IoT 2024 Project Report: Any-cast

Patrick Pastorelli 247440

February 2025

Contents

1 Introduction

In this project, we extended the converge-cast protocol developed during Lab 6 to enable any-cast communication between nodes. The project builds upon the original routing protocol, enhancing it and providing the necessary tools for network recovery and routing between nodes in different sub-trees. The resulting protocol is tested in Cooja to evaluate Packet Delivery Ratio (PDR), duty cycle, and latency in a network of 10 nodes. Additionally, a PDR analysis is conducted on the UniTN testbed to assess performance in real-world scenarios, specifically in networks of 16 and 36 nodes. The routing was tested with both ContikiMAC and nullrdc as link layers.

2 Protocol details

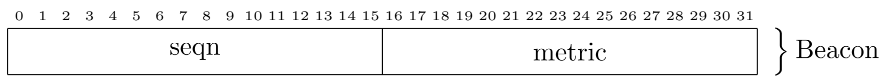
The proposed protocol has the following features:

- Periodic beaconing for routing tree construction
- Recovery beaconing to address inconsistencies or isolation in network nodes
- Maintenance of routing tables to support any-cast communication
- Periodic topology reports to maintain upward routes, with optional piggybacking of information to reduce control traffic
- Loop detection through hop count and early identification of loops via routing inconsistencies
- Two selectable routing metrics: hop count and link quality indicator (utilizing the same strategy as MultiHop LQI)

2.1 Protocol messages

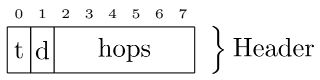
The protocol offers different messages for unicast and broadcast communication.

Broadcast The broadcast messages are small beacon messages that include, similar to those in Lab 6, the sequence number and the metric of the sending node. The sequence number is a 16-bit unsigned integer, while the metric can be either a 16-bit unsigned integer or a floating-point number, depending on the metric being used (16-bit integer for hop count and float for LQI). Additionally, beacons can have an optional trailing byte to specify a ”request response” flag. This flag indicates that, even if the beacon would normally be ignored due to an outdated sequence number, the receiving node is still expected to send another beacon in response. This mechanism is used to address local routing inconsistencies identified by the protocol. Here’s an example of a beacon message using hop count as a metric and without the optional byte:



Unicast The unicast messages, on the other hand, are more complex and serve a dual purpose: carrying traffic data (like topology reports) and application data (the actual anycast exchange). These messages share a common header, which can be extended depending on the specific message type, and they have their own unique payload format.

The common header is formatted as follows:



where:

- t is the type bit, 1 indicates a topology report while 0 indicates a data message
- d is the "downwards" flag, this flags is used only by data messages and indicates that the message has been sent down the sub-tree to reach the designated target.
- hops is the hop count of the messages, if it goes above a certain threshold the packet is dropped as it probably ended up in a loop

Additionally the data message add 4 more bytes including the source and destination address (both 2 bytes each).

For the actual data payloads, the data messages contain the application payload provided by the template, which is of fixed size. In contrast, topology reports include a chain of nodes that notifies the receiving node about the composition of its sub-tree. These chains start with the address of the node that originally sent the update to be propagated upwards, followed by the addresses of nodes that utilized piggybacking to include themselves. This way, the destination node can learn about the contents of its sub-tree.

2.2 Beaconing and routing metrics

Metrics The protocol offers two different routing metrics that can be selected at compile time:

- Hop Count: The number of hops needed for a message to travel from the source to the sink.
- Link Quality Indicator (LQI): This metric indicates the correlation between a received symbol and its decoded version by the radio. To make this metric additive, we utilize a method similar to MultiHop LQI, where the parent metric of a node is defined as follows:

$$m = m_{parent} + \frac{1}{LQI^3} \tag{1}$$

Beaconing Beacons start from the sink which is the one that increases the sequence numbers and are sent periodically (every 60 seconds) to maintain the tree structure and to reflect changes over time. The selected metric is included in the beacons along with the sequence number of the node, allowing other nodes to identify the best available parent and propagate the beacons along with their updated metrics down the tree. The parent node is chosen based on the lowest metric from nodes that have sent a beacon with an up-to-date sequence number.

While this approach ensures the standard creation of trees in stable and reliable networks, it is possible for nodes to become isolated or to receive messages from unknown nodes. To handle recovery in such cases, nodes can send a beacon requesting beacons from neighboring nodes. This method aims to locally fix network issues without generating excessive control traffic or disturbing stable areas of the network. The nodes will send a recovery beacon in the following scenarios:

- When a node fails to receive a beacon within the expected time frame plus a specific buffer time, it will attempt to reconnect to the network by sending a recovery beacon.
- When a loop is detected, either because a packet has exceeded the maximum hop count or because a packet with the "downwards flag" is being forwarded to the parent of the node.

