Tree-based any-to-any routing
an architectural overview

Diego Lobba, Mat. number 198331
diego.lobba@studenti.unitn.it

***Abstract*—In this report it is described a protocol implementing tree-based any-to-any routing, detailing its functioning and architectural decisions. The report is organised as follows: in the first section it is given a brief description on the requirements and goals of the protocol, its architecture and implementation are then discussed, finally a brief set of statistics is presented to illustrate the performance of the implementation, considering both a duty-cycled and a non duty-cycled execution.**

I. Introduction

Wireless sensor networks are characterised to employ different communication patterns with respect to the usual IP LAN, due to a more loose interconnection among nodes which limits the network connectivity. In this context RPL [1] defines how to achieve effectively various communication patterns on a low-power and lossy (LLN) network.

In this report the focus is on the implementation of any-to-any communication, where point-to-point communication among two arbitrary nodes in the network is allowed. To this end, nodes have to gather enough information in order to route packets towards the destination's direction. This is achieved by establishing a hierarchical structure, namely a tree rooted at the sink node, where each node has routing information about all nodes within its sub-tree. The sink will therefore have information about all nodes in the network and will be the reference node when routing information are not available at intermediate nodes.

Each node contains routing information about each node belonging to its sub-tree, in addition to neighbouring nodes that can be contacted directly. If a node has not enough information on the destination, it forwards the packet to its parent until a common ancestor between the source and the destination is reached, the sink being the highest common ancestor of any two nodes. In case the sink has no information on a given destination, forwarding cannot be performed and the packet is discarded.

II. Protocol architecture

The protocol requires the establishment of a tree structure to build routing information upon. The tree is built by propagating a beacon message, providing a sequence number describing the actual building phase repetition and a cost metric which increases linearly with the hop count (Figure 1).

A node receiving a beacon sets the sender as the parent if one of the following conditions is met:

- the sequence number is more recent;
- in case a beacon with the same sequence number has already been received, it costs less.

In order not to consider senders with bad radio signal, a threshold on the sender's RSSI is considered, discarding all data received with power less than the threshold.

Routing information are organised within a table stating the node to contact in order to reach a given destination node. To each entry is additionally attached a timestamp (expressed in clock ticks) representing the insertion time of the entry. Entries are considered in routing decision until their expiration, defined by the `EXPIRATION_TIME` parameter.

The routing table is firstly initialised with neighbouring information gained in the tree-building phase, in which adjacent nodes are inserted into the table, or corresponding entries are updated in case they are already present. Subtree information are gathered by collecting *topology report* messages, where child nodes forward their routing table to their parent node.

To not trigger too many topology reports in the tree-building phase, topology reports start to be sent only after a given time, defined by the `STABLE_REPORT_DELAY` constant, set to two seconds. Although this strategy reduces the number of technology reports sent and therefore the possibility of collisions, it is expected to increase nodes duty-cycle, since the time between beacon reception and the corresponding report sending is quite small and the device has to be awaken repeatedly within a short interval.

Note that a new topology beacon is scheduled only in case the node's parent changes, while modifications in the node's neighbourhood do not trigger new topology reports. In order to refresh correct routing information, each node schedules periodically a new topology report, which allows to establish routing information also in collision scenarios, where a new report is not scheduled by the node's parent change.

Upon receiving a topology report, a node merges its routing table with entries contained in the message, refreshing existing entries. A *new topology report*, from the current node to the corresponding parent, is then sent. The process is repeated until the sink is reached. This approach avoids the flooding of repetitive topology reports flowing from all nodes towards the sink by limiting the report communication to occur only between a node and its direct parent (Figure 2).

Unfortunately, this strategy does not allow nodes in higher levels waiting for information coming from their

