Implementation and Simulation of YSF in 6TiSCH

Siddharth Charan 190101085
Shrey Verma 190101083
Sanidhya Patel 190101059
Rishav Mondal 190101072

***Abstract*—We implemented YSF scheduling algorithm, which is an advancement of the MSF scheduling algorithm that builds the communication schedule in TSCH (Time Slotted Channel Hopping). YSF algorithm is aimed to minimize the end-to-end latency, specifically designed for data gathering applications in IIoT (Industrial Internet of Things). We also evaluate the performance of our algorithm using the 6TiSCH simulator. During simulation, we use the following simulation parameters :- We vary slotframe length to 29,47,101, and keep topology linear with traffic 2P/SF.**

I. Introduction

A 6TiSCH network is a multihop wireless IPv6 network which uses time-slotted channel hopping (TSCH). TSCH is a medium access mode of IEEE802.15.4 which provides deterministic properties, and increases robustness against external interference and multipath fading. A key component of TSCH is its scheduling function that builds the communication schedule, which greatly impacts network performance.
6TiSCH uses MSF(Minimal Scheduling Function) for the purpose of the scheduling.

II. About YSF

YSF is an advancement of MSF. YSF aims at minimizing latency and maximizing reliability for data gathering applications.
The following are aspects in which YSF has an improvement over MSF:

- Scheduling Process
- Permanent Cell Allocation Algorithms
- Handling Parent Switching
- Consideration of Joining Nodes

YSF avoids a schedule which is tightly coupled with the whole routing topology. Instead, it schedules cells on a per traffic flow basis. In YSF, a traffic flow is defined as a sequence of packets from a source node to the sink. YSF proceeds per traffic flow by assigning a set of cells to each node visited by the flow, in a top-down approach. The set of cells assigned on any node for a given traffic flow are scheduled one after the other. These cells are called permanent cells, which are optimized for application traffic to the sink.
YSF introduces two additional types of cells in addition to the types of cells in MSF:

1) the autonomous cell
2) the transient cell

Using the above mentioned optimizations, YSF achieves better reliability and efficiency as compared to MSF.

III. Changes in source code

We implemented the following changes in the code downloaded from the repository link given in the assignment.

A. Implementing YSF

The following files were changed:

- `sf.py`: The main source code for MSF was in this file. Instead of changing that we created a file `ysf.py`.
- `config.json`: We set YSF as scheduling function in place of MSF.
- `MoteDefines.py`: Some of the values of parameters were defined in this file already. We defined same of YSF.

```
YSF_MAX_NUMCELLS                    = 100
YSF_LIM_NUMCELLSUSED_HIGH           = 0.75 # in [0-1]
YSF_LIM_NUMCELLSUSED_LOW            = 0.25 # in [0-1]
```

Fig. 1. Definitions of some parameters

- `ysf.py`:
  (1) Scheduling Process: We implemented concept of hop_count and based on that while and while scheduling, we scheduled cells based on value of hop_count in top to down manner.

```
    special_list = [candidate_cells.hope_count, candidate_cells]
    special_list.sort()
    candidate_cells = special_list.second

    # prepare callback
```

Fig. 2. Sorting of cells joining based on `hop_count`

(2) Permanent Cell Allocation Algorithms: For channel offset selection and permanent cell allocation 2 algorithms were proposed, hence we created 2 new functions for them.

```
def _permanent_cell_allocation(n, hop_count, downward_neighbour):
    available_slots = self.SLOTFRAME_HANDLE_AUTONOMOUS_CELLS
    available_slots.remove(0)
    for cell in TX_CELL_OP:
        n:slots = get_slots(cell_allocation.permanent_cells)

        busy_slots = slots

        if cell_allocation.match(downward_neighbor) :
            if cell_allocation.hop_count == 2 :
                busy_slots += left_slots(slots, n+1 )
            elif cell_allocation >2 :
                busy_slots += left_slots(slots, n )
                busy_slots += right_slots(slots, n+1 )

        available_slots -= busy_slots

    if hop_count == 1 :
        num_slots = n+1

    else:
        num_slots = n

    return consecutive_slots(available_slots, num_slots)


def _channel_offset_allocation(n, hop_count, slots_for_permanent_cells):
    available_slots = self.SLOTFRAME_HANDLE_AUTONOMOUS_CELLS
    num_slots = n+ hop_count + 1
    right_edge = right_shifted(slots_for_permanent_cells, 1)

    all_slots_on_path = left_shifted(right_edge, num_slots)

    for cell in TX_CELL_OP:
        if TX_CELL_OP.has_common_slots(all_slots_on_path):
            permanent_cells = cell_allocation.permanent_cell
            busy_one = channel_offset(permanent_cell)
            available_channel_offsets.remove(busy_one)
    return pick_one(available_channel_offsets)
```