2.3 Topology reports and routing table

Routing Table The nodes maintain a routing table that includes their neighbors and the nodes contained in their sub-tree. Each entry in the table contains the following information:

- The address of the node.
- The address of the next hop to send a message to the node.
- A timestamp of the last time information was sent up the tree.

The timestamp is used to check if the information is stale. Instead of using a timer to clear the table periodically, the nodes simply check the freshness of the information upon retrieval. This is done by subtracting the timestamp from the current clock time; if the difference exceeds a certain limit (e.g., 2 minutes), the entry is removed. The timestamp of an entry is updated when a new topology reports is received or when a data message going upwards is received. This last method is a very good way to gather routing information without adding control overhead.

Topology reports Topology reports are used to construct upwards routes in the tree, by having the nodes notify their parents of choice these messages create the infrastructure needed for anycast. Topology messages are send periodically by each node and propagated upwards until they reach the sink this allows to create a route from the sink to the node using the same hops used by the topology report. If a node is about to send a topology report but it receives one at around the same time (2-3 seconds before the actual message) it can decide to piggyback its own information (i.e. add itself after the source address in the payload) in the same topology report before forwarding it upwards and cancel its next topology report to reduce control traffic.

The protocol sends topology reports periodically with the same rate at which beacons are sent: when there is an update in the sequence number the node will schedule both the beacon propagation and the topology reports (at different times). This inherently allows a node disconnected from the network to also schedule its topology report in after a successful recovery beacon. Additionally nodes send a topology report in the event of an unexpected parent change (i.e. change in the topology after a recovery beacon).

2.4 Collision reduction

To reduce collision the protocol tries to keep messages as small as possible by exploiting the same byte to encode multiple flags and fields. It also adds random slacks to beacons and topology reports timers to reduce the likelihood of collisions during control traffic propagation. Lastly, the period at which beacons are sent, was chosen by testing different values in the simulation to strike a balance between collisions and reliability.

An additional change that was made to improve the performance using ContikiMac was to repeat the first beacon twice and to change the channel check rate from 8 to 16. This might not be directly connected to collisions but simulation has shown that ContikiMac tends to lose the first beacon easily, this is probably due to the node not yet knowing the wake up time of neighbors, by putting an higher check frequency and a bit of redundancy at the start the problem was mitigated and performance increased (the first tree construction is more reliable).

2.5 Scrapped Ideas

During the development of the protocol, some approaches were changed because they yielded worse results than those achieved with the proposed solution. The first idea to be discarded was the inclusion of a "pull" bit in the unicast header. This bit was intended to request a node to send a new topology report when an ancestor detected stale information about a node in the sub-tree. However, this approach produced poorer results than a recovery beacon, likely due to the increased traffic during the periodic application message exchange, which led to more collisions. While it was possible to wait a bit longer before sending a pull request, excessive waiting would result in a new beacon being sent by the sink, achieving the same outcome that the pull bit would have produced but without increasing traffic.

Another idea that was scrapped involved a different approach for the topology reports. In the original approach, these reports were not meant to be forwarded all the way up to the sink but were instead intended to stop at the parent node. This meant that nodes would have to include their entire routing table in the message to gradually construct the tree from the ground up. However, this approach faced significant challenges. Ideally, the leaf nodes would need to be the first to send updates, allowing higher-level nodes to send only one report containing the entire sub-tree. However, achieving the right timing is difficult: if a node waits too long, the latency of building the tree may increase; if it does not wait, it may have to send multiple reports due to new information arriving after the initial report, resulting in even more traffic than the chosen approach. Overall, in the simulation, this approach led to more collisions and worse results compared to the one that was ultimately used.

3 Protocol evaluation

The protocol was evaluated on Cooja with a network of 10 nodes and the UniTN testbed on both a network of 16 nodes and 36 nodes. The tests duration was, in both cases, 30 minutes.

