Course Project: Time Synchronization and Periodic Data Collection

145811 – Low-power wireless networking for the IoT (UniTn)

May 18th , 2021

Sebin Shaji Philip
(sebin.shajiphilip@studenti.unitn.it)

Lab6-7: Implementation of a multi-hop data collection protocol with many-to-one routing (with random delay and without scheduling/ time synchronization.)

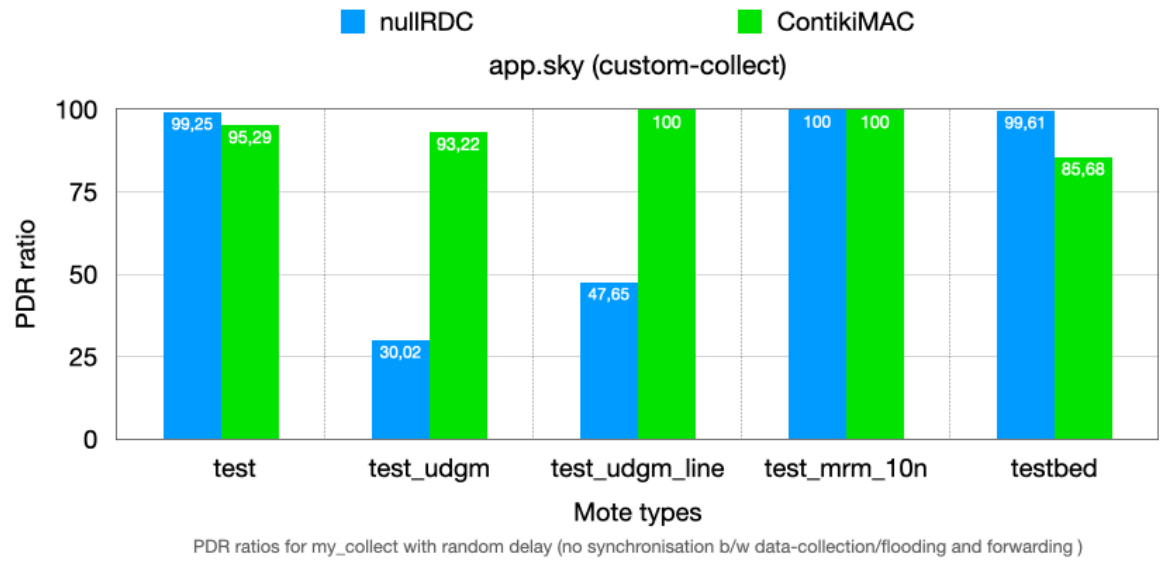


Figure 1: PDR for my_collect

Figure 1 show an increased performance in terms of overall PDR ratio with the use of ContikiMAC, compared to nullRDC. Running the same in testbed gives increased performance with nullRDC.

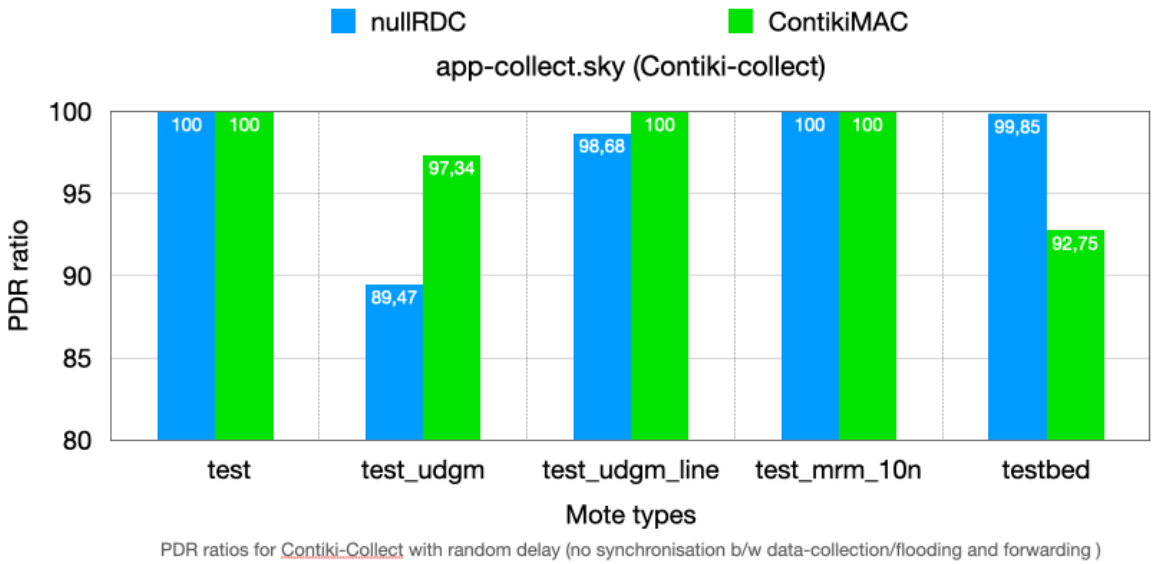


Figure 2: PDR for Contiki-Collect

The same trend (increased PDR for ContikiMAC) can be seen replacing our custom collect layer with the Contiki-Collect. Although the performance of Contiki-Collect is much superior to the custom collect implementation in case of nullRDC.