Fig. 3. Code for permanent cell allocation and channel offset allocation

(3) Handling Parent Switching: In parent switching like in paper we implemented concept of delayed removal of nodes related to old_parent, for that we changed _clear_cells function and added a paremeter ts, and used python function time.time() for delayed removing. Apart from that also maintained offset 15 for autonomous cell.

```
def _clear_cells(self, neighbor, ts):
    start = time.time()
    if(time.time() - start > ts )
        cells = self.mote.tsch.get_cells(
            neighbor,
            self.SLOTFRAME_HANDLE_NEGOTIATED_CELLS
        )
```

Fig. 4. Delayed removal of nodes

B. To vary parameters

The following files were changed:

- `config.json`: We changed following parameters:-
  1) `sf_class` : Set it's value to `YSF`.
  2) `exec_numMotes` : Varied it's value from 10 to 80 to get different data.
  3) `exec_numSlotframesPerRun` : Set it's value to 29, 47, 101 corresponding to 3 different slotframe lengths.

C. Generating graphs

We gathered raw data from file `exec_numMotes_4dat.kpl` and used MatPlotLib python library for generating graphs.

IV. Graphs generated by simulation

We compiled the results of our simulation in the form of 3 graph plots for each slotframe length, which we vary to 29, 47 and 101 (thus we obtain 9 graphs in total).
For each slotframe length, the 3 graphs are on PDR(Packet Delivery Ratio), end-to-end latency and energy consumption on the Y-axis, vs number of nodes on the X axis.

A. Slot frame length 29

Following graphs were generated:

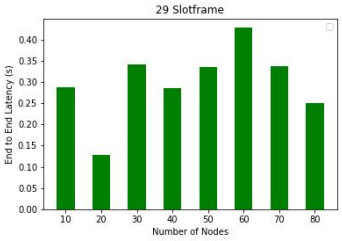

Fig. 5. End-to-end latency vs nodes

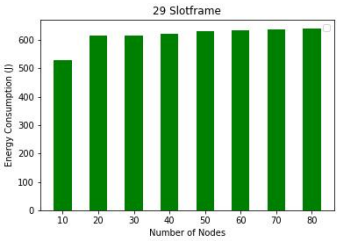

Fig. 6. Energy Consumption(J) vs nodes

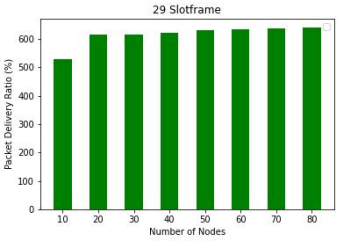

Fig. 7. PDR (%) vs nodes

B. Slot frame length 47

Following graphs were generated:

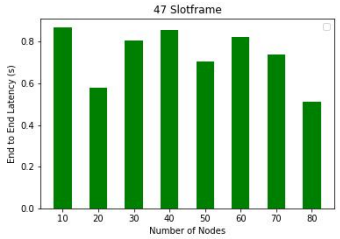

Fig. 8. End-to-end latency vs nodes

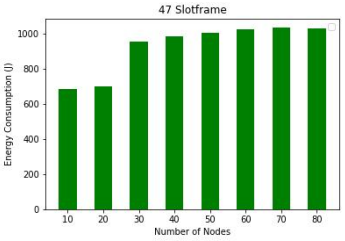

Fig. 9. Energy Consumption(J) vs nodes

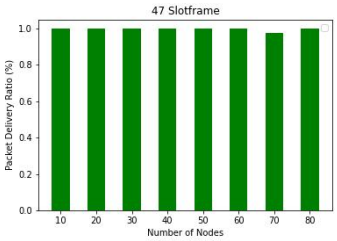

Fig. 10. PDR (%) vs nodes

C. Slot frame length 101

Following graphs were generated:

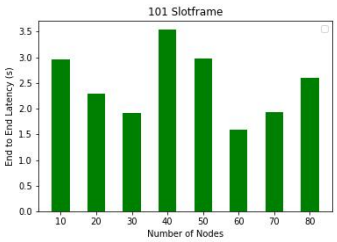

Fig. 11. End-to-end latency vs nodes

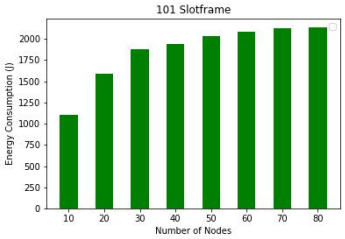

Fig. 12. Energy Consumption(J) vs nodes

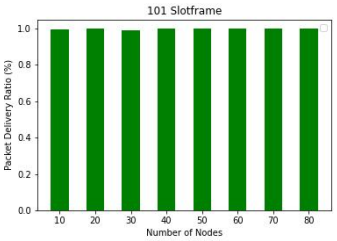

Fig. 13. PDR (%) vs nodes

V. Shortcomings of YSF

We are not very sure about this, but latency didn't show predictable relation with nodes, it might be an error in implementation too, but we think this is one of the shortcomings noticed.

Also, as mentioned in paper higher reliability and transmission rates can be achieved then current version of YSF.