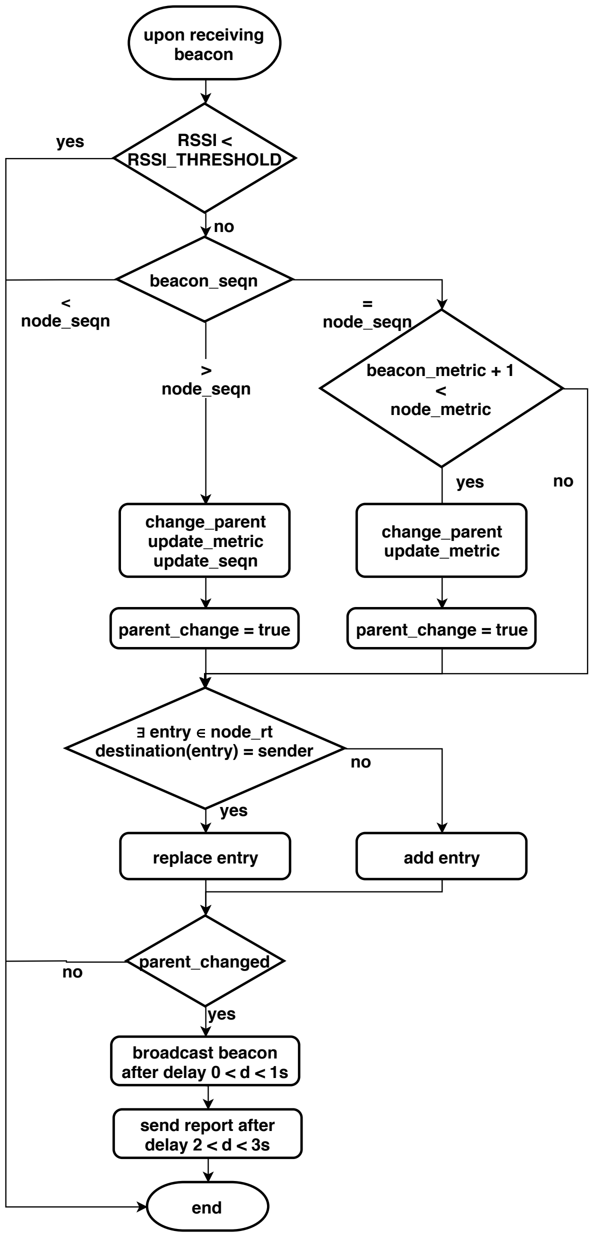
upon receiving beacon
RSSI < RSSI_THRESHOLD
yes
no
beacon_seqn
< node_seqn
= node_seqn
> node_seqn
beacon_metric + 1 < node_metric
yes
no
change_parent update_metric update_seqn
change_parent update_metric
parent_change = true
parent_change = true
∃ entry ∊ node_rt destination(entry) = sender
no
yes
replace entry
add entry
parent_changed
no
yes
broadcast beacon after delay 0 < d < 1s
send report after delay 2 < d < 3s
end

Figure 1. Flowchart describing the actions taken by a node upon a beacon reception.

subtrees before transmitting the report to the parent. This implementation aspect offers room for improvement that would impact positively both on the duty cycle and on the collision probability, at the cost of an increased complexity.

Data packets generated are forwarded to the destination node by looking at the routing table of the node. In case the destination is present, the next node to forward the packet to is retrieved, otherwise the packet is sent to the node's parent. The process is repeated until the correct destination is reached or the packet, once coming to the sink, is dropped due to lack of information on the destination.

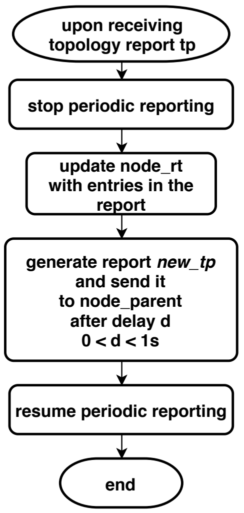
upon receiving topology report tp
stop periodic reporting
update node_rt with entries in the report
generate report *new_tp* and send it to node_parent after delay d 0 < d < 1s
resume periodic reporting
end

Figure 2. Flowchart describing the actions taken by a node upon a topology report reception.

III. Considerations on the implementation

The protocol implementation has been done exploiting three separate channels:

- a broadcast channel for beacons;
- a unicast channel for report messages;
- an additional unicast channel for data packets.

The choice of using an additional unicast channel for topology reports has been done to reduce the chance of collisions between packets exchanged on other channels that would have led to a possible loss of routing information and therefore the possibility of not forwarding data packets correctly. A dedicated message for topology reports has been taken into account due to the size of the routing table, that would hardly fit into a packet header. In addition, by having a separate connection, with dedicated communication callbacks, the overall logic in the packet decoding, and in further actions to be taken, is decreased.