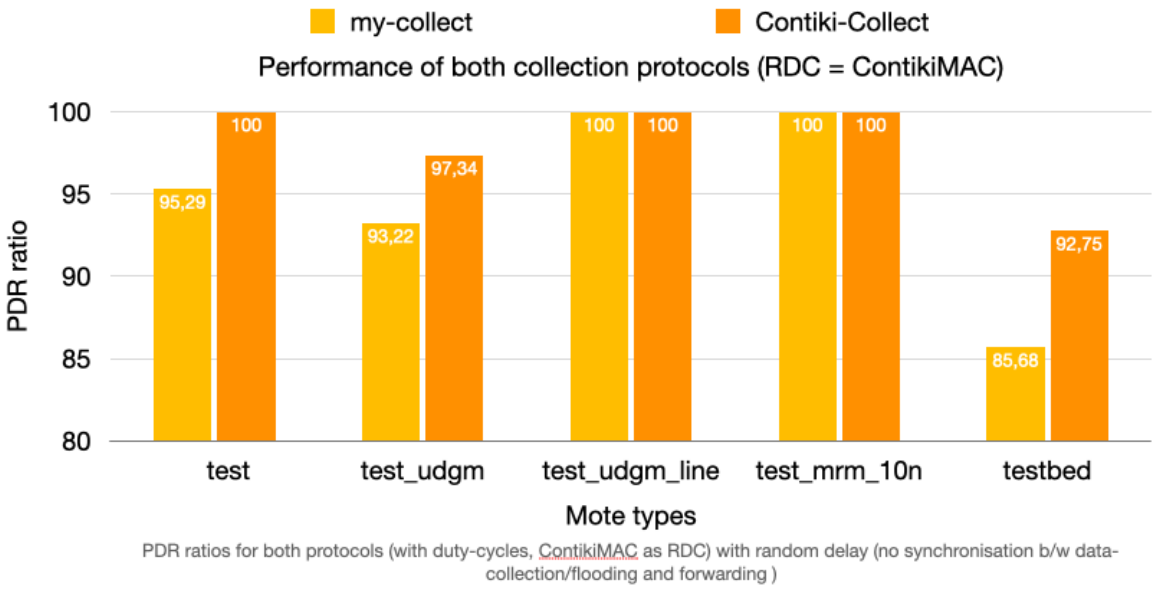


Figure 3: PDR comparison for my_collect vs Contiki-Collect with DC

Figure 3 compares performance of both my-collect with Contiki-Collect with ContikiMAC as the RDC layer. In all the cases the Contiki-Collect performs superior to the custom implementation.

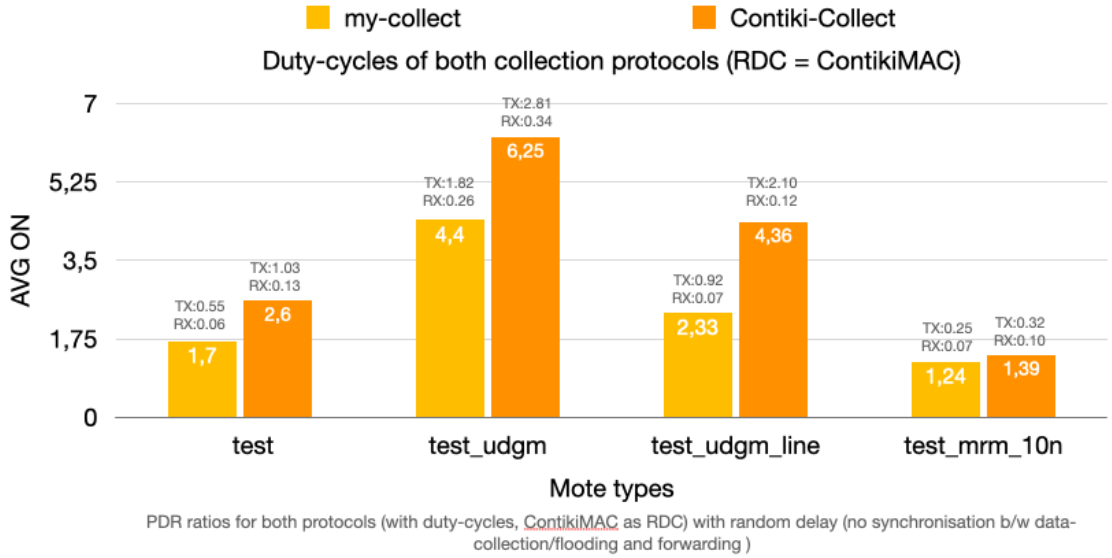


Figure 4: DC comparisons for my_collect vs Contiki-Collect

Figure 4 shows that the increased performance of the Contiki-Collect comes at a price. The average radio-ON time of the Contiki-Collect is slightly higher than the custom implementation, as shown above.

Table 1: Performance evaluation for my_collect

Data collection protocol implementation: app.sky (custom implementation)		
Type	NullRDC	ContikiMAC
test.csc	PDR : 99.25% AVG ON : 100% AVG TX : 0.01% AVG RX : 0.04% AVG INT : 0.06%	PDR : 95.29% AVG ON : 1.70% AVG TX : 0.55% AVG RX : 0.06% AVG INT : 0.14%
test_udgm.csc	PDR : 30.02% AVG ON : 100 % AVG TX : 0.01 % AVG RX : 0.05 % AVG INT : 0.04 %	PDR : 93.22 % AVG ON : 4.40 % AVG TX : 1.82 % AVG RX : 0.26 % AVG INT : 0.41 %
test_udgm_line.csc	PDR : 47.65 % AVG ON : 100 % AVG TX : 0.01 % AVG RX : 0.02 % AVG INT : 0.00 %	PDR : 100% AVG ON : 2.33 % AVG TX : 0.92 % AVG RX : 0.07 % AVG INT : 0.04 %
test_mrm_10n.csc	PDR : 100 % AVG ON : 100 % AVG TX : 0.01 % AVG RX : 0.06 % AVG INT : 0.00 %	PDR : 100 % AVG ON : 1.24 % AVG TX : 0.25 % AVG RX : 0.07 % AVG INT : 0.03 %
testbed	PDR : 99.61% AVG ON : <NA> AVG TX : <NA> AVG RX : <NA> AVG INT : <NA>	PDR : 85.68 % AVG ON : <NA> AVG TX : <NA> AVG RX : <NA> AVG INT : <NA>

The data collected for different radio models and for different RDC layers (nullRDC vs ContikiMAC) are collected into Table 1 (for custom collection layer implementation).

