Wyres

LoWAPP Validation Document

2016

Nathan Olff
08/12/2016

This document

Objectives

Versions

Version	Author	Date	Changes
V0.1	N.Olff	03/10/2016	Creation
V0.2	N.Olff	05/12/2016	Update with the latest version of the core
V0.3	N.Olff	08/12/2016	Update with the latest version of the core

Validation report

Test cases

Several cases will be tested, both on the simulation environment and on the real hardware.

Statistics reports will be joined to some test case results. These are used to simulate the energy efficiency of the LoWAPP protocol.

Be careful though not to take the CPU numbers for granted as the simulation was running on an Intel CPU running at several GHz in contrast to the few MHz of the STM32L1 microcontroller !

AT configuration commands

Test case A-: Send AT configuration commands to a node.

- A-1: AT+GROUPID=groupId, AT+DEVICEID=deviceid, AT+GWMASK=gwMask, AT+CHANID=chanid, AT+TXDR=txDatarate, AT+PTIME=preambleTime and AT+ENCKEY=enckey : set configuration values
- A-2: AT+GROUPID, AT+DEVICEID, AT+GWMASK, AT+CHANID, AT+TXDR, AT+PTIME and AT+ENCKEY: get configuration values
- A-3: AT&W: write current configuration into persistent memory

Test case A-1 : Checked by sending a series of AT SET commands and doing an AT&V between each other to check that the value is changed. We also send wrong AT SET commands and verify that the value is not modified. In addition to verifying the output of the previous and the following AT&V, we also check that the AT SET returns "OK" or "NOK". All seven correct commands returned OK, were processed and all seven invalid commands returned NOK and were rejected.

Test case A-2 : Same process used as A-1 test case but with AT GET commands. Six commands returned OK and the correct value. AT+ENCKEY returned an error message notifying the user that the encryption key cannot be displayed, which is the expected behaviour. Invalid commands returned NOK error messages.

Test case A-3 : Two versions of this test case exists.

- A-3a : The first executes a series of SET command, then an ATZ to reboot the device and checks the output of an AT&V before and after reboot.
- A-3b : The second checks the actual configuration files. It executes a set a AT set command followed by an AT&W (AT WRITE) and checks the content of the configuration file. A second set of AT set commands are executed to reset the configuration file to its original state.

Both versions have been run successfully.

AT Validation commands

Test case B-: Send AT validation commands to a node

- B-1: AT&V: print current configuration as json
- B-2: AT+SELFTEST: perform hardware selftest
- B-3: AT+WHO: get RSSI of the recently seen nodes
- B-4: AT+PING: send ping packet to a node and wait for the ACK

- B-5: AT+HELLO: send a hello packet. Responses happen when remote nodes decide to reply

Test case B-1 : AT&V is already being used through the A-* test cases. In order to be sure of its behaviour, a manual check is also done. The output looks like:

```
OK
{"chanId":"00","txDatarate":"07","bandwidth":"0","coderate":"1","power":"14"
,"gwMask":"00000000","deviceId":"01","groupId":"0037","pTime":"01000"}
```

Test case B-2 will not be implemented in the simulation because the hardware selftest already exists.

Test case B-3 : Two messages are sent from two different nodes to a third node. Executing an AT+WHO command on the receiver node displays a json object containing the timestamp and rssi value from the last messages it received. Although RSSI values are not reported in the simulation, we can still see that the two devices are present in the json objects and that two different timestamp were recorded.

Test case B-4 : Send AT+PING commands to 2 different devices. Both ping are received and acknowledged by the corresponding receivers. This has been run successfully.

Test cases B-5 has not been implemented. Use AT+SEND instead for now

AT Operation commands

Test case C-: Send AT operation commands to a node

- C-1: AT+SEND: Send a message to a node and wait for the ACK
- C-2: AT+POLLRX: Check for received packets since last check
- C-3: AT+PUSHRX: Change to push mode
- C-4: AT+DISCONNECT: Stop listening for packets and disable transmissions.
- C-5: AT+CONNECT: Start listening for packets and allow transmissions.

C-1 : 10 consecutive send command, every 3 seconds. Test automated as Tests/test-send-1. All 10 messages have been received and acknowledged by the receiver node.

C-2 and C-3 : Both test cases are checked in one automated test called C-2. Execute 3 SEND commands, execute an AT+POLLRX on receive node and send another 2 SEND commands. Change the mode of the receiver to PUSH mode and send 3 other packets. We have checked manually that :

1. The first three packets are grouped together and returned as a response to AT+POLLRX
2. The 2 following packets were returned when the change of mode were made with AT+PUSHRX
3. Each following packet was returned on reception individually. We can see from the logs that the node was in the process of sending ACK when it returned the packets.

C-4 and C-5 : By default, the node is connected. Tests/test-send-3 disconnects and reconnects the receiver node while the transmitter continue to send packets. Automatic checks are done using the parser-missing.py script for checking that missing frames are detected on both sides of the transmission. Disconnection on the transmitter side made all following send fail until re-connection. This test case was successfully checked.

Use cases

Test case D-: Check the general functionality of the protocol.

- D-1: Send a message from one node to another (AT+SEND), both in the same group. The receiver is in **pull** mode.
- D-2: Send a message from one node to another, both in the same group. The receiver is **push** mode.
- D-3: Send a message to a node that is either not existing or out of range.
- D-4: Send a message to a node on another group (should not receive it).
- D-5: Send a message to a node B in the same group. We will check with a third node C on another group, same channel, with the same device id of node B.
- D-6: Send a broadcast message with nodes in and out of the group. Check that the nodes out of the group do not receive the message.

D-x.b : Same test cases but with a group filled with 250 nodes.

D-1 : Automated with Test/test-send-D1. A node sends three messages to an other node, which executes an AT+POLLRX after that. This is done four times. The test checks that the receiver got the 12 messages and displays them as response to AT+POLLRX, grouped by three messages. This test case was successfully tested.

D-2 : Automated testing with Tests/test-send-D2 where a node sends a few packets, then the receiver move from pollrx to pushrx mode. The first received packets are displayed in JSON when moving from PULL to PUSH mode. This test case was successfully tested: All 12 messages were received and 11 JSON were returned (1 containing 2 messages when the change from pull to push mode were made).

D-3 : Automated testing with Tests/test-send-D3 where a node send a packet to an non-existing node and a disconnected node. The disconnected node does not receive anything and no ACK is received by the transmitter. This test was successfully tested.

D-4 : Automated with Tests/test-send-D4 where several nodes from different groups (with different encryption keys) try to communicate. We can see on the parser report that no message were received and that each node detected the CRC failures. This test was successfully tested.

D-5 : Send a message to a node 2 in the same group, while another node of id 2 from another group is running. The other node 2 should fail to check the CRC of this message. A second try is done by sending to the node 2 in the second groups and making sure the one in the first group doesn't receive it. This test case was successfully tested.

D-6 : Send a broadcast message on a network of 3 nodes. The 2 non-transmitter nodes should receive without sending an ACK. This test was successfully tested.

Performance testing

Test case E-: Performance testing. Activity (time processing vs sleep time) and efficiency (percentage of message lost) of each nodes will be measured.

