MANET Voice Network architecture and communication protocol

Draft 1 *September 17, 2026*

MANET Voice Communicator R&D Group

www.manet-voice.org

Introduction

The aim of this document is to define architecture and communication protocol for short-range low-power radio frequency voice communication network that uses **MANET** technology. Network nodes that are not directly reachable communicate via other nodes that act as relays. Proposed network is self-organizing and doesn't require a central server.

Definitions

The network will be operating in single-talker broadcast mode, only one person may speak at any given time and voice is broadcasted to every node.

Communication is half-duplex; nodes are always listening and only occasionally switch to transmit.

Any node can act as a relay.

To make relaying possible, available total transmission bitrate should be much bigger than needed for the voice codec.

N is the maximum number of nodes in the network.

R is the maximum number of retransmissions (relays in a route).

Network architecture

Each node has its unique **ID**. Fig. 1 shows an example of network topology. Nodes connected by arrows can communicate with each other directly.

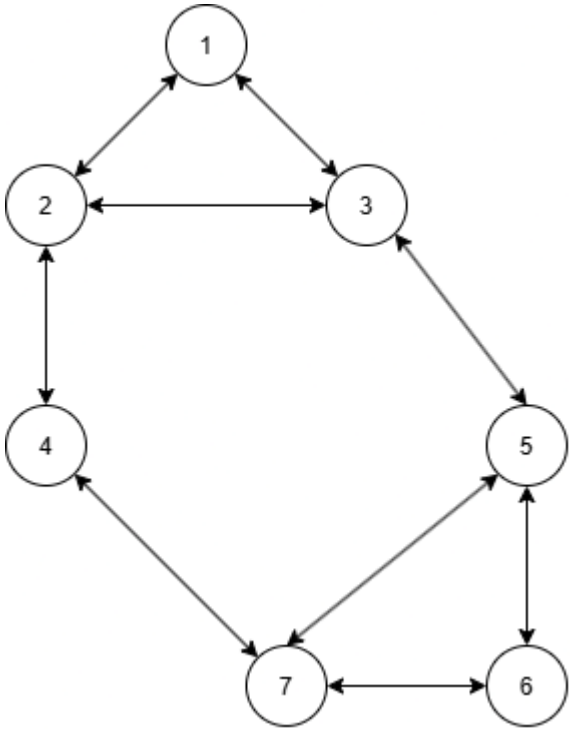


Fig. 1. Network topology example

Nodes can transmit **control** packets and **voice** packets.

Control packets have variable length.

Control packets are never relayed.

Before transmitting control packet, node waits for a random interval **Tr**. Node does not transmit control packet until it detects that the channel is free. That reduces probability of collisions without introducing complex synchronization mechanism.

After powering up, node is in **idle** mode and transmits **HELLO** control packets every **Th** seconds. The interval **Th** is selected to minimize network overhead while allowing the network to respond to topology and RF reception conditions changes as quickly as possible. An out of order HELLO packet may be transmitted right after receiving a HELLO packet from a new node.

HELLO packet contains:

- transmitting node's ID
- a list of node IDs from which the transmitting node has successfully received HELLO packets in the past (neighbors list). Each node in the list includes associated signal quality indicator **Qs**, from 0 = very poor to 3 = very good. A node is deleted from neighbors list if there were no packets received from it in the past **Tn** seconds.
- any additional data unrelated to network operation (like GPS coordinates, battery charge level, etc.)

Thus, each node can construct a topology map of its immediate neighborhood (one-hop neighbors) and identify nodes that are reachable through a single relay (two-hop neighbors). Furthermore, for each one-hop neighbor, the node knows the signal quality Qs in both directions (receive and transmit). This information is essential for constructing relay routes.

Talk session establishment and route selection

Establishing a talk session involves selecting relay routes from the session initiator to all other nodes in the network.

Each relay (including the initiator) selects downstream relays from its list of one-hop neighbors. Nodes selected as downstream relays have additional neighbors that the current relay cannot reach directly. The relay assigns voice transmission **slots** to the selected downstream relays in a way that relays don't interfere with each other.

This process continues until one of the following conditions is met:

- ✓ number of consecutive retransmissions reaches R
- ✓ current relay has no eligible downstream relays
- ✓ a conflict is detected (a node has already been selected as a relay)

Each relay (including the initiator) constructs and transmits **ROUTE_SETUP** packet containing:

- talk session initiator ID
- TALK_ID: session number, which, together with the initiator ID, uniquely identifies the session. It is incremented for each subsequent talk session
- relay level (HOP). Set to 0 for the initiator and incremented for each subsequent relay.
- list of upstream routes in the form: (node-slot, node-slot, …), (…). This allows the next node to determine route propagation direction and avoid loops. It also allows the node to determine which voice slots are available.
- list of downstream relays and their assigned slots, in the form: node-slot, node-slot, …

A node receiving a ROUTE_SETUP packet checks whether its ID is included in the list of downstream relays. If not, it doesn't participate in routing but still switches to **talk** mode and stops transmitting HELLO packets.

If its ID is present, the node checks proposed slot assignments for conflicts and determines whether further relaying is needed.

If relaying is needed, it selects the next relay nodes, assigns voice transmission slots, updates upstream route information, increments the hop count, and transmits its own ROUTE_SETUP packet.

If the route terminates at the node, it transmits a **ROUTE_END** packet containing:

- talk session initiator ID
- TALK_ID
- relay level (hop)
- list of established routes (route), in the form: (node-slot, node-slot, …), (…)

If a relay detects a slot assignment conflict, it resolves it and updates route lists in ROUTE_SETUP and ROUTE_END packets.

This means that for each ROUTE_SETUP packet that the node sends, it should receive a corresponding ROUTE_END packet from each selected downstream relay.

Once all required ROUTE_END packets have been received by the node, it sends its own ROUTE_END packet to the upstream relay node to indicate that the route is established.

When the initiator has received all ROUTE_END responses, route establishment is deemed complete and voice transmission can start.

If the initiator has no relays (a simple single-hop network), it transmits an empty ROUTE_SETUP packet so that all listening nodes can prepare for the talk session and stop HELLO packets transmission.

Reliable delivery of ROUTE_SETUP and ROUTE_END packets and collision prevention has not been addressed yet.

Voice transmission

Voice packets (**VOICE**) are transmitted using a synchronous frame structure consisting of equal-length slots. Slot duration **Ts** and total number of slots **S** are constrained by the maximum available bitrate and the need to minimize propagation delay (added by each retransmission).

Slot payload size is determined primarily by the amount of data produced by voice codec during frame duration **Tf**.

Frame duration: **Tf = (Ts × S) + Tp**,

where **Tp** is the data processing interval. This is maximum time allowed for receiver to process all packets received within one frame and be ready to retransmit one of them in the next frame, if necessary.

Frames are transmitted continuously without gaps. All slots (and therefore all voice packets) have equal lengths.

Ts Slot 1	**Ts Slot 2**	**…**	**Ts Slot S**	**Tp**

Fig. 2. Voice frame structure

Maximum voice propagation delay across the network:

Tmax = Tf × (R + 1)

All subsequent retransmissions are synchronized using received packets. Each slot contains its slot number, allowing receivers to determine frame start time from any received packet.

During talk session only VOICE packets are transmitted. Last packet contains a flag indicating session termination. If a node doesn't receive **Tv** consecutive voice frames, it considers talk session ended and returns to idle mode.

VOICE packet contains:

- talk session initiator ID
- TALK_ID: session number, which, together with the initiator ID, uniquely identifies the session. It is incremented for each subsequent communication session
- SEQ_NO: packet sequence number, used to detect missing packets
- Voice data

Example of route selection for network topology from Fig 1

Node **1** starts talk session at time **T**. It selects nodes **2** and **3** as downstream relays:

T+0: 1 -> ROUTE_SETUP hop = 0; upstream = [1, 1]; downstream = [2, 2], [3,3]

The process continues downstream to nodes **4** and **5**:

T+1: 2 -> ROUTE_SETUP hop = 1; upstream = [1, 1 -> 2,2], [1,1 -> 3,3]; downstream = [4, 4]

T+1: 3 -> ROUTE_SETUP hop = 1; upstream = [1, 1 -> 2,2], [1,1 -> 3,3]; downstream = [5, 4]

T+2: 4 -> ROUTE_SETUP hop = 2; upstream = [1, 1 -> 2,2 -> 4,4], [1,1 -> 3,3]; downstream = [7, 5]

T+2: 5 -> ROUTE_SETUP hop = 2; upstream = [1, 1 -> 2,2], [1,1 -> 3,3 -> 5,4]; downstream = [7, 5]

We assume node **7** receives ROUTE_SETUP packet from node **4** first. It selects node **5** as the next relay:

T+3: 7 -> ROUTE_SETUP hop = 3; upstream = [1, 1 -> 2,2 -> 4,4 -> 7,5], [1,1 -> 3,3]; downstream = [5, 1]

However, node **5** is already part of the route and has already selected node **7** as its downstream relay. Upon receiving this packet, node **5** responds that it will not act as a downstream relay for node **7**:

T+4: 5 -> ROUTE_END hop = 3; route = [1, 1 -> 2,2 -> 4,4 -> 7,5], [1,1 -> 3,3]

Node **7** then receives ROUTE_SETUP (T+2) and ROUTE_END packets from node **5**. It resolves the conflict:

T+6: 7 -> ROUTE_END hop = 3; route = [1, 1 -> 2,2 -> 4,4 -> 7,5], [1,1 -> 3,3 -> 5,6 -> 7,5]

The process continues upstream as relays can now send ROUTE_END responses:

T+7: 5 -> ROUTE_END hop = 2; route = [1, 1 -> 2,2 -> 4,4 -> 7,5], [1,1 -> 3,3 -> 5,6 -> 7,5]

T+7: 4 -> ROUTE_END hop = 2; route = [1, 1 -> 2,2 -> 4,4 -> 7,5], [1,1 -> 3,3 -> 5,6 -> 7,5]

T+8: 2 -> ROUTE_END hop = 1; route = [1, 1 -> 2,2 -> 4,4 -> 7,5], [1,1 -> 3,3 -> 5,6 -> 7,5]

T+8: 3 -> ROUTE_END hop = 1; route = [1, 1 -> 2,2 -> 4,4 -> 7,5], [1,1 -> 3,3 -> 5,6 -> 7,5]

After receiving ROUTE_END packets from nodes **2** and **3**, node **1** can start voice transmission. Each relay now knows its assigned voice slot.

Each potential relay node independently decides whether to be a relay or not.

Each potential relay can correct voice slot assignment of an upstream relay if it detects a collision. For example, in the scenario above, nodes 4 and 5 are assigned the same slots because they cannot hear each other. This creates a conflict for node 7 (who can hear both 4 and 5) and node 7 resolves it.

Note that node 7 is reached twice by different routes and so will receive two copies of the same voice packet in each frame.