Table 2 lists all the data collected for the Contiki-Collect layer for different radio models and RDC layers.

Table 2: Performance evaluation for Contiki-Collect

Data collection protocol implementation: app-collect.sky (Contiki-Collect)		
Type	NullRDC	ContikiMAC
test_contiki_collect.csc	PDR : 100% AVG ON : 99.98% AVG TX : 0.02% AVG RX : 0.07% AVG INT : 0.09%	PDR : 100% AVG ON : 2.60 % AVG TX : 1.03 % AVG RX : 0.13 % AVG INT : 0.18 %
test_udgm.csc	PDR : 89.47% AVG ON : 100 % AVG TX : 0.07 % AVG RX : 0.28 % AVG INT : 0.21 %	PDR : 97.34 % AVG ON : 6.25 % AVG TX : 2.81 % AVG RX : 0.34 % AVG INT : 0.58 %
test_udgm_line.csc	PDR : 98.68 % AVG ON : 100 % AVG TX : 0.07 % AVG RX : 0.10 % AVG INT : 0.02 %	PDR : 100 % AVG ON : 4.36 % AVG TX : 2.10 % AVG RX : 0.12 % AVG INT : 0.10 %
test_mrm_10n.csc	PDR : 100 % AVG ON : 99.99 % AVG TX : 0.01 % AVG RX : 0.09 % AVG INT : 0.00 %	PDR : 100 % AVG ON : 1.39 % AVG TX : 0.32 % AVG RX : 0.10 % AVG INT : 0.03 %
testbed	PDR : 99.85 % AVG ON : <NA> AVG TX : <NA> AVG RX : <NA> AVG INT : <NA>	PDR : 92.75 % AVG ON : <NA> AVG TX : <NA> AVG RX : <NA> AVG INT : <NA>

Final project implementation: Implementation of a Time synchronized periodic multi-hop data collection protocol with many-to-one routing.

The following MACROs defined in the code is initially selected and its performance in COOJA is evaluated. Based on the analysis from the simulation results and overall performance, different parameters are further tuned/readjusted to get an optimum efficiency (which is presented later in the report). The values marked in Figure 5 are given corresponding to the MACROs in ()

EPOCH_DUARTION : **- The duration of one epoch**
RSSI_THRESHOLD: **- The threshold value above which new nodes will be considered for routing decision.**
BEACON_FORWARD_DELAY: **- The random delay with which each node will re-broadcast the beacon packets for routing.**
MAX_UNICAST_PROCESSING_DELAY: **- It's the product of maximum delay calculated (ceil) for processing a unicast receive callback (just before sending the received packet to its parent) and (MAX_HOP-1). This will be the time required for a unicast packet from the last node (with highest hop count) to reach the sink.**
RADIO_TURN_OFF_DELAY: (b+c) **- This will be the maximum time taken for all the nodes to complete the unicast sending (once the time sync/routing process is completed).**
RADIO_TURN_ON_DELAY: **- This will be time required to turn-on the radio, immediately after which was being turned-off (after completing data-collection). This value was selected by trail-and-error/experimental setup with sub-optimal performance (duty cycle).**
MAX_HOPES: **- The maximum hopes expected in the network**
DATACOLLECTION_COMMON_GREEN_START_DELAY: (a) - This is the time required for a specific node to finish the time-sync (LED blue will be off after this timer expires). This doesn't mean the actual unicast of the specific node begins (which could be performed after a delay, depending upon the node_id).
COLLECTION_SEQUENCE_DELAY: (b) – The delay needed for a specific node to start sending the unicast packet, after the time-sync/routing part is over.

EPOCH_DUARTION : **30*CLOCK_SECOND**
RSSI_THRESHOLD: **-95dB**
BEACON_FORWARD_DELAY: **random_rand() % CLOCK_SECOND**
MAX_UNICAST_PROCESSING_DELAY: **(MAX_HOP -1)*6**
RADIO_TURN_OFF_DELAY: **MAX_NODES*MAX_UNICAST_PROCESSING_DELAY + 200**
RADIO_TURN_ON_DELAY: **EPOCH_DUARTION - (MAX_HOPS*CLOK_SECOND)**
MAX_HOPES: **3**
DATACOLLECTION_COMMON_GREEN_START_DELAY: **(((MAX_HOPS-1)*CLOCK_SECOND) - beacon.delay)**
COLLECTION_SEQUENCE_DELAY:

From Figure 5, g – is the guard time. (d1+d2+d3+d4+ Δ) – time delay experienced by node5 to rebroadcast a beacon packet. Where (d1+d2+d3) is the accumulated calculated delay imposed by parent nodes (2,3 and 4), d4 is the random delay imposed by node5 before retransmitting and Δ is the slight time delay introduced because of radio transmission (TOF) and/or unknown delays from sink to node5. This depends on the hop count as-well.

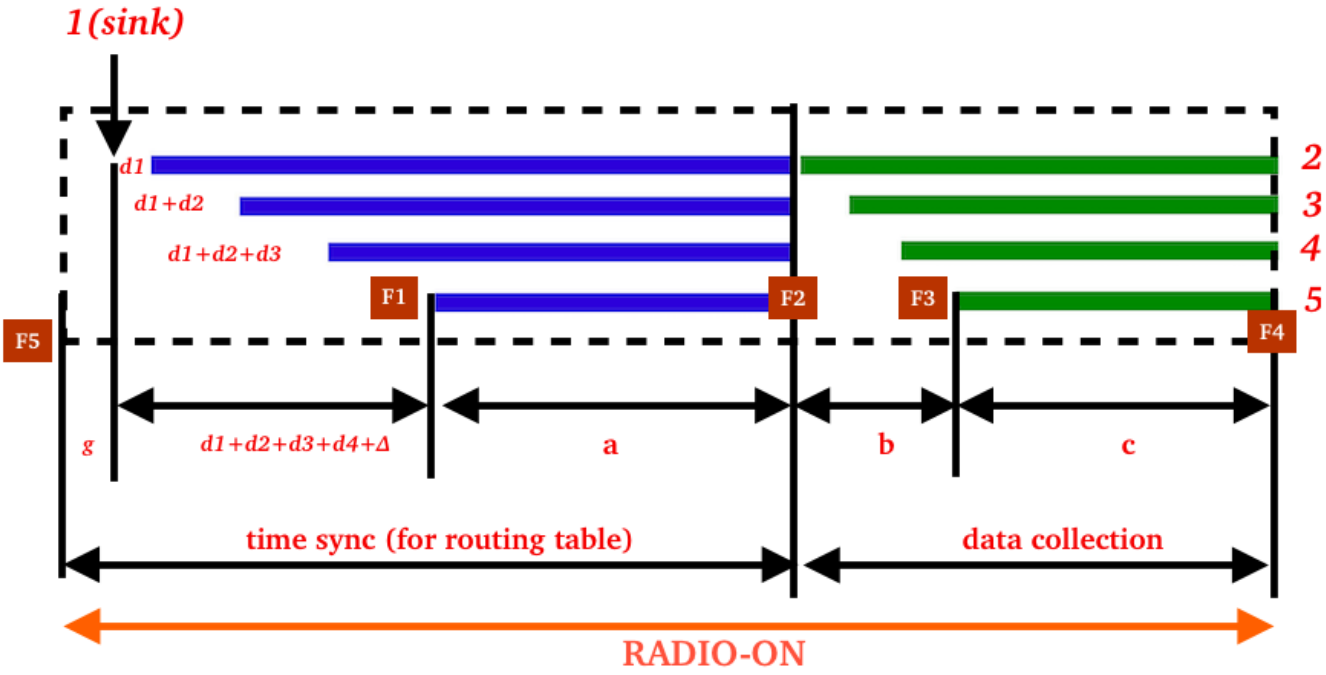


Figure 5: Time synchronized data-collection framework

Figure 5 is marked with red-boxes to indicate the specific function calls (from Figure 6) in the timeline.

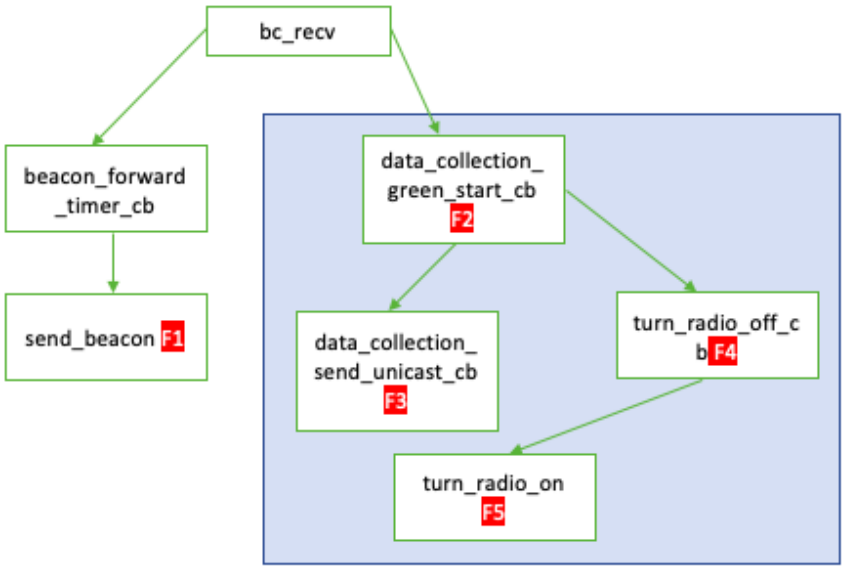


Figure 6: Control flow of a beacon receive callback for a node

Figure 6 explains the generic timer - "call-back" based approach implemented for the time-synchronized data-collection framework. Its clear that all the data-collection phases like ending the time-sync phase of each node (data_collection_green_start_cb with delay **DATACOLLECTION_COMMON_GREEN_START_DELAY**), sending the unicast packet from a specific node (data_collection_send_unicast_cb with delay **COLLECTION_SEQUENCE_DELAY**), turning-off the radio after data-collection (turn_radio_off_cb with delay **RADIO_TURN_OFF_DELAY**), turning radio-on before the next time-sync phase (turn_radio_on with delay **RADIO_TURN_ON_DELAY**) and re-

broadcasting the beacon packet received from the parent (beacon_forward_timer_cb - **BEACON_FORWARD_DELAY**).

