6TiSCH Minimal Scheduling Function (MSF) draft-chang-6tisch-msf-00

Tengfei Chang

Malisa Vucinic

Xavi Vilajosana

Abstract

This specification defines the 6TiSCH Minimal Scheduling Function (MSF). This Scheduling Function describes both the behavior of a node **when joining the network**, and **how the communication schedule is managed in a distributed fashion.** MSF builds upon the 6top Protocol (**6P**) and the **Minimal Security Framework** for 6TiSCH.

In a nutshell

1. Start with a single cell
   - 6tisch-minimal
2. Perform secure join
   - 6tisch-minimal-security
3. Add/delete cells to parent
   - 6tisch-6top-protocol

→ Completely defined behavior, fully standardized story ☺

Interaction with 6TiSCH-minimal

- Frames exchanged over the minimal cell:
    1. EBs
    2. DIOs
    3. Join request/response messages between pledge and JP
    4. the first 6P Transaction a node initiates
- Access rules to the minimal cell: cut bandwidth in portions:
    - 1/(3(N+1)) for EBs (N= number of neighbors)
    - 1/(3(N+1)) for DIOs
    - Rest for join and 6P (see above)
- Slotframe organization:
    - Slotframe 0 for minimal cell
    - Slotframe 1 for cells added by MSF

Node Behavior at Boot (1/2)

- Start state
  - PSK
  - Any other configuration mentioned in minimal-security
- *[7-step join]*
- End state
  - node is **synchronized** to the network
  - node is using the **link-layer keying** material it learned through the secure joining process
  - node has identified its **preferred routing parent**
  - node has a **single dedicated cell** to its preferred routing parent
  - node is periodically sending **DIOs**, potentially serving as a router for other nodes' traffic
  - node is periodically sending **EBs**, potentially serving as a JP for new joining nodes

Node Behavior at Boot (2/2)

- Step 1 - Choosing Frequency
  - Listen on random frequency
- Step 2 – Receiving Ebs
  - Listen for multiple neighbors, shoes one as JP
- Step 3 - Join Request/Response
  - First hop over minimal cells, rest over dedicated (same for response)
- Step 4 - Acquiring a RPL rank
  - Select preferred parent
- Step 5 - 6P ADD to Preferred Parent
  - Single TX|RX|SHARED cell to parent
- Step 6 - Send EBs and DIOs
  - Accept children
- Step 7 - Neighbor Polling
  - Keep-alive to each neighbor you have cells to every 10s; remove if dead.

did you spot the typo?

Dynamic Scheduling (1/4)

- 3 reasons for adding/removing/relocating cells:
  - Adapting to Traffic
  - Switching Parent
  - Handling Schedule Collisions
- 6P carries out the work

Dynamic Scheduling (2/4)

- