# Optimizing Multicast in MaritimeManet: Diagnosing Communication Failures and Evaluating BATMAN-adv Optimizations

JORIM HEBBINK, University of Twente, The Netherlands

MaritimeManet is a Mobile Ad-hoc Network (MANET) using directional antennas for full 360-degree coverage, creating a long-range IEEE 802.11s mesh network for maritime environments. This study investigates the effectiveness of multicast video streaming over MaritimeManet using optimizations in B.A.T.M.A.N. advanced (BATMAN-adv). Communication issues in the digital twin environment, notably VLAN-related packet loss, were resolved to ensure reliable testing. Three multicast optimizations in BATMAN-adv, dedicated multicast packets, multicast as unicast, and multicast as broadcast, were evaluated under varying network conditions. Results show that broadcast yields artificially high throughput due to simulation artifacts, while multicast as unicast suffers from uneven performance at higher loads. The dedicated multicast packet mechanism delivers the most stable performance over all the tested network conditions, making it the preferred option for multicast video streaming in MaritimeManet.

Additional Key Words and Phrases: MaritimeManet, MANET, BATMAN-adv, Multicast optimization, Digital Twin, Multicast Video Streaming

## 1 INTRODUCTION

Ship-to-ship communication has evolved from semaphore flags to modern satellite systems, yet affordable, high-bandwidth ship-to-ship connectivity remains a challenge. While very high frequency (VHF) radio is widely used, it only offers low bandwidth [1]. Other wireless technologies like LTE depend on shore-based infrastructure, limiting their range, while satellite networks, although global, suffer from high costs, high latency, and potential security risks due to third-party involvement [1]. These limitations underscore the need for robust ship-to-ship solutions capable of supporting bandwidth-intensive applications such as video streaming.

Mobile Ad-hoc Networks (MANETs) provide a promising alternative, as they require no fixed infrastructure and can adapt to changing topologies [13]. Thales Netherlands developed MaritimeManet, an experimental MANET based on IEEE 802.11s mesh networking [3, 6]. Unlike conventional mesh networks with omnidirectional antennas, MaritimeManet uses multi-beam antennas (MBAs) arranged in a sunflower pattern, enabling directional communication in a 360-degree radius for increased range and bandwidth [4].

To handle the dynamic nature of such networks, MaritimeManet uses the Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.) protocol, specifically its B.A.T.M.A.N. advanced (BATMAN-adv) implementation [11, 14]. Operating at the MAC layer rather than the IP layer, BATMAN-adv enables more efficient routing and reduced overhead, which is crucial for rapidly changing network topologies.

Physical testing of MaritimeManet has proven expensive and time-consuming [4]. To address this, a Digital Twin was developed using virtual machines (VMs) to emulate network behaviour and facilitate scalable testing.

A key application for MaritimeManet is live video streaming from a few nodes to many receivers. Because video streaming consumes significant bandwidth, multicast is proposed as a more efficient alternative to unicast or broadcast. Through the Internet Group Management Protocol (IGMP) [2], multicast enables data delivery only to interested receivers, reducing unnecessary network load [19].

This research investigates the available multicast optimizations within BATMAN-adv to address a gap in understanding their performance in MaritimeManet. By evaluating these mechanisms, the work aims to improve video streaming and overall network efficiency, contributing to enhancing the effectiveness and reliability of MaritimeManet.

## 2 RELATED WORK

Efficient maritime mesh networks are increasingly important for applications such as video streaming and data sharing between ships [1]. BATMAN-adv has been widely studied for routing efficiency and mesh network performance [8, 17], and video streaming over BATMAN-based networks has been evaluated for unicast scenarios [16].

While multicast communication has been extensively analysed in wireless mesh networks [5, 7], little research has focused on integrating multicast with BATMAN-adv, particularly in maritime contexts. This work aims to bridge that gap by evaluating BATMAN-adv's multicast optimizations in the Digital Twin of MaritimeManet.

### 2.1 Digital Twin for MaritimeManet

MaritimeManet has been the focus of several research projects, leading to the development of a Digital Twin to simulate its network behaviour. The Distributed Simulator for MaritimeManet (DSM) generates network topologies and node connections, which the BATMAN Topology Configurator (BTC) translates into configurations for the Network Controller (NC) [12]. Initially implemented on a physical testbed, the system now operates entirely virtually in Proxmox for improved scalability [4].

The virtual environment contains the Simulator VM, comprised of the DSM, BTC, and a controller that automates virtual machine creation. Each simulated node consists of two VMs: a Debian-based Execution Environment (EE) for running applications and an OpenWRT BATMAN node for mesh connectivity via BATMAN-adv. An architectural overview is shown in Fig. 1.

EE and BATMAN VMs connect through a Proxmox Linux bridge (Proxmox LAN Bridge) using IEEE 802.1q VLANs for intra-node isolation. The BATMAN Mesh network uses a separate bridge (Proxmox Mesh Bridge) with IEEE 802.1ad (Q-in-Q) VLAN tagging to segregate node traffic. This ensures all inter-node communication routes through the NC, where the outer VLAN tag identifies the

*TScIT 43, July 4, 2025, Enschede, The Netherlands*

node and the inner tag distinguishes MBAs. Fig. 2 details the network configuration for two nodes.

Because the bridged setup relies on Ethernet while MaritimeManet uses Wi-Fi, it initially fails to reflect realistic wireless conditions. To compensate, a `qdisc` network emulator applies delay (1 ms), packet loss (0.2%), and DSM-calculated bandwidth limits on each Q-in-Q interface in the NC for more accurate network behaviour.

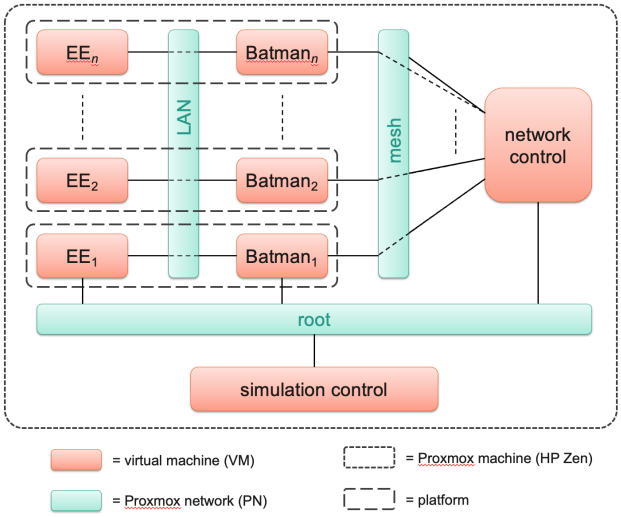


Fig. 1. Overview of the Digital Twin architecture

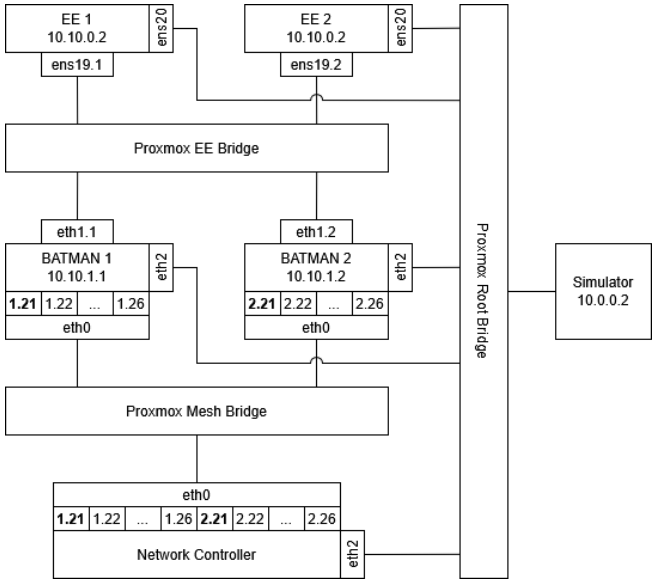


Fig. 2. Detailed network topology for two nodes

Despite the Digital Twin's flexibility, a critical issue persists: although BATMAN nodes discover each other and populate neighbour tables, actual data transmission fails, preventing reliable application-level testing until resolved.

## 3 PROBLEM STATEMENT

Effective multicast communication is essential for enhancing ship-to-ship interactions in maritime environments, particularly for bandwidth-intensive applications such as video streaming. This research aims to improve the multicast capabilities of MaritimeManet by evaluating the multicast optimizations offered by BATMAN-adv. These optimizations will be assessed in terms of throughput and packet loss using a multicast application deployed across various nodes and topologies within the Digital Twin environment.

A significant challenge in the Digital Twin implementation arises from a communication issues between BATMAN nodes. These issues interfere with packet forwarding and distort multicast performance metrics, undermining the reliability of throughput and packet loss measurements.

This leads to the following research question:

> *How can the configuration and optimization of B.A.T.M.A.N. nodes and network setup within the Digital Twin improve multicast video streaming performance in MaritimeManet, particularly with respect to bandwidth and packet loss?*

To answer this research question, the following sub-research questions are formulated:

(1) What causes the communication issues between BATMAN nodes in the Digital Twin of MaritimeManet, and how can these be resolved to ensure reliable data transmission?
(2) What multicast optimizations are available in BATMAN-adv, and how can they be effectively implemented and tested within the Digital Twin of MaritimeManet?
(3) Which multicast optimization provides the best performance for MaritimeManet in terms of bandwidth usage and packet loss, and how do these results compare to each other?

## 4 METHODOLOGY

To address the main research question, this study is structured around three core investigative steps. First, it examines the communication issues present in the Digital Twin of MaritimeManet. Next, it explores available multicast optimizations in the B.A.T.M.A.N. advanced protocol and how these can be integrated into the simulation. Finally, it outlines the approach for evaluating multicast performance through structured testing across various simulated network conditions. The following sections describe each of these steps in detail.

## 5 DIAGNOSING AND RESOLVING NODE COMMUNICATION ISSUES

To resolve the unreliable network connections, we first needed insight into the actual behaviour of the network. A static simulation was created in which the nodes remained fixed in position and orientation. This simulation included two nodes connected by a single link. The nodes configured in the simulation are shown in Fig. 3, and, an abstraction of, the expected network topology is illustrated in Fig. 2.

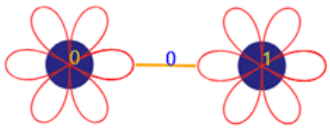


Fig. 3. Connected nodes in the simulator

In the expected topology, the two BATMAN nodes, BATMAN_1 and BATMAN_2, establish a link over their respective hard interfaces, `eth0.1.21` and `eth0.2.21`. [1] In the Digital Twin, these hard interfaces correspond to VLAN interfaces as illustrated in Fig. 2.

Due to the complexity of the simulated network, the validation process was structured into three key stages to systematically verify the different components:

(1) Mesh connectivity between BATMAN_1 and BATMAN_2.
(2) Local connectivity between each BATMAN node and its corresponding EE.
(3) End-to-end communication from EE_1 to EE_2, validating the entire network path.

## 5.1 BATMAN Mesh Connectivity

Once the simulation was running, the BATMAN nodes were accessed via SSH from the simulator. Using the `batctl neighbour` command provided by BATMAN-adv[2], we verified that both nodes successfully discovered and established connectivity with each other through the expected hard interfaces: `eth0.1.21` for BATMAN_1 and `eth0.2.21` for BATMAN_2.

To further analyse the behaviour of the connection between the BATMAN nodes, we captured traffic transmitted over the BATMAN mesh. This was achieved by placing `tcpdump` probes on the relevant hard interfaces (`eth0.1.21` and `eth0.2.21`), generating packet capture (PCAP) files for analysis in Wireshark.[3]

Since PCAP files were generated on the BATMAN nodes, they were transferred to the Simulator via Secure Copy (SCP). To automate this, a custom shell script was created to start a `tcpdump` capture on a specified interface and automatically transfer the PCAP file upon completion. The script, provided in Appendix E, was used consistently throughout all experiments involving `tcpdump` probes.

Analysis of the captures in Wireshark confirmed that both nodes were transmitting originator messages (OGM) and Echo Location Protocol (ELP) messages[4], validating that network discovery between the BATMAN nodes was functioning as expected.

*Mesh Connectivity Testing with ICMP Traffic.* To further evaluate mesh behaviour, basic IP connectivity between the BATMAN nodes was tested using the `ping` command. As in previous tests, `tcpdump` probes were placed on the hard interfaces (`eth0.1.21` and `eth0.2.21`) to capture packet-level traffic.

From the simulator, an SSH session was opened to BATMAN_1, and a `ping` was initiated to `10.10.1.2`, the IP address of BATMAN_2. After approximately one minute, the ping and both capture probes were stopped, and the resulting PCAP files were transferred to the simulator for analysis in Wireshark.

Under normal conditions, BATMAN_1 should broadcast an ARP request for `10.10.1.2`, which is received by BATMAN_2, triggering an ARP reply. Once received, BATMAN_1 would begin sending ICMP echo requests and receive replies in response.

However, no ping response was observed and 100% packet loss was reported. Wireshark confirmed that BATMAN_1 transmitted the ARP request, which BATMAN_2 received and replied to, but the ARP reply never arrived at BATMAN_1's interface. Consequently, BATMAN_1 continued retransmitting the ARP request, which BATMAN_2 repeatedly answered, forming a loop of unanswered ARP communication.

*Diagnosing and Resolving Packet Loss.* The previous observations indicated that the ARP reply was lost between the BATMAN hard interfaces, suggesting the Network Controller as the likely point of failure. Since the Proxmox Mesh bridge, responsible for relaying BATMAN mesh traffic between the BATMAN VMs and the Network Controller, operates at Layer 2, it should forward packets based solely on MAC addresses, without filtering or modification.

To investigate, `tcpdump` probes were placed directly on the `eth0.1.21` and `eth0.2.21` interfaces of the Network Controller during a repeat ping test from BATMAN_1 to `10.10.1.2`. Wireshark analysis of the resulting PCAP files showed that the ARP reply entered the Network Controller on `eth0.2.21` and exited on `eth0.1.21`, appearing to confirm correct Layer 2 forwarding. However, this contradicted the earlier observation that the reply never arrived at BATMAN_1, casting doubt on the assumption that the Linux bridge was behaving transparently.

Given that both BATMAN nodes and the Network Controller are virtual machines hosted on Proxmox using the KVM hypervisor, and are connected via TAP interfaces to the shared bridge as illustrated in Fig. 4, it became more likely that packet loss was occurring at the TAP interface level. Supporting this hypothesis, *Chapter 16* of [15] notes that VLAN-tagged traffic from KVM guests can be exposed via TAP interfaces on the host, where it is selectively accepted and forwarded based on VLAN tags.

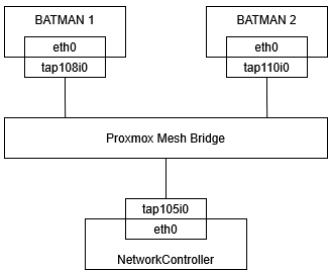


Fig. 4. Visualization of the TAP interfaces within Proxmox

[1] BATMAN-adv uses virtual interfaces called `bat0` which act like distributed switch ports. The actual network traffic is handled by underlying physical or virtual interfaces known as *hard interfaces*, which connect to the mesh [9].
[2] `batctl` is a tool to configure and debug the BATMAN-adv kernel module. It offers an interface to all the module's settings as well as status information, such as the neighbour and originator tables [9].
[3] `tcpdump` and Wireshark are widely used tools for capturing and analysing network traffic. They allow detailed inspection of packet-level behaviour and are essential for debugging low-level communication issues [18].
[4] OGM and ELP messages are internal control messages used by the BATMAN-adv routing protocol to discover and maintain mesh topology.

Upon verifying that the TAP interfaces were not VLAN-tagged, a new configuration was applied: the outer VLAN tag of each BATMAN node was explicitly added to its respective TAP interface, and the Network Controller's TAP interface was set to accept all outer VLAN tags. This was achieved using the following command:

```
bridge vlan add dev {tap_interface} vid {vlan}
```

After this reconfiguration, the ping test between BATMAN_1 and BATMAN_2 was repeated with tcpdump probes on eth0.1.21 and eth0.2.21. This time, 100% of the ping packets were successfully delivered, as confirmed both by the ping utility and Wireshark analysis. The ARP reply was now correctly received at eth0.1.21, allowing BATMAN_1 to initiate ICMP echo requests, which were answered by BATMAN_2.

However, this still did not explain why the ARP reply was consistently lost only during the final hop from the Network Controller to BATMAN_1, nor why broadcast messages were always correctly received, even though all packets were VLAN-tagged. This anomaly remained unresolved and warrants further investigation.

### 5.2 Local BATMAN-to-EE Connectivity

After confirming that the BATMAN mesh connectivity was functioning correctly, tests were conducted to verify connectivity between each BATMAN node and its corresponding EE over their local VLAN-based link on the Proxmox LAN Bridge. Given the similarity to the BATMAN mesh setup, it was suspected that the TAP interfaces on the EE and BATMAN nodes might also require explicit VLAN tagging.

To test this hypothesis, tcpdump probes were placed on the ens19.1 interface of EE_1 and the eth1.1 interface of BATMAN_1. A ping was then initiated from BATMAN_1 to 10.10.0.1, the IP address assigned to EE_1. As anticipated, the ping resulted in 100% packet loss. PCAP analysis in Wireshark confirmed packet loss consistent with the earlier mesh connectivity issue.

To resolve this, the correct VLAN tags were assigned to the TAP interfaces of all the EE and BATMAN VMs. When the test was repeated under identical conditions, the ping achieved 100% successful delivery, confirming that the packet loss had been caused by missing VLAN tags on the TAP interfaces.

### 5.3 End-to-End EE Connectivity

After resolving the VLAN-related issues in both the BATMAN mesh and the local BATMAN-to-EE links, a final test was performed to validate end-to-end communication between the EEs.