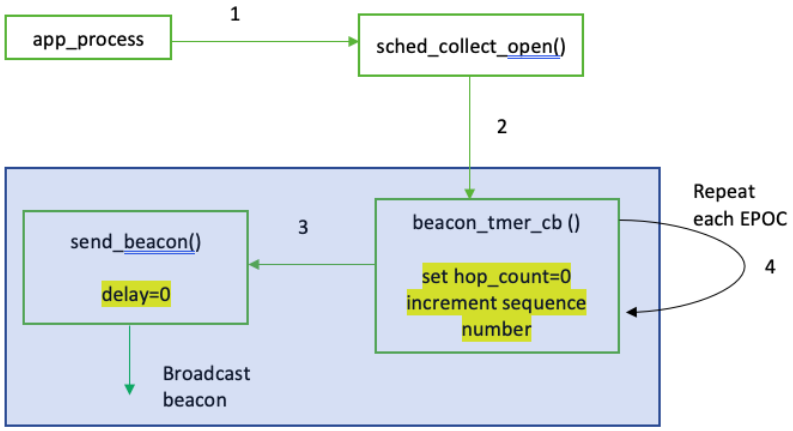


Figure 7:Control flow of a beacon broadcast from sink node

The implementation flow of the beaconing logic from node-1 is represented in Figure 7. In the following section different parameters are explored/tuned and the resulting performance is analyzed to select the optimum values.

I. **RSSI_THRESHOLD**

RSSI_THRESHOLD = -95 dB

test_udgm

PDR : 96.55%

AVG DC : 10.101%

The packet loss of nodes (5,7,9) was (1,14,1) out of 58 tries from each node. Packet from 7 is lost - because of the congestion (8 and 9 sends packets before transmission of 7 got over and thus, after several retries node 3 (common parent) drops 7) - possible solution increase unicast_processing => push further collection sequence delay. (Refer: image)

test_mrm_line

PDR : 94.64%

AVG DC : 10.096%

The packet loss of nodes (7,8,9,10) was (3,4,7,14) out of 58 tries from each node. (Refer images: image1, image2, image3)

(i) 5-4 and 7-6 sequence numbers jumbled-up

5-4: Even though 5 is farther from 4, 4->5, which gives additional delay in hops. Also 5->1 (because somehow weak RSSI -95dB got detected).
Solution: increase RSSI threshold to avoid weaker links, and enforce reliable connection which at least makes sense by distance.

The data collection of 5 starts before 4! (green led- unexpected). Since the data-collection phase starting is misaligned (due to the phase differences in node starting time) 5 starts first have extra advantage. Also the extra sequence delay allocated for 5 is not big enough to mitigate the slack difference. Solution: Increase the collection sequence delay.
7-6: Here both have same parent (5) and there's no misalignment in data-collection start phase (due to slack!). But then again the packets from 7 is successfully

acknowledged first from 5, even-though 6 started sending packets way earlier. This is because of the congestion and collision at 5, the packets from 6 gone unacknowledged/lost before 7. Solution: 5 is common parent to lot of nodes since it has low metric (contributed by the weak link -95 dB). Solution: Increase RSSI threshold.

(ii) packets from 8,9 and 10 lost: Even Though 8 and 10 successfully send packets to 9 (parent), the packet to 5 from 9 (weak link -95dB) does not get through either because of congestion and packet loss.

RSSI_THRESHOLD: -91 dB and MAX_HOPES = 4

test_udgm

PDR : 90.61%

AVG DC : 14.945%

Refer images: image1, image2

The PDR efficiency decreased and DC increased in this case. The packet loss in (7,8) is (25,18) which is substantial. This is because of the congestion in the network. Other higher nodes start sending packets before the unicast of lower nodes gets over and the packets are lost. Also higher nodes have increased DC.Solution: Increase unicast processing time, such that higher nodes will only start sending once lower packets are received at sink(node 1).

Test_mrm_line

PDR : 99.62%

AVG DC : 16.425%

Effect of increasing RSSI: Increased settling time (frequent reorganisation of rotung tree) as a result of RSSI and more reliable connections (which also means less congestion to a particular parent, giving distributed parents). This contributed towards increased PDR. But some packets (average 1.5 per node) were not able to send from nodes 4-10, because of the incorrect setting time (data-collection phase incorrect time sync., extended/incorrect blue – Refer image) : Solution: Increase routing tree period!

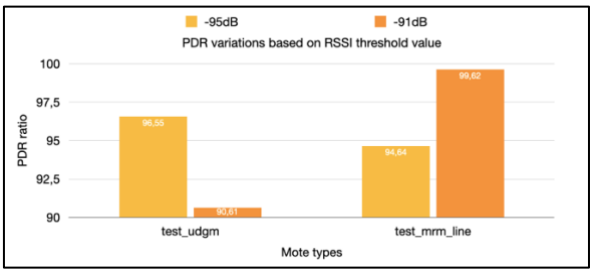


Figure 8: PDR for RSSI

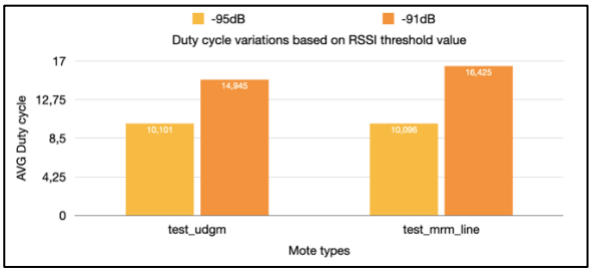


Figure 9: Duty cycle for RSSI

From the Figure 9 it can be observed that the duty cycle increased due to increase in RSSI, because of increased settling time for route creation and longer routes.