- E-1: Application send unicast messages to X nodes, waiting for ACK each time

    - E-1a: 10 nodes
    - E-1b: 50 nodes
- E-2: Application send unicast messages to X nodes, not waiting for ACK (burst tx requests).
    - E-2a: 10 nodes
    - E-2b: 50 nodes
- E-3: Application send X messages to the same node (burst rx).
    - E-3a: 10 messages
    - E-3b: 50 messages
- E-4: Test transmission with a preamble length of
    - E-4a: 1000 ms
    - E-4b: 500 ms
    - E-4c: 2000 ms
    - E-4d: 5000 ms
- E-5: Send one message to a specific node every X seconds and check no message is lost:
    - E-5a: 1 s
    - E-5b: 5 s
    - E-5c: 30 s
    - E-5d: 60 s
    - E-5e: 120 s
- E-6: Send broadcast message every X seconds:
    - E-6a: 1 s
    - E-6b: 5 s
    - E-6c: 30 s
    - E-6d: 60 s
    - E-6e: 120 s
- E-7: Sending messages from several nodes at the same time (AT+SEND at the same time, processing should be delayed)
    - E-7a: 2 nodes at the same time
    - E-7b: 3 nodes at the same time
- E-8: X nodes send messages to other nodes every Y seconds with each node having a different interval
    - E-8a: 10 nodes
        - E-8a1: Between 3 and 10 s
        - E-8a2: Between 10 and 20 s
        - E-8a3: Between 30 and 60 s
    - E-8b: 50 nodes
        - E-8b1: Between 3 and 10 s
        - E-8b2: Between 10 and 20 s
        - E-8b3: Between 30 and 60 s
    - E-8c: 100 nodes
        - E-8c1: Between 3 and 10 s
        - E-8c2: Between 10 and 20 s
        - E-8c3: Between 30 and 60 s
    - E-8d: 250 nodes
        - E-8d1: Between 10 and 20 s

- E-8d2: Between 30 and 60 s
- E-8d3: Between 60 and 90 s

E-1 : Sending unicast packets to several nodes, one at a time, waiting for an ACK between each send request. This has been tested with 10 (E-1a) and 50 (E-1b) receiver nodes.

E-2 : Sending unicast packets to several nodes in burst (every 500ms), without waiting for an ACK. The messages are put into a queue for transmitting later. This has been tested successfully with 10 receiver nodes. It takes about 45 seconds to send the 10 messages as each send blocks the transmission for some time. Here are the statistics of this test :

```
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
f64478ef    54326 ms (90.62%)    69 ms (0.11%)    5730 ms (9.56%)    41 ms (0.07%)
5c6fc5b9    54277 ms (90.54%)    84 ms (0.14%)    5768 ms (9.62%)    41 ms (0.07%)
91c9621e    54309 ms (90.59%)    75 ms (0.12%)    5742 ms (9.58%)    41 ms (0.07%)
471d2156    54304 ms (90.59%)    71 ms (0.12%)    5752 ms (9.59%)    41 ms (0.07%)
b5a731aa    54303 ms (90.58%)    66 ms (0.11%)    5759 ms (9.61%)    41 ms (0.07%)
1a04fe34    54303 ms (90.58%)    52 ms (0.09%)    5774 ms (9.63%)    41 ms (0.07%)
3f26c561    44227 ms (73.78%)    86 ms (0.14%)    5318 ms (8.87%)    10533 ms
(17.57%)
0af6f826    54299 ms (90.58%)    63 ms (0.10%)    5770 ms (9.63%)    40 ms (0.07%)
dfed7891    54329 ms (90.63%)    86 ms (0.14%)    5707 ms (9.52%)    41 ms (0.07%)
26f3b8df    54317 ms (90.61%)    66 ms (0.11%)    5743 ms (9.58%)    40 ms (0.07%)
81b8648b    54313 ms (90.60%)    91 ms (0.15%)    5720 ms (9.54%)    40 ms (0.07%)
```

With 50 nodes however, this seems to be more problematic : Only 23 out of 50 messages are actually sent and acknowledged. This is because the tx queue gets filled after 16 transmission requests are done within a few seconds (burst requests). When the queue is full, send requests are lost. LoWAPP is not aimed at sending large quantity of messages over a short period of times, this is therefore an expected behaviour.

E-3 : Sending several packets to the same node as burst (every 500ms). With 10 messages, all are received and acknowledged. It takes about 60 seconds to for the 10 full transmissions to occur.

For the 50 messages test, the same problem as in E-2 occurs where only 23 messages are sent. In addition to that, the receiver needs to be put in PUSH so that we are not limited by the size of the RX queue (16 messages). With this in mind, all 23 packets sent are received and acknowledged.

E-4 : Set the preamble to about X ms and send a few messages to a node. A preamble of 1000 ms is the default value for preamble so this is similar to any previous send test. Here are the results with a preamble of around 1000 ms, over a 13 seconds run :

```
Device        CPU_SLEEP        CPU_ACTIVE
3f26c561    16981 ms (99.97%)    5 ms (0.03%)
91c9621e    16985 ms (100.00%)    1 ms (0.00%)
1a04fe34    16986 ms (100.00%)    1 ms (0.00%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
3f26c561    11278 ms (66.39%)    7 ms (0.04%)    2123 ms (12.50%)    4212 ms (24.80%)
91c9621e    14350 ms (84.48%)    8 ms (0.05%)    3102 ms (18.26%)    162 ms (0.95%)
```

```
1a04fe34    14512 ms (85.43%)    6 ms (0.04%)    3104 ms (18.27%)    0 ms (0.00%)
```

With a preamble of about 500ms, over a 20 seconds period :

```
Device        CPU_SLEEP        CPU_ACTIVE
3f26c561    17069 ms (99.96%)    6 ms (0.04%)
1a04fe34    16976 ms (100.00%)    1 ms (0.00%)
91c9621e    16776 ms (99.99%)    1 ms (0.01%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
3f26c561    12852 ms (75.27%)    20 ms (0.12%)    2126 ms (12.45%)    2219 ms (13.00%)
1a04fe34    16109 ms (94.88%)    15 ms (0.09%)    1094 ms (6.45%)    0 ms (0.00%)
91c9621e    15142 ms (90.25%)    15 ms (0.09%)    1898 ms (11.31%)    162 ms (0.97%)
```

For the 2000ms and 5000ms preamble, we had to increase the time between each send request. Otherwise, the transmission would get delayed thanks to the blocking mechanism and all transmission would not be processed before the end of the test. For the 2000ms preamble, we ran the test over a 40 seconds period :

```
Device        CPU_SLEEP        CPU_ACTIVE
3f26c561    40984 ms (99.99%)    6 ms (0.01%)
1a04fe34    41388 ms (100.00%)    1 ms (0.00%)
91c9621e    41187 ms (100.00%)    1 ms (0.00%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
3f26c561    31661 ms (77.24%)    23 ms (0.06%)    2125 ms (5.19%)    8214 ms (20.04%)
1a04fe34    39318 ms (95.00%)    15 ms (0.04%)    2691 ms (6.50%)    0 ms (0.00%)
91c9621e    38348 ms (93.10%)    18 ms (0.04%)    3496 ms (8.49%)    162 ms (0.39%)
```

We chose to run the test case on a period of about 80 seconds. Here are the results :

```
Device        CPU_SLEEP        CPU_ACTIVE
1a04fe34    81390 ms (100.00%)    1 ms (0.00%)
3f26c561    80984 ms (99.99%)    6 ms (0.01%)
91c9621e    81189 ms (100.00%)    2 ms (0.00%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
1a04fe34    79350 ms (97.49%)    15 ms (0.02%)    2672 ms (3.28%)    0 ms (0.00%)
3f26c561    59684 ms (73.69%)    15 ms (0.02%)    2123 ms (2.62%)    20214 ms (24.96%)
91c9621e    78385 ms (96.54%)    16 ms (0.02%)    3473 ms (4.28%)    162 ms (0.20%)
```

All 4 messages sent by 3f26 to 91c9 were received and acknowledged in E-4a, b, c and d test cases.

E-5 : We send 10 messages, one every X seconds to a specific node. After sending all the AT send commands, we need to wait some time for them to be processed from the transmit queue.