For this test, tcpdump probes were placed on the ens19.1 interface of EE_1 and the ens19.2 interface of EE_2. A ping was initiated from EE_1 to 10.10.0.2, the IP address assigned to EE_2.

As anticipated, the ping was successful with 0% packet loss. PCAP analysis in Wireshark confirmed the bidirectional exchange of ICMP echo requests and replies, demonstrating that end-to-end connectivity through the entire virtual network—including the BATMAN nodes and the Network Controller—was functioning correctly.

This outcome validated the effectiveness of the VLAN tagging on the TAP interfaces and confirmed that all critical network paths in the simulation were operating as intended.

### 5.4 Automating Network Configurations

To eliminate manual VLAN configuration for each simulation, the NetworkController class, located in the Controller of the Simulator, was extended to automatically apply correct VLAN settings during each simulation reload, ensuring consistent connectivity across nodes.

During this process, existing VLAN tags on TAP interfaces are cleared to avoid unintended traffic leaks. New tags are then assigned based on node roles: TAP interfaces connected to the EE bridge receive VLAN tags matching the node's unique ID, isolating each EE and its corresponding BATMAN node. For the Mesh bridge, BATMAN nodes are similarly tagged with their node IDs, while the Network Controller's TAP interface is assigned all VLAN IDs to enable mesh-wide communication. These tags are set by establishing an SSH connection to the Proxmox host and executing the same bridge vlan command described in Section 5.1.

This automated tagging, managed entirely by the Simulation Controller, allows the network to be reliably reloaded and scaled without manual intervention.

## 6 MULTICAST OPTIMIZATION IDENTIFICATIONS AND INTEGRATION

Multicast forwarding optimizations in BATMAN-adv were identified based on the official Open-Mesh documentation[10]. BATMAN-adv supports the following optimizations for multicast traffic:

(1) Encapsulation of multicast packets using a dedicated BATMAN-adv multicast packet type,
(2) Replication of multicast traffic to individual recipients via unicast transmissions,
(3) Flooding of multicast traffic through broadcast as a fallback strategy.

BATMAN-adv internally determines the multicast strategy based on the decision tree in Fig. 5 and only exposes a single configuration option, multicast_forceflood. When enabled, this setting forces all multicast traffic to be broadcasted, bypassing the optimization methods.

To study the effects of each optimization individually, a more granular control mechanism was necessary. Based on the decision tree, we concluded that BATMAN-adv can be forced to use a specific optimisation.

To enable the multicast packet optimization, all nodes needed to support the multicast optimization and the encapsulated IP packet, including all its destination node entries needed to fit into a 1280 bytes frame (excluding the outer Ethernet frame header).

All BATMAN nodes in our simulator used BATMAN-adv version 2024.3, which automatically enabled support for the multicast optimizations, satisfying the first requirement.

For the second requirement, packets were restricted to limited Ethernet frame sizes. To illustrate these frame sizes the BATMAN-adv documentation has provided size limits (referred to as Ethernet frame size) for various group sizes. These size limits are provided in Table 1.

These values can be derived by subtracting the overhead of various headers from the maximum frame size of 1280 bytes. The overhead includes: 12 bytes for the BATMAN-adv multicast packet

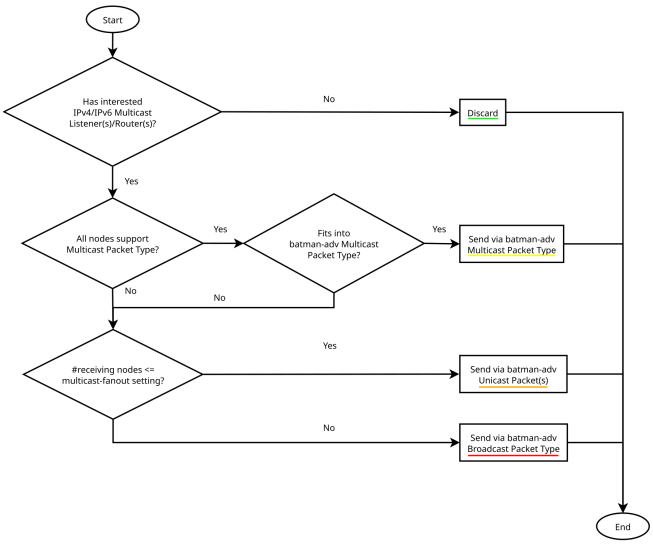


Fig. 5. BATMAN-adv Multicast optimization decision tree [10]

Table 1. Ethernet frame size for the number of destinations

| # Destinations | Ethernet frame size (in Bytes) |
|---|---|
| 2 | 1222 |
| 8 | 1186 |
| 32 | 1030 |
| 128 | 454 |
| 196 | 46 |

header, 6 bytes per destination (MAC address), 14 bytes for the inner Ethernet header, and 20 bytes for the inner IPv4 header.

This yields a general formula:

$$\text{Ethernet frame size} = 1280 - 12 - (6 \cdot n) - 14 - 20 \qquad (1)$$

where $n$ is the number of destination addresses.

To enable multicast packet optimization, packet sizes must remain less than or equal to the value calculated in Formula 1. To ensure consistency across tests, the maximum Ethernet frame size will be determined based on the test scenario with the largest number of listeners. The specific value used will be provided in Section 7.1.

To isolate the unicast optimization, it is necessary to prevent the multicast packet optimization from being triggered. Therefore, the Ethernet frame size must exceed the multicast packet threshold, while remaining small enough to avoid distorting performance results. A frame size of 1234 bytes satisfies these requirements, corresponding to the case where the multicast destination list is empty ($n = 0$). Additionally, the unicast optimization depends on the `multicast_fanout` setting. When the number of destinations is less than or equal to this fanout value, unicast forwarding is used; otherwise, packets are broadcasted. To ensure that the unicast optimization is consistently applied during testing, the `multicast_fanout` must be set to at least the number of listeners in each scenario.

Broadcasting represents the fallback mechanism within the multicast forwarding decision flow. It is utilized when neither the multicast packet nor the unicast optimization is viable or when the `multicast_forceflood` setting is explicitly enabled, which was the method used for testing the broadcast optimization in this study. Although broadcast behaviour will be included in the experimental evaluation, it is important to note that the simulator used in this study operates over Ethernet rather than Wi-Fi. Unlike Wi-Fi, where broadcast transmission is usually performed at lower data rates and may be less efficient for sustained data transfer, Ethernet supports more efficient broadcast delivery. As a result, performance measurements for the broadcast optimization in this simulated environment are expected to be artificially high and may not accurately reflect real-world performance in wireless mesh deployments.

### 6.1 Implementation of the optimizations

The three multicast optimizations supported by BATMAN-adv were tested under controlled conditions by adjusting specific configuration parameters. Table 2 summarizes the settings used for each optimization. In this table, $x$ is the maximum payload size calculated using Formula 1, and $n$ is the number of multicast listeners, which determines both the multicast payload size and the `multicast_fanout` threshold.

Table 2. Configuration for each multicast optimization

| Optimization | forceflood | fanout | payload size |
|---|---|---|---|
| Multicast packet | 0 | $n$ | $x$ |
| Unicast replication | 0 | $n$ | 1234 |
| Broadcast | 1 | 0 | 1234 |

The `forceflood` parameter forces all multicast traffic to be broadcast, bypassing other optimizations, and was set in the BATMAN VMs via:

```
batctl multicast_forceflood 1 # Enable forceflood
batctl multicast_forceflood 0 # Disable forceflood
```

The `multicast_fanout` was set in the BATMAN VMs via:

```
batctl multicast_fanout n
```

Unlike these parameters, the payload size is not directly configurable in BATMAN-adv but is determined by the application-layer on the EE VMs. In this study, payload sizes were chosen at the application layer to trigger the desired forwarding behaviour based on BATMAN-adv's internal logic.

## 7 EVALUATING MULTICAST OPTIMIZATIONS

### 7.1 General Test Plan

To identify the most suitable multicast forwarding approach for MaritimeManet, a structured set of tests will be conducted. These tests measure throughput and reliability across various network conditions and configurations, focusing on BATMAN-adv's three multicast optimizations within the Digital Twin of MaritimeManet.

The performance of each strategy is evaluated by varying several parameters:

(1) **Number of Sessions** — single vs. multiple concurrent multicast sessions.
(2) **Hop Count** — the number of intermediate nodes between sender and receiver.
(3) **Casting Mode** — unicast vs. multicast traffic.
(4) **Topology Type** — static networks, mobile nodes, and changing topologies.
(5) **Number of Receivers** — evaluating scalability under different group sizes.

Details on these parameters, including expectations and reasoning, are provided in Appendix B.