II. **MAX_UNICAST_PROCESSING_DELAY**

MAX_UNICAST_PROCESSING_DELAY: (MAX_HOP-1)*6
(RSSI_THRESHOLD: -91 dB and MAX_HOPES = 4)
test_udgm
PDR : 90.61%
AVG DC : 14.945%
test_mrm_line
PDR : 99.62%
AVG DC : 16.425%
The already obtained result from previous step.

MAX_UNICAST_PROCESSING_DELAY: (MAX_HOP -1)*7
(RSSI_THRESHOLD: -91 dB and MAX_HOPES = 4)
test_udgm
PDR : 100 %
AVG DC : 13.463 %
test_mrm_line
PDR : 99.81 %
AVG DC : 13.462 %

Increased settling time (caused by RSSI increase) causes final few beacon to be send even after the predicted/estimated data collection phase/callback (where we turn off LEDS_BLUE, sets callbacks for collection sequence and radio-off). Thus the estimated routing time is not enough! Solution: Increase routing tree period! (Refer: image).

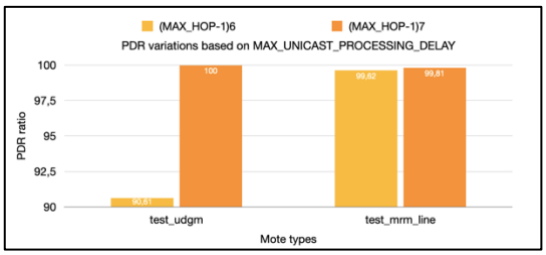


Figure 10: PDR for MAX_UNICAST_PROCESSING_DELAY

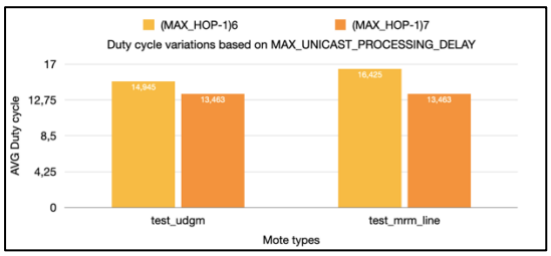


Figure 11: Duty cycle for MAX_UNICAST_PROCESSING_DELAY

From figure X and Y, we see an improved performance in both PDR and duty cycles.

III. DATACOLLECTION_COMMON_GREEN_START_DELAY

We increased the routing period by adding a guard time into the **DATACOLLECTION_COMMON_GREEN_START_DELAY** as shown below:

DATACOLLECTION_COMMON_GREEN_START_DELAY: (((MAX_HOPS)*CLOCK_SECOND + BLUE_LED_GUARD) - beacon.delay)

(BLUE_LED_GUARD = 200 for capturing the final few beacons lost due to increased settling time**)**

This caused increase in duty cycle, DC of mrm:14.5% and DC of udgm:17%. This can be seen in this image with lot of time left after time sync (long tail of blue LED).
Also, the effect of this change was that the PDR decreased by almost 7%, causing an overall 93% for both radio types. This was caused because of the shifting in **RADIO_TURN_ON_DELAY** which was beyond the first beacon from node 1 (hence some nodes missed some initial beacons). So we rewrote **RADIO_TURN_ON_DELAY as (EPOCH_DURATION - (RADIO_TURN_OFF_DELAY + DATACOLLECTION_COMMON_GREEN_START_DELAY + GUARD_TIME))** relative to **DATACOLLECTION_COMMON_GREEN_START_DELAY.** Previously this was a constant value which shifted every time we readjusted any delays coming before radio-on. These changes ensured a 100% PDR as shown below:

test_udgm
PDR : 100 %
AVG DC : 14.4 %
test_mrm_line
PDR : 100%
AVG DC : 13.69%
From image1, we can see that the data-collection starting phase of each node is not aligned. Also image2 shows a misaligned radio-on sequence for different nodes.

The observation that nodes with higher hop count (farthest from the sink), was more misaligned compared to the nearer nodes, helped in adding another parameter bc_recv_metric. This will be calculated as (hop_count of parent * 20). The resulting equation is:

DATACOLLECTION_COMMON_GREEN_START_DELAY: (((MAX_HOPS-1)*DELAY_CEIL + BLUE_LED_GUARD) - bc_recv_delay) - bc_recv_metric

The result of this change in DATACOLLECTION_COMMON_GREEN_START_DELAY is here**.** Also we made use of these value to obtain a uniform radio-on sequence for all the nodes as shown here, using the below equation.

RADIO_TURN_ON_DELAY (EPOCH_DURATION - (RADIO_TURN_OFF_DELAY + DATACOLLECTION_COMMON_GREEN_START_DELAY + (bc_recv_delay + bc_recv_metric))) + GUARD_TIME

(The optimized GUARD_TIME for Cooja is -50)

The resulting performance of these changes were:
test_udgm
PDR : 100%
AVG DC : 12.314%
test_mrm_line
PDR : 100%
AVG DC : 12.314%

These changes in alignment, based on hop-count decreased the duty cycle to 12%.

IV. DELAY_CEIL

The next aim was to decrease the Duty cycle. Adjusting the random delay (which we selected previously as CLOCK_SECOND) taken to avoid congestion before re-transmitting the beacon from a node proved to affect this.