Sending 10 messages, one every 1s takes about 45 seconds for the transmissions to complete. When entering Idle state, we process the AT commands received. For every AT+SEND command, we add the message to the tx queue. If the queue already contains at least one element, we answer to the send request with a "SEND DELAYED" message to indicate the message will be sent later. 9 messages out of 10 are getting delayed when sending every 1 second. Here are the statistics retrieved from this test case :

```
Device        CPU_SLEEP        CPU_ACTIVE
91c9621e    68939 ms (100.00%)    3 ms (0.00%)
3f26c561    68926 ms (99.98%)    15 ms (0.02%)
1a04fe34    68940 ms (100.00%)    2 ms (0.00%)
```

```
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
91c9621e    62631 ms (90.85%)    94 ms (0.14%)    6647 ms (9.64%)    402 ms (0.58%)
3f26c561    53836 ms (78.09%)    96 ms (0.14%)    5307 ms (7.70%)    10534 ms (15.28%)
1a04fe34    63046 ms (91.45%)    83 ms (0.12%)    6646 ms (9.64%)    0 ms (0.00%)
```

As one message takes about 3 seconds to transmit (including ACK reception), waiting 5 seconds between each command is enough for every command to be processed directly (even taking into account the blocking period after a transmission). No send request is therefore delayed. Here are the statistics :

```
Device        CPU_SLEEP        CPU_ACTIVE
1a04fe34    49956 ms (100.00%)    2 ms (0.00%)
91c9621e    49956 ms (100.00%)    2 ms (0.00%)
3f26c561    50069 ms (99.97%)    14 ms (0.03%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
1a04fe34    42601 ms (85.27%)    24 ms (0.05%)    8141 ms (16.30%)    0 ms (0.00%)
91c9621e    42273 ms (84.62%)    42 ms (0.08%)    8037 ms (16.09%)    404 ms (0.81%)
3f26c561    34857 ms (69.60%)    31 ms (0.06%)    5320 ms (10.62%)    10547 ms (21.06%)
```

The time 3f26's radio is in receive mode represents the time the node is receiving (or waiting for) ACK.

Sending a message every 30 seconds is similar to sending every 5 seconds as we leave enough time for the node to process, send and receive ACK between each send request.

```
Device        CPU_SLEEP        CPU_ACTIVE
91c9621e    300732 ms (100.00%)    7 ms (0.00%)
3f26c561    300719 ms (99.99%)    18 ms (0.01%)
1a04fe34    300733 ms (100.00%)    7 ms (0.00%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
91c9621e    293935 ms (97.74%)    685 ms (0.23%)    5924 ms (1.97%)    403 ms (0.13%)
3f26c561    284480 ms (94.59%)    624 ms (0.21%)    5309 ms (1.77%)    10533 ms (3.50%)
1a04fe34    294452 ms (97.91%)    575 ms (0.19%)    5922 ms (1.97%)    0 ms (0.00%)
```

Here are the statistics for sending a message every minute :

```
Device        CPU_SLEEP        CPU_ACTIVE
1a04fe34    600474 ms (100.00%)    12 ms (0.00%)
91c9621e    600469 ms (100.00%)    13 ms (0.00%)
3f26c561    600457 ms (100.00%)    24 ms (0.00%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
1a04fe34    595022 ms (99.09%)    1259 ms (0.21%)    4898 ms (0.82%)    0 ms (0.00%)
91c9621e    594528 ms (99.01%)    1349 ms (0.22%)    4897 ms (0.82%)    404 ms (0.07%)
3f26c561    583943 ms (97.25%)    1390 ms (0.23%)    5308 ms (0.88%)    10536 ms (1.75%)
```

And those for sending a message every 2 minute :

```
Device        CPU_SLEEP        CPU_ACTIVE
1a04fe34    1200927 ms (100.00%)    21 ms (0.00%)
3f26c561    1200906 ms (100.00%)    37 ms (0.00%)
91c9621e    1200922 ms (100.00%)    22 ms (0.00%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
1a04fe34    1193732 ms (99.40%)    2617 ms (0.22%)    5132 ms (0.43%)    0 ms (0.00%)
3f26c561    1183039 ms (98.51%)    2589 ms (0.22%)    5315 ms (0.44%)    10535 ms (0.88%)
```

```
91c9621e    1193147 ms (99.35%)    2796 ms (0.23%)    5133 ms (0.43%)    404 ms (0.03%)
```

As we can see from those test case, the more we wait between each request, the more percentage of time we spend in sleep mode (both CPU and radio).

E-6 : Send 10 broadcast messages, one every 1 second. All messages are received by the other nodes in the group. Just like with the E-5a test case, most of the messages (9 out of 10 here) are getting delayed because the transmission takes longer than 1 second. As a broadcast message does not expect acknowledge they are however faster to send than standard unicast messages. Here are the statistics :

```
Device        CPU_SLEEP        CPU_ACTIVE
1a04fe34    28975 ms (100.00%)    1 ms (0.00%)
3f26c561    28959 ms (99.95%)    16 ms (0.05%)
91c9621e    28974 ms (100.00%)    1 ms (0.00%)
Device        RADIO_OFF        RADIO_CAD        RADIO_RX        RADIO_TX
1a04fe34    22992 ms (79.35%)    38 ms (0.13%)    6688 ms (23.08%)    0 ms (0.00%)
3f26c561    19153 ms (66.10%)    33 ms (0.11%)    0 ms (0.00%)    10531 ms (36.35%)
91c9621e    22992 ms (79.35%)    39 ms (0.13%)    6687 ms (23.08%)    0 ms (0.00%)
```

Here we can see that the transmitter's radio does not spend even 1 ms in RX. This is because no ACK is expected after an broadcast message is sent.

Sending broadcast packets every 5 seconds does not generate delay messages are 5 seconds is more than transmission time of one message. Here are the statistics obtained for this test case :

```
Device          CPU_SLEEP              CPU_ACTIVE
91c9621e        49957 ms (100.00%)     2 ms (0.00%)
3f26c561        49947 ms (99.97%)      14 ms (0.03%)
1a04fe34        49957 ms (100.00%)     2 ms (0.00%)
Device          RADIO_OFF              RADIO_CAD              RADIO_RX       RADIO_TX
91c9621e        42654 ms (85.38%)      63 ms (0.13%)   8036 ms (16.09%)       0 ms (0.00%)
3f26c561        40151 ms (80.37%)      63 ms (0.13%)   0 ms (0.00%)    10534 ms (21.08%)
1a04fe34        42632 ms (85.33%)      53 ms (0.11%)   8071 ms (16.16%)       0 ms (0.00%)
```

Sending broadcast messages every 30 seconds gives us the following statistics :

```
Device          CPU_SLEEP              CPU_ACTIVE
1a04fe34        300730 ms (100.00%)    5 ms (0.00%)
3f26c561        300717 ms (99.99%)     16 ms (0.01%)
91c9621e        300729 ms (100.00%)    5 ms (0.00%)
Device          RADIO_OFF              RADIO_CAD              RADIO_RX       RADIO_TX
1a04fe34        294294 ms (97.86%)     662 ms (0.22%)  5953 ms (1.98%) 0 ms (0.00%)
3f26c561        289713 ms (96.34%)     660 ms (0.22%)  0 ms (0.00%)    10534 ms (3.50%)
91c9621e        294279 ms (97.85%)     676 ms (0.22%)  5952 ms (1.98%) 0 ms (0.00%)
```

Sending broadcast every 1 minute gives us this :

```
Device          CPU_SLEEP              CPU_ACTIVE
3f26c561        600452 ms (100.00%)    22 ms (0.00%)
1a04fe34        600463 ms (100.00%)    11 ms (0.00%)
91c9621e        600462 ms (100.00%)    11 ms (0.00%)
Device          RADIO_OFF              RADIO_CAD              RADIO_RX       RADIO_TX
3f26c561        589150 ms (98.11%)     1358 ms (0.23%) 0 ms (0.00%)    10534 ms (1.75%)
1a04fe34        594314 ms (98.97%)     1296 ms (0.22%) 5435 ms (0.91%) 0 ms (0.00%)
```

```
91c9621e        594344 ms (98.98%)      1265 ms (0.21%) 5434 ms (0.91%) 0 ms (0.00%)
```