*7.1.1 Test Plan.* Based on these parameters, six primary tests are defined:

(1) **Unicast baseline over multiple hops**
A single unicast session tested over 1 to 4 hops in a static network topology, establishing a performance baseline for multicast comparisons.
(2) **Multicast with a single listener**
Identical to Test 1, but using multicast transmissions with only one receiver.
(3) **Multicast with multiple listeners**
A multicast session spanning 2 hops with 2 or 3 receivers, assessing how performance scales with increasing group size.
(4) **Multicast with multiple senders**
Two simultaneous multicast sessions over 2 hops, each with its own receiver, evaluating network load under concurrent multicast sources.
(5) **Multicast with moving receiver**
Similar to Tests 2 over 2 hops, but with a mobile receiver node that constantly connects to other nodes during the session.
(6) **Multicast with changing topology**
Similar to test 3 spanning 2 hops with 3 receivers, a single node joins or leaves the multicast group during the session, simulating dynamic group membership.

Table 3 provides a detailed overview of the entire test plan.

Table 3. Overview of the defined test plan, detailing the parameters used in each test scenario.

| Test # | Sessions | Hops | Type | Topology | Receivers |
|---|---|---|---|---|---|
| 1 | 1 | 1–4 | Unicast | Static | 1 |
| 2 | 1 | 1–4 | Multicast | Static | 1 |
| 3 | 1 | 2 | Multicast | Static | 2-3 |
| 4 | 2 | 2 | Multicast | Static | 1 per session |
| 5 | 1 | 2 | Multicast | Moving | 1 (mobile Rx) |
| 6a | 1 | 2 | Multicast | Changing | 2 → 3 (1 joins) |
| 6b | 1 | 2 | Multicast | Changing | 3 → 2 (1 leaves) |

Full details, including test expectations and precise configurations, are provided in Appendix B.

*7.1.2 Traffic Generation.* To execute the tests defined in Section 7.1.1, a tool was needed to generate UDP and multicast traffic with configurable bandwidth and packet sizes.

iPerf2 was chosen for its precise traffic control and multicast support, which iPerf3 lacks. All tests used UDP (`-u`), which was required for multicast traffic and to avoid transport-layer effects, and ran for 60 seconds (`-t 60`) to average out transient fluctuations. The specific iPerf2 options are described below.

(1) **Bandwidth Constraints**
Links in the simulated topology have an approximate throughput of ~26 Mbps (details in Appendix F). The following bandwidths were tested:
- **5, 10, 15 Mbps** Well below link capacity.
- **20, 25 Mbps** Near saturation, with iPerf2 generating always more traffic then specified.
- **26 Mbps:** Slightly over-saturated to test congestion handling.

(2) **Packet Size Considerations**
BATMAN-adv's forwarding depends on packet size, as described in Section 6.1. For the multicast tests with up to 3 listeners, the Maximum Ethernet Frame size was calculated as 1216 bytes (Formula 1).
Since iPerf2's `-l` flag specifies the payload size excluding the UDP header, 8 bytes were subtracted, resulting in 1208 bytes for multicast tests. For consistency, this same adjustment reduced the maximum packet size for other tests from 1234 bytes to 1226 bytes.

The final iPerf configurations are provided in Appendix C

## 7.2 Test Framework Implementation

To execute the test plan efficiently and reduce manual work, a custom Python-based framework was developed. It interfaces with the simulation environment to automate multicast optimization settings, initiate traffic flows, and collect performance metrics.

The framework does not create or modify the network topology; instead, it assumes a predefined scenario manually loaded in the simulator. Topology-specific tests therefore require manual initialization. However, once the correct scenario is active, the framework automatically executes all relevant tests, iterating over specified parameters such as multicast optimizations, session counts, and receiver numbers.

The framework was crucial in generating the final dataset used for analysis. The full source code and documentation are available in the GitLab repository listed in Appendix D.

# 8 RESULTS

## 8.1 Unexpected Behaviour and Resolution

While performing test 1 and 2 of the test plan, Unicast, Multicast Forceflood, and Multicast Packet optimizations showed the expected throughput across all hop counts. However, the Multicast Unicast optimization performed significantly worse at all tested bandwidths, the results for a configured bandwidth of 20 Mbps over 1 hop is shown in Table 4.

The significant performance drop suggested a potential misconfiguration. To investigate, a `tcpdump` probe was placed on the `eth0.1.21` interface of the first BATMAN node (see Section 5) and the test for Multicast Unicast was repeated. Analysis of the captured traffic in Wireshark revealed that multicast packets were duplicated and flooded to all nodes instead of only the single recipient, indicating

Table 4. Expected vs. actual bandwidth for multicast optimizations over one hop.

| Optimization | Hops | Expected bandwidth | Actual bandwidth |
|---|---|---|---|
| Unicast | 1 | 20 Mbits/sec | 20.9 Mbits/sec |
| Multicast Forceflood | 1 | 20 Mbits/sec | 20.9 Mbits/sec |
| Multicast Packet | 1 | 20 Mbits/sec | 20.0 Mbits/sec |
| Multicast Unicast | 1 | 20 Mbits/sec | 247 Kbits/sec |

a potential issue with the BATMAN-adv multicast group management.

Reviewing BATMAN-adv documentation showed that the Linux bridge connected to `bat0` must either act as a multicast querier or be connected to one, which was missing initially. To ensure proper multicast group management in BATMAN-adv and avoid unwanted packet loss or flooding, the following settings were enabled in the network configuration:

- `multicast_querier` set to 1 on the bridge device to enable multicast querier functionality,
- `igmp_snooping` set to 1 to allow the bridge to track multicast group memberships and support the querier,
- `/sys/class/net/br-mesh/brif/bat0/multicast_flood` set to 1 to permit multicast flooding on the `bat0` interface; without this, multicast packets originating from the EE connected to the bridge would not be forwarded to the BATMAN interface.

This combination allowed the bridge to manage multicast groups correctly while ensuring multicast packets were not prematurely dropped. After applying these changes, the Multicast Unicast optimization performed as expected, matching expected throughput of 20.9 Mbit/sec over one hop.

### 8.2 Final Measurement Results Under Correct Configuration

All measurement results are summarized below. Detailed figures and plots for all tests are provided in Appendix G.

(1) **Unicast Baseline**
Throughput slightly decreased with increasing hop count but remained largely stable from 1 to 4 hops. Unicast showed near-zero packet loss up to 25 Mbps bandwidth, achieving a maximum throughput of approximately 24.6 Mbps. Jitter decreased modestly at higher bandwidths and increased slightly with hop count.

(2) **Single-Receiver Multicast**
`multicast_forceflood` behaved similarly to Unicast with near-zero packet loss and comparable throughput. Meanwhile, `multicast_unicast` and `multicast_packet` experienced around 1% packet loss, increasing slightly at higher bandwidths, causing moderate throughput reductions likely due to additional BATMAN-adv header overhead. Jitter trends were similar to Unicast.

(3) **Multicast with Multiple Receivers**
With two receivers, `multicast_forceflood` and `multicast_packet` maintained performance comparable to Single-Receiver Multicast. At three receivers, `multicast_forceflood` throughput dropped significantly at 25 Mbps due to high packet loss on Node 2, while Nodes 3 and 4 remained consistent with test 2. `multicast_unicast` showed irregular behaviour consistent with Section 8.1, with Node 3 suffering significant packet loss above 10 Mbps and variable throughput across nodes.

(4) **Two Concurrent Multicast Sessions**
`multicast_forceflood` throughput capped near half network capacity ($\tilde{1}$3 Mbps), as could be expected with two simultanious streams. `multicast_packet` maintained stable performance comparable to single-session tests, and `multicast_unicast` exhibited throughput patterns similar to Multicast with Multiple Receivers at low bandwidths, aligning with `multicast_packet` above 20 Mbps. Jitter measurements reflected these trends. Due to unrealistic throughput in `multicast_forceflood` (see Section 6), subsequent tests focused on `multicast_packet` and `multicast_unicast`.

(5) **Mobile Receiver**
`multicast_unicast` was the only strategy maintaining some data transfer at low bandwidth ($\tilde{1}$ Mbps), though throughput dropped sharply afterward with near 100% packet loss, similar to `multicast_packet`, likely due to slow routing convergence. Jitter remained consistently high, indicating dynamic link changes.

(6) **Changing Topology**
Node join and leave events caused transient disruptions, but overall performance after convergence was comparable to static scenarios except for the changing nodes. The joining node achieved higher throughput with increased packet loss and low jitter (less than 1 ms), while the leaving node had lower throughput and significantly higher jitter (more than 10 ms), reflecting mobility-related patterns. This occurs because the joining node starts far but ends up close and briefly static near the sender, whereas the leaving node moves away immediately and remains out of range.

## 9 DISCUSSION AND CONCLUSIONS

This section discusses how the research addressed the challenge of optimizing multicast video streaming over MaritimeManet using BATMAN-adv, as outlined in Section 3. The discussion reflects on the research questions, interprets key findings, and highlights implications for maritime network design and future work.

First, regarding communication issues in the Digital Twin (RQ1), the study identified that the TAP interfaces connecting virtual machines to Linux bridges caused packet loss when VLAN tags were detected on a non-VLAN aware TAP interface. These issues disrupted forwarding and distorted performance measurements, making initial tests unreliable. By configuring the TAP interfaces to be VLAN-aware, the communication problems were resolved, enabling reliable data transmission and ensuring the Digital Twin environment accurately reflected the intended network behaviour.