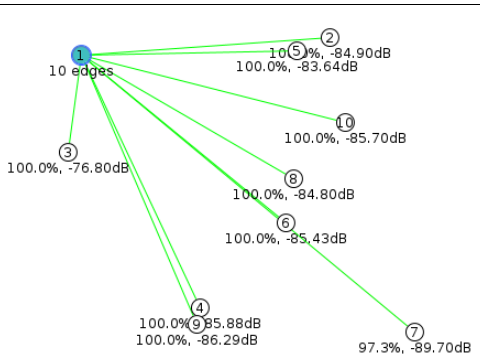
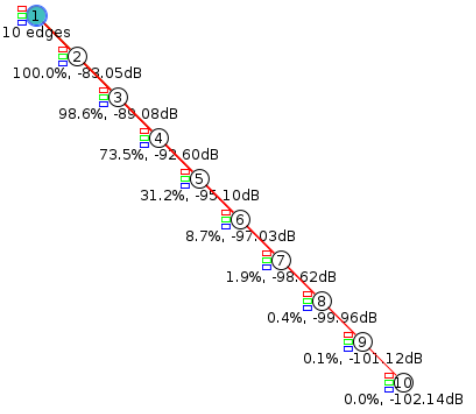
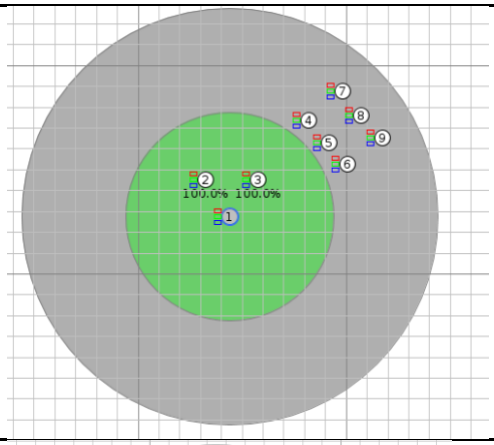
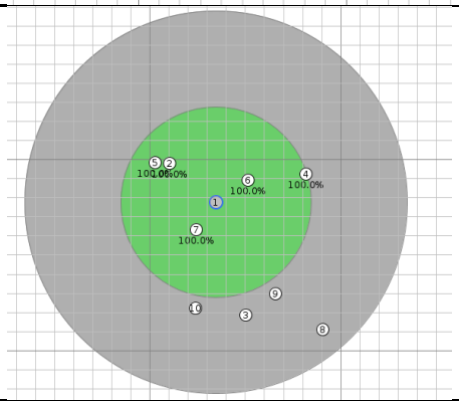
Experiments	a	b	c	d	e
DELAY_CEIL	**(CLOCK_SECOND/5) = 204 ticks**	**CLOCK_SECOND/4) = 256 ticks**	**(CLOCK_SECOND /3) = 341 ticks**	**(CLOCK_SECOND/2) = 512 ticks**	**350 ticks**
Results	**test_udgm** PDR: 100% AVG DC : 5.850% **test_mrm_line** PDR: 100% AVG DC : 5.850%	**test_udgm** PDR: 100% AVG DC : 6.353% **test_mrm_line** PDR: 100% AVG DC : 6.353%	**test_udgm** PDR: 100% AVG DC : 7.176% **test_mrm_line** PDR: 100% AVG DC : 7.176%	**test_udgm** PDR: 100% AVG DC : 7.245% **test_mrm_line** PDR: 100% AVG DC : 7.245%	**test_udgm** PDR: 100% AVG DC : 5.651% **test_mrm_line** PDR: 100% AVG DC : 5.649%
Remarks	Even though the PDR is 100%, one packet from lower nodes (>2) was not able to schedule due to congestion. Hence DELAY_CEIL needs to be increased.	Same as column a	Same as column a	Here we were able to send all the 58 packets scheduled by the App through unicast, without dropping any packets. Hence the optimal DELAY_CEIL will be below 512.	At DELAY_CEIL = 350, we were also able to obtain 100% PDR with all the packets scheduled from the App successfully send from the collection layer. Hence this value can be considered as the optimal value for the Cooja.

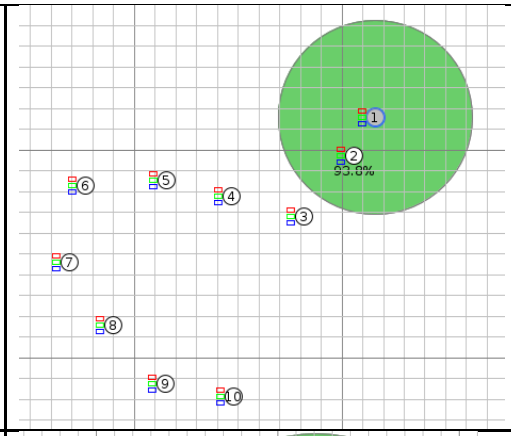
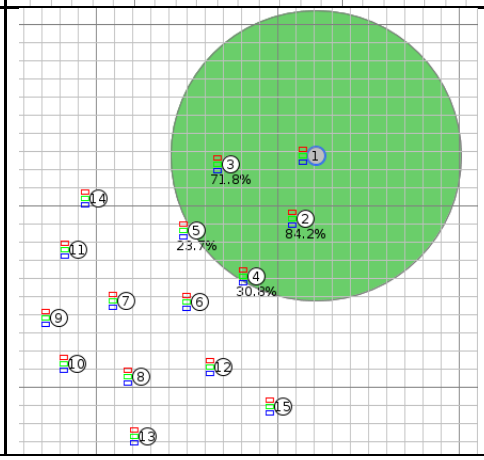
Testbed
PDR : 99.10 %
AVG DC : 7.630%

The duty cycle of testbed is slightly higher compared to the Cooja, also 6 packets are lost out of 665 send (hence 99.10%).