And finally, sending broadcast messages every 2 minutes gives us these statistics :

```
Device          CPU_SLEEP               CPU_ACTIVE
1a04fe34        1200914 ms (100.00%)    20 ms (0.00%)
91c9621e        1200916 ms (100.00%)    21 ms (0.00%)
3f26c561        1200890 ms (100.00%)    41 ms (0.00%)
Device          RADIO_OFF               RADIO_CAD               RADIO_RX        RADIO_TX
1a04fe34        1193374 ms (99.37%)     2700 ms (0.22%) 5519 ms (0.46%) 0 ms (0.00%)
91c9621e        1193434 ms (99.38%)     2630 ms (0.22%) 5527 ms (0.46%) 0 ms (0.00%)
3f26c561        1188344 ms (98.95%)     2714 ms (0.23%) 0 ms (0.00%)    10532 ms (0.88%)
```

E-7 : Sending 2 messages at approximately the same time from two different nodes with two different destinations.

The first node sends its message. The second one listens before sending its message. As it sees the message from the first mode, it returns an "NOK TX RETRY" message and goes into reception mode. After receiving and skipping the ACK window (the two nodes do not communicate with each other but with 2 other nodes), the second node sends its message. Both messages are received and acknowledged.

A second test case E-7b consists in sending 3 messages at the same time from 3 different nodes. One of the messages is sent first. The two other transmitter go into reception mode and try sending their messages after a random time (one preamble + maximum transmission time + ACK slot + random time). All three messages are received and acknowledged.

A third similar test case with 5 messages at the same time has been run. Only 3 out of the 5 messages are transmitted, received and acknowledged. This is because the retry mechanism when an LBT find something is set to 3. After 3 failed LBT, the transmission is cancelled.

```
                Done:   Error:  Tout:   LBT:    Duty:
TX      1->2    1       0       0       0       0
TX      3->4    1       0       0       1       0
TX      5->6    1       0       0       2       0
TX      7->8    0       0       0       2       0
TX      9->10   0       0       0       2       0
TX Ack  2->1    1       0       0       0       0
TX Ack  4->3    1       0       0       0       0
TX Ack  6->5    1       0       0       0       0
```

E-8 : Send messages to an other node on a fixed interval. Every node had a different interval.

For E-8a1, we put 10 nodes in the group with 3 of them sending packets at intervals between 3 second and 10 seconds:

- Node 3f26 sends a message approximately every 3 seconds to device id 2
- Node 1a04 sends a message approximately every 4 seconds to device id 4
- Node 26f3 sends a message approximately every 6 seconds to device id 6

Here we are approaching the limit of the system as some conflicts occur. 7 messages out of 13 are delivered. Only 5 of those acknowledged are received by the transmitters.

For E-8a2, 3 nodes send packets at intervals between 10 and 20 seconds. As the intervals between sends are bigger, there are no more conflicts and all messages are received and acknowledged.

Range testing on hardware

All previous tests were tested on the simulation.

Range performance testing was done on actual hardware using STM32L151 and a Semtech SX1272 LoRa chip.

To validate the possible range for each spreading factor, we used two devices. One was fixed, the other was placed at different points along a straight path. Measurements were done in line of sight situations.

In SF 7, we were able to communicate up to about 250m.

In SF9, communication worked up to about 850m.

Gateway functionality

Test case G-: Gateway related activity

- G-1: A gateway receives a non-gateway specific message
- G-2: A gateway send a non-gateway specific message
- G-3: A gateway receives an outbound gateway message from a node of its group to an external network (IPv4 for example)
- G-4: A gateway receives an inbound gateway message destined to its group
- G-5: A gateway receives a gateway message of a type it cannot handle
- G-6: A gateway receives an intergroup LoWAPP gateway message and forward it to another group on the same channel.
    - G-6b: Same but on another channel

As the gateway functionality is not yet implemented in the protocol, these test cases were not checked.