Second, in exploring available multicast optimizations in BATMAN-adv (RQ2), the research examined optimizations documented in the BATMAN-adv wiki [10], leading to the selection of three approaches:

the dedicated BATMAN-adv multicast packet type, multicast transmitted as unicast, and multicast transmitted as broadcast. These optimizations were not configurable using settings but were instead triggered automatically by conditions such as packet size and number of listeners. This behaviour required developing automation scripts to systematically generate the required traffic conditions for each optimization, allowing precise and repeatable testing.

Third, the study implemented a structured test plan to evaluate the performance of these optimizations under diverse network conditions (RQ3). Tests varied critical parameters such as hop count, number of receivers, bandwidth, mobility, and dynamic topology changes. The automated framework facilitated extensive measurement collection and ensured that all scenarios were tested consistently and could easily be replicated for further research.

Finally, the test results provided clear insights into the suitability of each multicast optimization for MaritimeManet. Multicast transmitted as broadcast showed artificially high throughput results due to the simulator's reliance on Ethernet rather than true Wi-Fi behaviour, making it unreliable for realistic maritime scenarios. Multicast as unicast performed well at low bandwidths and with few nodes, but at higher bandwidths and larger receiver groups, it exhibited significant asymmetry, with some nodes consistently achieving higher throughput than others, making it unsuitable for multicast video streaming with multiple listeners. In contrast, the dedicated multicast packet demonstrated stable and reliable performance, maintaining low packet loss and high throughput even with increased hop count and multiple receivers. This indicates that the dedicated multicast packet approach is the most promising strategy for supporting high-throughput multicast video streaming in MaritimeManet.

Overall, this research demonstrates that careful network configuration and the selection of appropriate multicast optimizations in BATMAN-adv can significantly improve multicast performance in MaritimeManet. These findings are valuable for designing robust communication systems for bandwidth-intensive applications in maritime environments.

## 9.1 Limitations

While this research provides valuable insights into multicast optimization for MaritimeManet, several limitations must be acknowledged.

First, the Digital Twin environment used Ethernet-based virtualization, which does not fully replicate wireless medium behaviour, even though network emulation was used to introduce delays, packet loss and maximum throughput based on simulated RSSI values. This limitation contributed to the artificially high throughput measurements observed for multicast transmitted as broadcast, making those results less representative of real maritime scenarios.

Second, the experiments were conducted with a limited number of nodes, focusing on small-scale network topologies. Performance and scalability in larger networks remain to be investigated.

Third, testing was restricted to specific traffic patterns, primarily data transmissions at defined bandwidths. Other application types or variable traffic conditions might yield different results.

Finally, while dynamic scenarios like mobility and topology changes were considered, the simplified movement patterns and stable link conditions in the Digital Twin may not fully capture the complexity of real-world maritime environments.

## 9.2 Future Work

Several opportunities exist for extending this research.

Future work should validate the findings on physical hardware in realistic maritime conditions to confirm the applicability of the multicast optimizations under true wireless constraints.

Integrating a detailed wireless simulation layer into the Digital Twin would help capture essential radio effects, allowing more accurate evaluation of multicast performance, especially for broadcast scenarios.

Scaling experiments to larger mesh networks is crucial to assess routing convergence, multicast group management, and overall network stability under higher loads.

Further research could explore mixed traffic scenarios, evaluating how multicast optimizations coexist with other types of maritime network traffic.

Lastly, investigating enhancements to BATMAN-adv itself—such as adaptive mechanisms for selecting multicast strategies based on real-time conditions—could further improve multicast performance in MaritimeManet.

## ACKNOWLEDGMENTS

I would like to thank my supervisor, Jan Laarhuis, for his valuable feedback and support throughout my research project. His guidance helped me overcome challenges and deepen my understanding.

## REFERENCES

[1] Fahad S. Alqurashi, Abderrahmen Trichili, Nasir Saeed, Boon S. Ooi, and Mohamed-Slim Alouini. 2023. Maritime Communications: A Survey on Enabling Technologies, Opportunities, and Challenges. *IEEE Internet of Things Journal* 10, 4 (2023), 3525–3547. https://doi.org/10.1109/JIOT.2022.3219674

[2] Bill Fenner. 1997. Internet Group Management Protocol, Version 2. RFC 2236. https://doi.org/10.17487/RFC2236

[3] Guido R. Hiertz, Dee Denteneer, Sebastian Max, Rakesh Taori, Javier Cardona, Lars Berlemann, and Bernhard Walke. 2010. IEEE 802.11s: The WLAN Mesh Standard. *IEEE Wireless Communications* 17, 1 (2010), 104–111. https://doi.org/10.1109/MWC.2010.5416357

[4] Nathan Jongejan. 2024. Virtualisation of Swarms : Designing a digital twin prototype for MaritimeManet. http://essay.utwente.nl/101759/

[5] K.S. Kumar and Saumya Hegde. 2009. Multicasting in Wireless Mesh Networks: Challenges and Opportunities. 514 – 518. https://doi.org/10.1109/ICIME.2009.92

[6] Jan H. Laarhuis. 2010. MaritimeManet: Mobile ad-hoc networking at sea. In *2010 International WaterSide Security Conference*. 1–6. https://doi.org/10.1109/WSSC.2010.5730256

[7] P. Lavanya, V. Siva Kumar Reddy, and A. Mallikarjuna Prasad. 2017. Research and survey on multicast routing protocols for MANETs. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. 1–4. https://doi.org/10.1109/ICECCT.2017.8117929

[8] Ligang Liu, Jianpo Liu, Hanwang Qian, and Jun Zhu. 2018. Performance Evaluation of BATMAN-Adv Wireless Mesh Network Routing Algorithms. 122–127. https://doi.org/10.1109/CSCloud/EdgeCom.2018.00030

[9] Open-Mesh.org. 2018. *B.A.T.M.A.N. advanced Wiki.* https://www.wireshark.org/docs/wsug_html_chunked/AppToolstcpdump.html Accessed: 2025-06-26.

[10] Open-Mesh.org. 2024. *batman-adv Multicast Optimizations.* https://www.open-mesh.org/projects/batman-adv/wiki/Multicast-optimizations Accessed: 2025-06-22.

[11] Open-Mesh.org. 2025. *Open-Mesh Branches Explained.* https://www.open-mesh.org/projects/open-mesh/wiki/BranchesExplained Accessed: 2025-06-25.

[12] Michal Raczkiewicz. 2023. Better Approach To Mobile Ad-hoc Networking in MaritimeManet. http://essay.utwente.nl/96407/

[13] Dinesh Ramphull, Avinash Mungur, Sheeba Armoogum, and Sameerchand Pudaruth. 2021. A Review of Mobile Ad hoc NETwork (MANET) Protocols and their Applications. In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. 204–211. https://doi.org/10.1109/ICICCS51141.2021.9432258

[14] Benjamin Sliwa, Stefan Falten, and Christian Wietfeld. 2019. 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 Conference (VTC2019-Spring)*. 1–7. https://doi.org/10.1109/VTCSpring.2019.8746361

[15] Bhanu Prakash Reddy Tholeti. 2013. *Hypervisors, Virtualization, and Networking*. Elsevier, 387–416. https://doi.org/10.1016/b978-0-12-401673-6.00016-7

[16] Mok Shao Chung Thomas, Mau-Luen Tham, Yi Jie Wong, and Yoong Choon Chang. 2024. Performance Evaluation of Video Streaming in Wireless Mesh Networks. In *2024 IEEE 12th Conference on Systems, Process & Control (ICSPC)*. 390–394. https://doi.org/10.1109/ICSPC63060.2024.10862838

[17] Zhe Wang, Jiao Zhang, Yichi Zhang, Haitao Zhao, and Jibo Wei. 2022. Comparison of Mobile AdHoc Network Routing Protocols Based on NS3. In *2022 IEEE 22nd International Conference on Communication Technology (ICCT)*. 11–16. https://doi.org/10.1109/ICCT56141.2022.10072455

[18] Wireshark.org. 2025. *Capturing with "tcpdump" for viewing with Wireshark*. https://www.wireshark.org/docs/wsug_html_chunked/AppToolstcpdump.html Accessed: 2025-06-26.

[19] Aleš Švigelj and Melisa Junuzović. 2017. Network Coding-Assisted Retransmission Scheme for Video- Streaming Services over Wireless Access Networks. In *Broadband Communications Networks*, Abdelfatteh Haidine and Abdelhak Aqqal (Eds.). IntechOpen, Rijeka, Chapter 8. https://doi.org/10.5772/intechopen.71784

# A AI STATEMENT

During the preparation of this work, I used ChatGPT GPT-4o to assist with improving academic language and to help identify issues within the network configurations. After using this tool/service, I thoroughly reviewed and edited the content as needed, taking full responsibility for the final outcome.

# B DETAILED TEST PARAMETERS AND CONFIGURATIONS

## B.1 Test Parameters

The following elaborates on the key test parameters introduced in Section 7.1.