The routing table is defined as a statically allocated array of routing entries, where each entry comprises a destination node, a next-hop node and the timestamp of the most recent update to the entry. The number of entries within the table is therefore fixed. For what concerns the implementation, this size has been made equal to the double of the maximum number of nodes in test files (ten nodes are present in the test files considered). In addition, it has been assumed that —in any case— *one* topology report message is sufficient to contain the node's routing table.

A topology report message contains the same information on the node's routing table, except for timestamps, which are removed to reduce the size of the overall message. When sending a report message, the sender inserts itself to the message routing table, in order to communicate its presence to the parent and provide a communication path to nodes in its subtree.

Routing table entries are updated whenever a new topology report is received. Each routing entry has an expiration time, expressed in clock cycles, after which the entry is considered obsolete and it is therefore required to be removed. Expired entries are removed right before sending a new topology report, which means that the routing table is cleaned at least at each tree-building phase, when a beacon with a more recent sequence number is received.

The choice of the expiration time for routing entries plays a crucial role for an acceptable protocol performance. A small value leads to missing information when routing a packet, and therefore its dropping. A large value instead means that paths to unreachable nodes could be taken into account when forwarding a packet.

In the implementation, the expiration time is set to be slightly bigger than twice the interval between two consecutive tree-building phases. This allows each node to tolerate one packet collision in the building phase without affecting the protocol performance.

Data packets are associated to a header defining the destination and the source nodes, together with the number of hops encountered in the forwarding. To put a limit on the amount of time a given packet is allowed to navigate within the network, a maximum number of hops the packet is allowed to be relayed is set (stated by `MAX_HOP_COUNT`), after which the packet is dropped.

IV. Protocol performance

To measure the protocol performance, two sets of simulations have been performed. In the first one, the protocol has been tested without nodes being able to turn off their wireless transceivers. In the second one, the ContikiMAC mechanism has been enabled.

In both scenarios, twenty runs on the file `test_nogui_-dc.csc`, with random seed, have been made. Given all runs, the reported values on the overall packet loss rate (PDR) and duty cycles (DC) have been averaged. These values are shown in Table I.

Table I
PDR and DC values using nullRDC and ContikiMAC

	PDR (%)	DC (%)
nullRDC	99.45	100
ContikiMAC	93.07	2.38

V. Conclusions

In this report it has been described a protocol for any-to-any routing exploiting a tree hierarchy for wireless sensor networks. Considerations on the given architectural choices and on particular parameters' values have been detailed. Finally a set of statistics describing the overall protocol behaviour has been presented.

Appendix A

Additional details on the implementation are presented in this section.

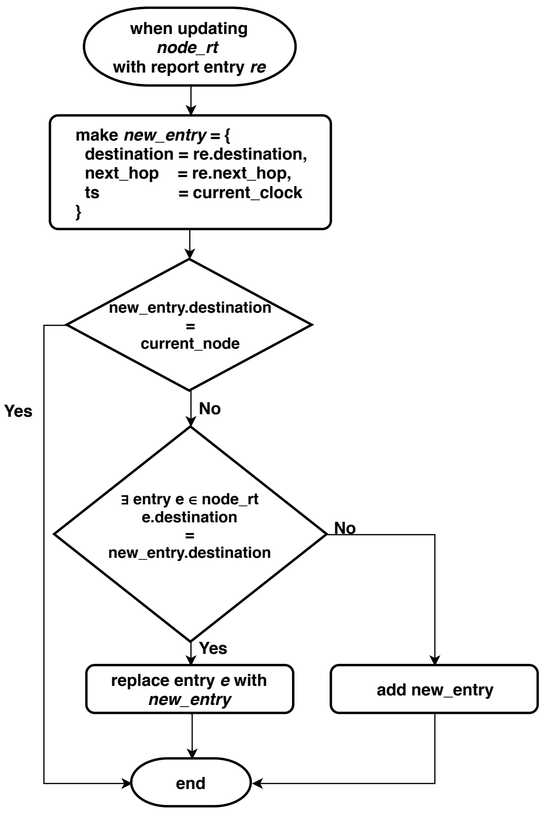
when updating *node_rt* with report entry *re*
make *new_entry* = { destination = re.destination, next_hop = re.next_hop, ts = current_clock }
new_entry.destination = current_node
Yes
No
∃ entry e ∈ node_rt e.destination = new_entry.destination
No
Yes
replace entry *e* with *new_entry*
add new_entry
end

Figure 3. The flowchart describes the set of actions performed when adding topology report entries to the node's routing table.

References

[1] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012.