V. TOPOLOGY

Different radio models and topology are explored here for evaluating the performance of the time synchronized data collection protocol as show below.

test_mrm_line_10n PDR : 100% AVG DC : 7.245%	
test_mrm_line PDR : 100% AVG DC : 5.649%	
test_udgm (Lab9) PDR : 100% AVG DC : 5.651%	
test (udgm) PDR : 99.41% AVG DC : 7.245%	

test_udgm_line PDR : 97.51% AVG DC : 96.82%	
test_udgm (Lab6-7) PDR : 75.98% AVG DC : 17.365%	

Conclusion:

The final optimized results obtained for Cooja simulations (the topologies mainly considered for optimizing the implementation and parameters) are:

test_udgm
PDR : 100%
AVG DC : 5.651%

test_mrm_line
PDR : 100%
AVG DC : 5.649%

It can be concluded that adjusting the random delay with which we forward a broadcast beacon packet can greatly affect the overall duty cycle of the application. This is because the time synchronization phase takes twice as much time as actual data collection phase. Re-aligning the delay for each phase of the time-sync and data collection depending upon the application can be very useful in reducing duty cycle and increasing PDR. Making the radio-turn-on, radio-turn-off, end-of time sync phase etc. uniform by reducing the extra delays introduced (due to increased ho-count) can be very useful in modelling/predicting and to obtain a reliable result.

Congestion or collisions plays a critical role in the overall PDR of the network. In case of unicast transmissions, special care must be given to mitigate errors due to the collisions and congestions. This can be carefully selected to obtain a maximum PDR and thus decreased duty cycle (less retransmissions due to correct ACK).

So, if the nodes receive a packet with weak RSSI (near to -95dB) it will choose this instead of a stronger one, since the hop count of this would be better. But this doesn't

mean that the resulting route is better in-terms of reliability. Making use of good and multiple quality parameters like CORR (Link Quality indicator), ETX (Expected Transmission Count) etc. along with RSSI could be more useful in selecting reliable routing paths.

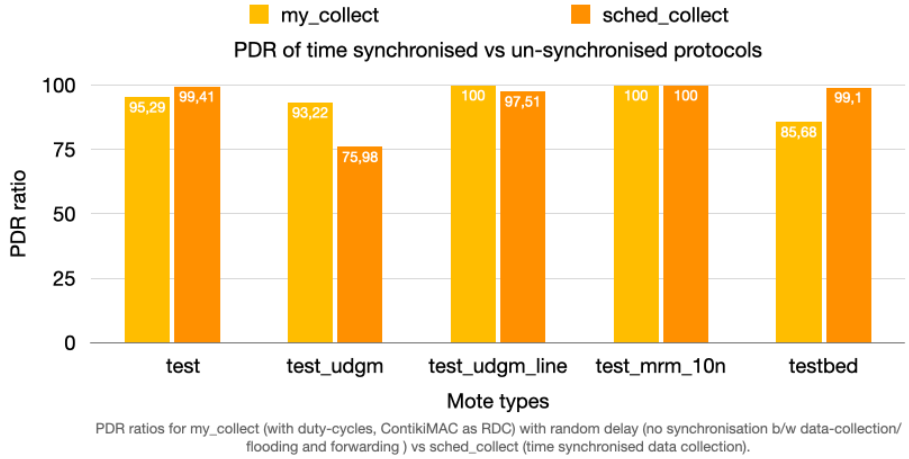


Figure 12:PDR comparisons for time-sync vs random delay data-collection

The time synchronized implementation preforms worse than the unsynchronized implementation with contikiMAC in 'test_udgm' and 'test_udgm_line'. In case of test_udgm lot of packets actually being send are lost during transmission either due to congestion of the complex network or not enough delays between each stage to accommodate the increased complex topology structure. A lot of packets are actually not able to be scheduled in case of test_udgm_line, even-though the reliability of the scheduled packets are good enough (97%). This is mainly because of the limited time-synchronization time (routing tree creation time) for the specific topology. If optimized for the network, we could perform better in this case with 100% PDR ratio.

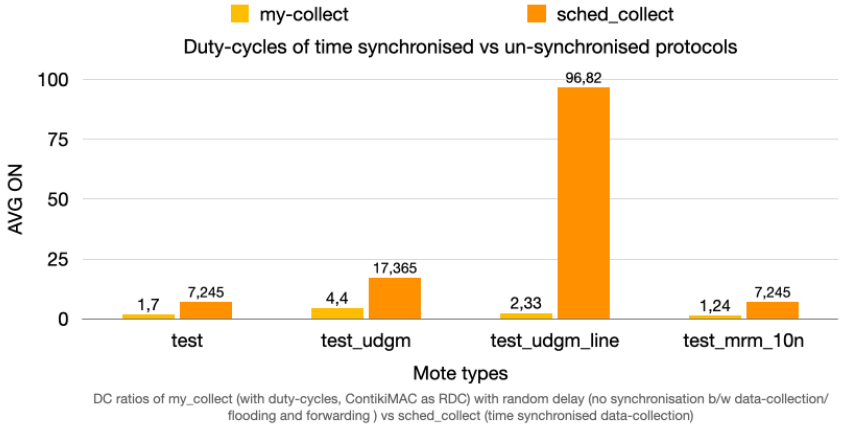


Figure 13: Duty cycle comparisons for time-sync vs random delay data-collection

Compared to the Contiki-MAC the sched_collect has more value for average duty cycle in almost all cases. In case of test_udgm_line, from node2 to node10, actual packets send drastically decreases from 53 to 2. The corresponding duty cycles from node2 to node10, increases from 15% to 96%. The outlier behavior of the test_udgm_line is directly related to this "un-schedulability" behavior which could be specific to that topology. Although this can be further tuned for the specific topology to optimize results better in future.