(1) **Number of Sessions**
A session consists of one multicast sender (Tx) and one or more receivers (Rx). By increasing the number of concurrent sessions sharing the same topology, we assess how session density impacts bandwidth utilization and forwarding efficiency.

(2) **Hop Count**
Hop count represents the number of intermediate nodes between a sender and receiver. Since forwarding complexity typically grows with path length, tests with 1–4 hops highlight how each multicast strategy performs under increasing forwarding demands.

(3) **Casting Mode**
Two traffic types are examined:
- **Unicast (UDP):** Used to establish a performance baseline for single-source, single-destination traffic. UDP is chosen over TCP to ensure consistency with multicast testing and to avoid the influence of transport-layer retransmissions.
- **Multicast (UDP):** Covers the three BATMAN-adv strategies. Each strategy is tested individually under identical conditions for direct comparison.

(4) **Topology Type**
Three topological configurations are tested to simulate both stable and dynamic maritime conditions:
- **Static:** Node positions and links remain fixed, providing a baseline for performance comparison.
- **Mobile Nodes:** Nodes move but remain connected, reflecting scenarios like ships maintaining radio contact during coordinated movement.
- **Changing Topology:** Nodes may join or leave the network mid-session, simulating intermittent connectivity due to movement in or out of range.

(5) **Number of Receivers**
This is relevant only to multicast tests. Varying the number of receivers reveals how well each strategy scales. Based on BATMAN-adv's forwarding mechanisms, it is expected that:
- Multicast packet forwarding performs best in larger groups.
- The unicast optimization incurs significant overhead as receiver count increases.
- Broadcast in the simulation environment may appear to perform well for both small and large groups. However, this does not accurately reflect real-world wireless behaviour, as the simulator does not model the increased contention, collisions, or reliability issues typically associated with broadcast in actual wireless networks.

## B.2 Detailed Test Plan and Expectations

Each test described in Section 7.1.1 has the following detailed configuration and expectations:

(1) **Unicast baseline over multiple hops**
- **Description:** A single unicast session is tested over 1 to 4 hops in a static network topology to establish baseline performance metrics for comparison with multicast optimizations.
- **Expectation:** Performance is expected to degrade linearly with increasing hop count due to accumulated delays and potential retransmissions, providing a reference for subsequent multicast tests.

(2) **Multicast with a single listener**
- **Description:** The same scenario as Test 1, but using multicast transmissions while maintaining only a single receiver.
- **Expectation:** Results should closely match the unicast baseline, as there is a single listener, making all optimizations similar to a unicast transmission.

(3) **Multicast with multiple listeners (2–3)**
- **Description:** A multicast session spanning 2 hops, now including 2 or 3 receivers. This test assesses how multicast scales with increasing group size.
- **Expectation:** It is expected that broadcast and multicast packet forwarding outperform unicast forwarding when multiple listeners are present, since they send a single transmission instead of separate transmissions for each node.

(4) **Multicast with multiple senders**
- **Description:** Two multicast sessions are active simultaneously over 2 hops, each with its own single receiver, to evaluate the network load under concurrent multicast sources.

- **Expectation:** An increase in network utilization is expected compared to a single multicast session. However, since there is only a single listener for each multicast sender, all senders are expected to have similar results for all the multicast optimizations.

(5) **Multicast with moving receiver**
- **Description:** The same network as in Tests 3 and 4 is used, but a single receiver node is mobile, continuously connecting and disconnecting from different nodes while maintaining an active session.
- **Expectation:** Increased packet loss, higher jitter, and low bandwidth are expected due to frequent route changes. The robustness of multicast forwarding strategies will be tested in dynamic link conditions.

(6) **Multicast with changing topology**
- **Description:** Two sub-tests are conducted:
  - **6a – Node joins:** A new node joins the multicast group, changing the receiver count from 2 to 3.
  - **6b – Node leaves:** A node leaves the multicast group, reducing the receiver count from 3 to 2.
- **Expectation:** Transient disruptions are expected during join/leave events due to tree reconfiguration or group management overhead. However, the steady-state performance after convergence is anticipated to be similar to Test 3, assuming the multicast protocols handle group changes efficiently.

## C IPERF CONFIGURATION

Unicast traffic was generated using the commands in Table 5.

Table 5. iPerf configuration for Unicast tests

| Node | IP Address | Command |
|---|---|---|
| Server | 10.10.0.x | `iperf -s -u` |
| Client | 10.10.0.y | `iperf -c 10.10.0.x -u -t 60 -b {rate}M -l 1226` |

For multicast, senders transmit to a group address to which listeners subscribe. The commands are specified in Table 6.

- Group addresses are chosen from the `239.0.0.x/24` range, reserved for private multicast use. A listener subscribes to a multicast group by binding to the group address using `-B 239.0.0.x`.
- A TTL value of 32 (`-T 32`) is set to prevent premature packet drops. However, BATMAN-adv operates at Layer 2 and does not consider IP TTL during forwarding.

Table 6. iPerf configuration for Multicast tests

| Node | IP Address | Command |
|---|---|---|
| Sender | 10.10.0.x | `iperf -c 239.0.0.x -u -T 32 -t 60 -b {rate}M -l {size}` |
| Listener | 10.10.0.y | `iperf -s -u -B 239.0.0.x` |

## D GITLAB REPOSITORY

For this study, a dedicated GitLab repository was created to host all source code, scripts, and Python programs developed during the research.

The repository is publicly available at: https://gitlab.utwente.nl/s2987716/maritimemanet-multicast

The repository includes (but is not limited to):

- The complete multicast testing environment
- An iPerf parser for generating CSV files from multicast test output
- Custom TCPDUMP probes for both execution environments and BATMAN nodes

Each component is documented in the repository README to support reproducibility and further experimentation.

## E TCPDUMP PROBE

This appendix includes the full TCPDUMP probe script used for various tests.

Listing 1. TCPDUMP Probe Script

```
#!/bin/bash
# TCPDUMP script for nodes using BASH (such as
    Debian nodes)

# Check if an interface argument is provided
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <interface>"
    exit 1
fi

# Define variables
ROOT_INTERFACE="ens20"
IP_ADDR="$(ip -4 addr show $ROOT_INTERFACE | awk
    '/inet / {split($2, a, "/"); print a[1]}')"

# Get the interface and hostname
INTERFACE="$1"
HOSTNAME="$(hostname)"

# Generate the file name
DUMP_FILE="capture-$INTERFACE-$IP_ADDR-$HOSTNAME.
    pcap"

# Define the SSH host to copy the files to
REMOTE_USER="maritimemanet"
REMOTE_HOST="10.0.0.2"

# Define the base directory for tcpdump on the
    remote server
BASE_DIR="/home/maritimemanet/digital_twin-main/
    tcpdump/"

# Find the highest numbered subdirectory
HIGHEST_DIR=$(ssh $REMOTE_USER@$REMOTE_HOST "ls -1
     $BASE_DIR | sort -n | tail -n 1")

REMOTE_PATH="$BASE_DIR/$HIGHEST_DIR/$DUMP_FILE"

# Check if the highest directory exists
if [ -z "$HIGHEST_DIR" ]; then
    echo "Error: No subdirectories found in
        $BASE_DIR on the remote server."
    exit 1
```

```
fi

# Start tcpdump in the background
echo "Starting tcpdump on interface $INTERFACE..."
tcpdump -i "$INTERFACE" -w "$DUMP_FILE" &

# Get the PID of the tcpdump process
TCPDUMP_PID=$!

# Function to stop tcpdump and copy the file
cleanup() {
    echo "Stopping tcpdump..."
    kill $TCPDUMP_PID
    wait $TCPDUMP_PID 2>/dev/null

    echo "Copying dump file to remote server..."
    scp "$DUMP_FILE" $REMOTE_USER@$REMOTE_HOST:"
        $REMOTE_PATH"

    echo "Done."
}

# Trap SIGINT (Ctrl+C) and SIGTERM (kill) signals
trap cleanup SIGINT SIGTERM

# Wait for tcpdump to finish
wait $TCPDUMP_PID
```

## F BANDWIDTH MODELLING IN THE SIMULATION

In the simulation's state handler, bandwidth values are predefined and assigned based on Received Signal Strength Indicator (RSSI) values. Higher RSSI values indicate stronger signal strength and therefore higher potential bandwidth. For this simulation, RSSI values below -90 dBm are considered insufficient for data transmission. The mapping between RSSI values and corresponding bandwidths is provided in Table 7.

Table 7. Mapping of RSSI values to bandwidth in the simulation.

| RSSI (dBm) | Bandwidth (Mb/s) |
|---|---|
| ≥ -64 | 65 |
| ≥ -65 | 58.5 |
| ≥ -66 | 52 |
| ≥ -70 | 39 |
| ≥ -74 | 26 |
| ≥ -77 | 19.5 |
| ≥ -79 | 13 |
| ≥ -82 | 6.5 |
| < -82 | 1 |
| < -90 | 0 |

The simulation models three different ship sizes, each equipped with a distinct number of antennas: small (6 antennas), medium (12 antennas), and large (24 antennas). Transmission power scales with the number of antennas, so smaller ships generate lower RSSI values at equivalent distances compared to larger ships. As a result, bandwidth also varies depending on ship size and separation distance.

Table 8 illustrates the relationship between transmission distance and resulting bandwidth for ships of equal size. For example, a pair of small ships (each with 6 antennas) separated by 2000 meters achieves a bandwidth of 19.5 Mb/s.

Table 8. Maximum transmission distances (in meters) for different ship sizes and bandwidths (in Mb/s).

| Antennas | 65 | 58.5 | 52 | 39 | 26 | 19.5 | 13 | 6.5 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 6 | < 568 | < 627 | < 691 | < 1023 | < 1514 | < 2032 | < 2472 | < 3317 | < 6586 |
| 12 | < 1245 | < 1373 | < 1514 | < 2241 | < 3317 | < 4450 | < 5413 | < 7264 | < 14423 |
| 24 | < 4035 | < 4450 | < 4908 | < 7264 | < 10749 | < 14422 | < 17545 | < 23540 | < 46742 |

## G DETAILED PLOTS AND FIGURES

This appendix contains all measurement plots referenced in Section 8.2, organized by test number and parameter settings for clarity. The figures present the results of Tests 1 through 6, showing measured network performance metrics, including bandwidth, packet loss, jitter, or a combination thereof, plotted against the configured target bandwidths used during the experiments. These plots help visualize how the network behaved under different load conditions and settings.

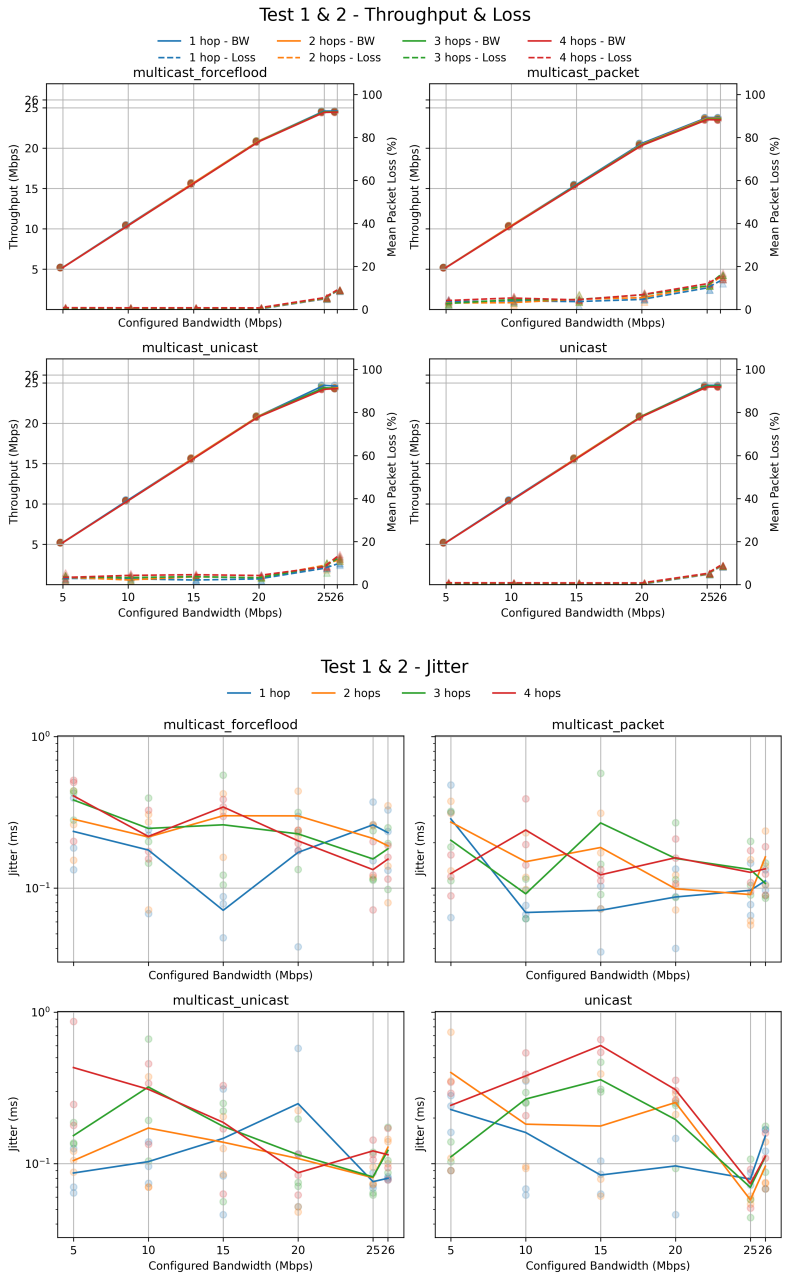


Fig. 6. Results of Test 1 and 2: bandwidth, packet loss, and jitter.

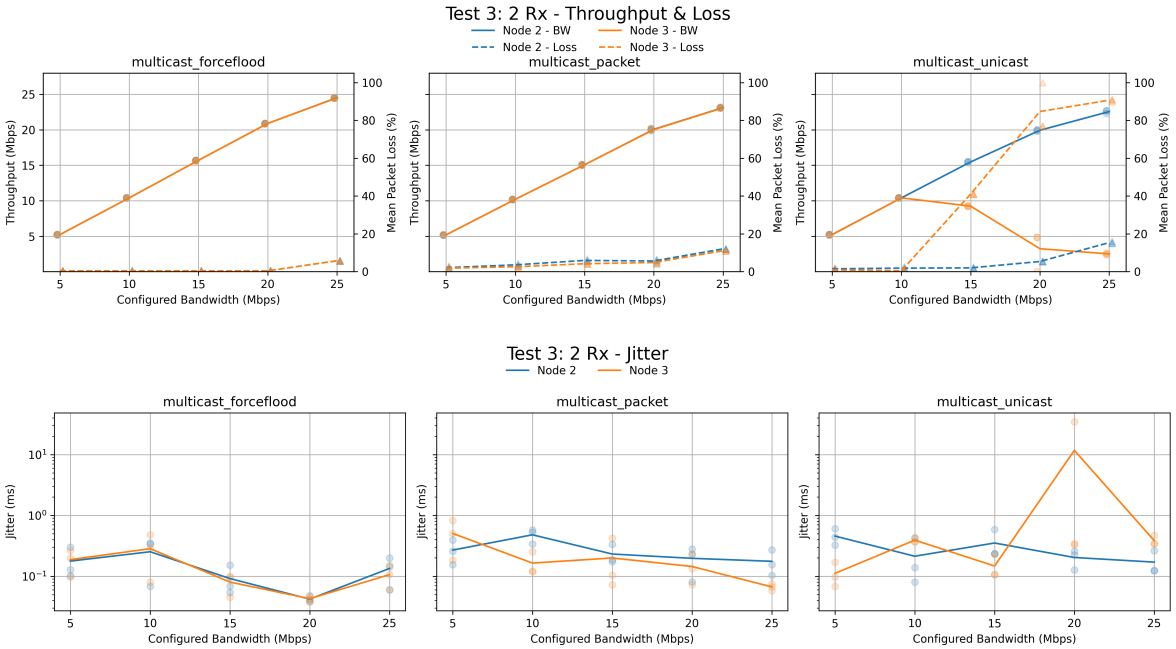


Fig. 7. Results of Test 3 (2 receivers): bandwidth, packet loss, and jitter.

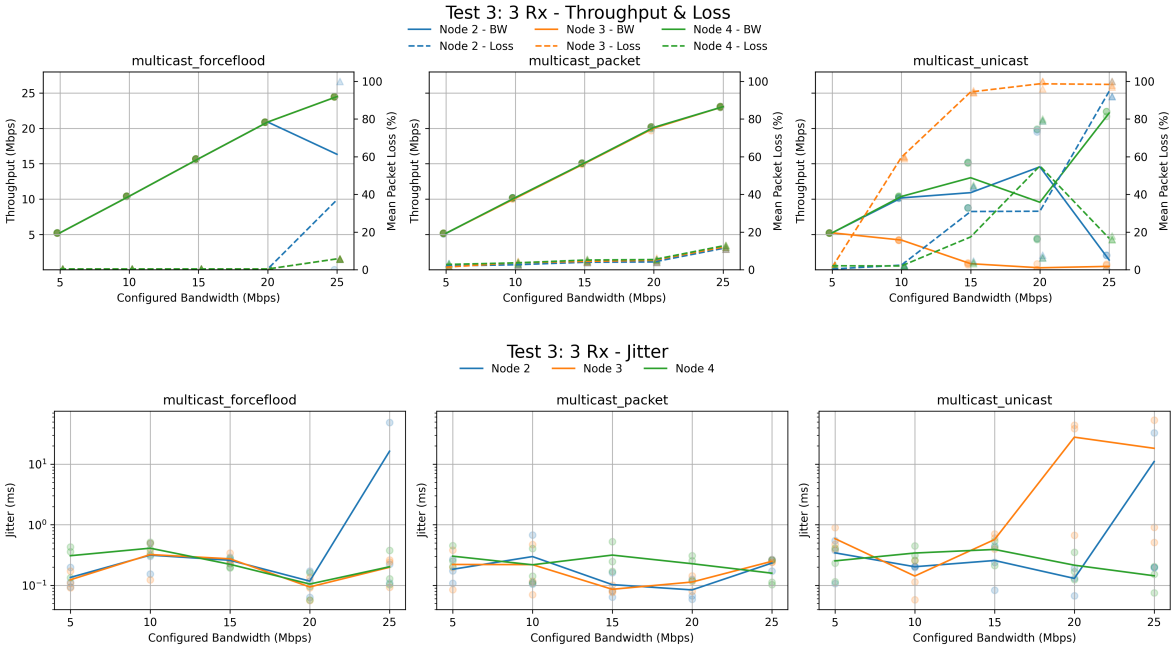


Fig. 8. Results of Test 3 (3 receivers): bandwidth, packet loss, and jitter.