3.1 Simulation results

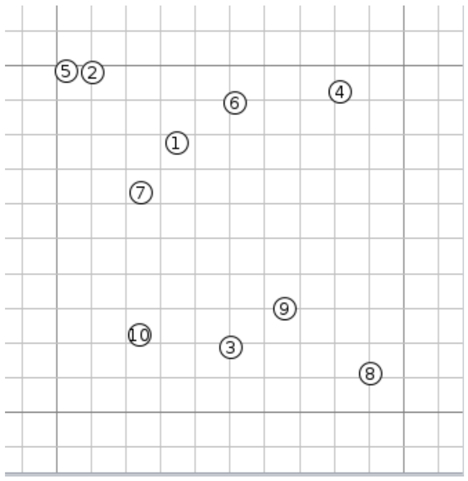


Figure 1: Cooja simulated network, sink: 1

Using Cooja (fig 1) the average PDR, duty cycle and latency of the protocol were evaluated, the results were as following:

MAC	Metric	PDR (%)	latency (ms)	duty cycle (%)
NullRdc	hop count	99 - 100	15 - 18	100
NullRdc	LQI	99 - 100	15 - 18	100
ContikiMac	hop count	96.5 - 98.5	100 - 150	2.0 - 2.5
ContikiMac	LQI	97 - 98.5	100 - 150	2.0 - 2.5

From the results we can see that ContikiMac performs really well considering that it has low duty cycle and loses only about 2% PDR (most likely due to same nodes having out of sync wake ups or collisions). The latency increases but this is to expected since the radio is being duty cycled. Lastly we can see that, while with nullrdc the two metrics don't seem to matter, with ContikiMac the minimum average increases a bit for LQI; this is to be expected as LQI is a more reliable metric compared to hop count which doesn't take into account node quality.

3.2 Testbed Results

On the testbed only PDR and duty cycle were evaluated, the two networks (fig. 2) were respectively node from 11 to 26 and from 1 to 36; in both cases the sink was node 11.

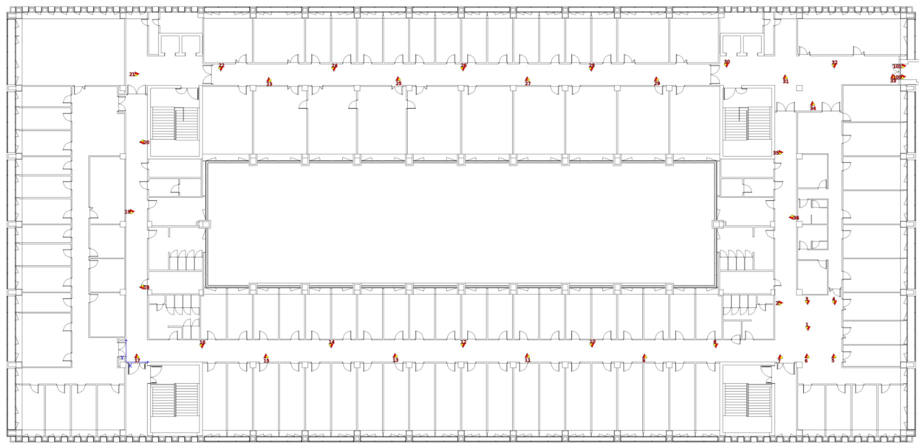


Figure 2: TestBed topology

The results were as follows:

MAC	Metric	#nodes	PDR (%)	duty cycle (%)
NullRdc	hop count	36	93	100
NullRdc	hop count	16	92	100
ContikiMac	hop count	36	82	3.2
ContikiMac	LQI	36	82	3.2
ContikiMac	hop count	16	86	3.2
ContikiMac	LQI	16	89	2.1

As we can see, compared to the simulation, the results worsen a bit, this is understandable as the number of nodes increases and the topology is more complex. In particular the overall PDR decreases also due to some nodes having a particularly hard time delivering messages probably because of their position in the network (for example node 25 and 4 had low PDRs in some test, decreasing the average value). Additionally, in real world scenario, communication between nodes is not reliable and messages can be lost due to ambient interference or environmental conditions. It is also worth noticing that those tests were done in the afternoon when the environment is more crowded.

To compare the two MACs, as expected, nullrdc performs better than ContikiMac at the cost of the radio being always on, despite this ContikiMac still offers reasonable results compared to its counterpart considering that the radio is duty cycled. Of course, as the number of nodes increases, the performance deteriorates a bit but nothing too major. Lastly, regarding the two metrics, LQI seems to perform a little better with 16 nodes but performed about the same in the 36 nodes test. However, such small changes in PDR might also be due to the different states of the environment during the tests.

4 Conclusion

After the protocol evaluation I think that LQI could be a better metric choice as it performed better than hop count in some cases and about the same in others. ContikiMac seems to work well with the proposed protocol as it has acceptable reliability considering the very low duty cycle obtained and has manageable latency (only about 150ms in the average case).