Skladannyi, et al., Model and Methodology for the Formation of Adaptive Security Profiles for the Protection of Wireless Networks in the Face of Dynamic Cyber Threats, in: Cyber Security and Data Protection (CSDP), vol. 4042, 2025, 17–36.