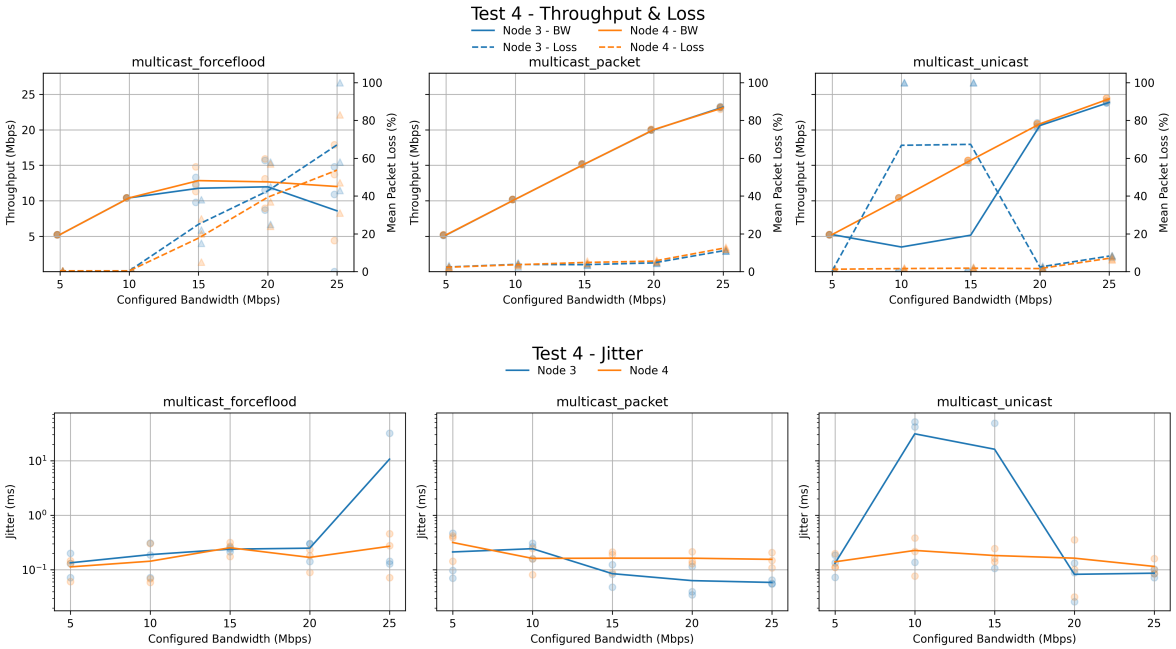


Fig. 9. Results of Test 4: bandwidth, packet loss, and jitter.

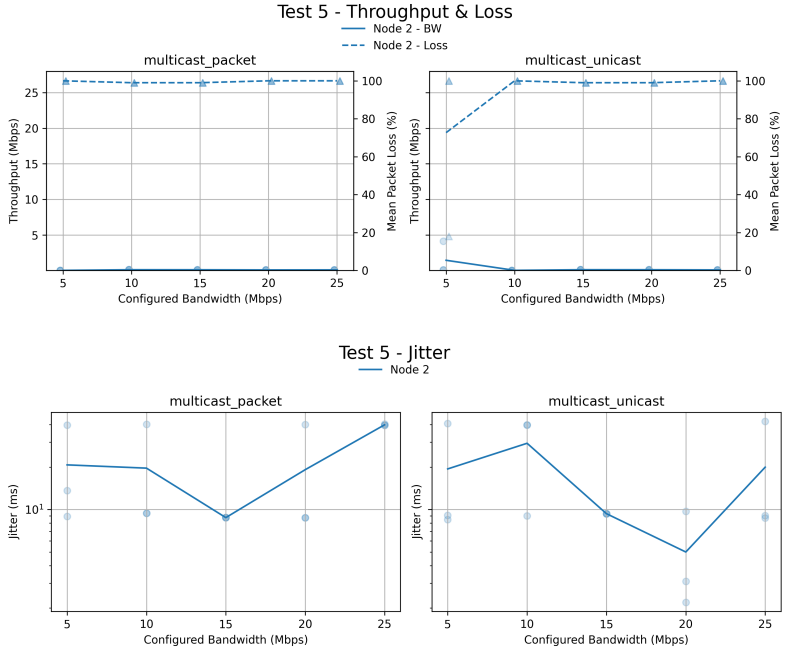


Fig. 10. Results of Test 5: bandwidth, packet loss, and jitter.

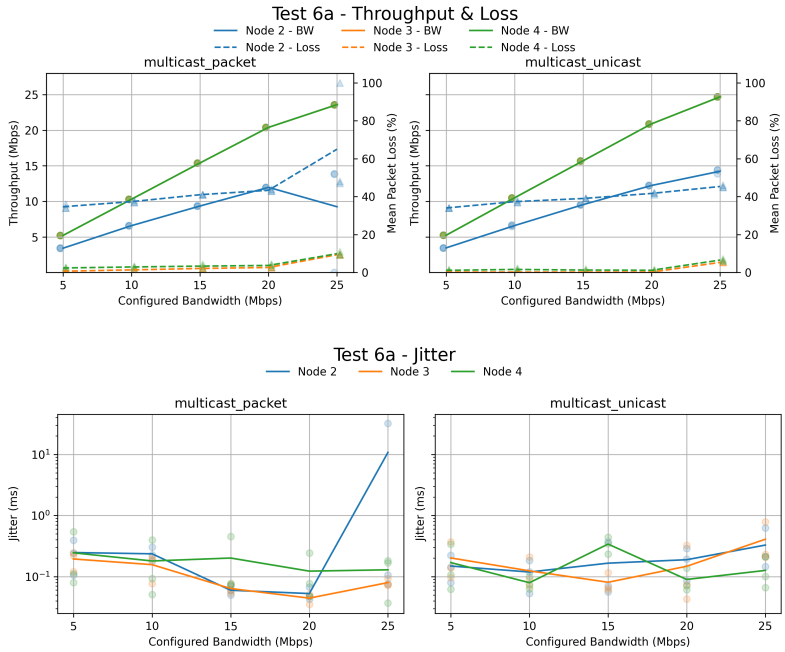


Fig. 11. Results of Test 6a: bandwidth, packet loss, and jitter.

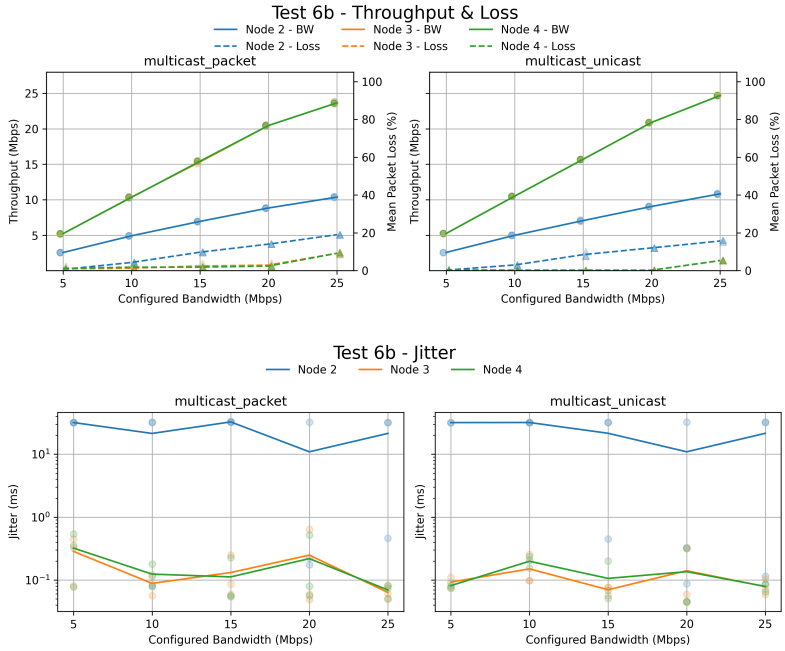


Fig. 12. Results of Test 6b: bandwidth, packet loss, and jitter.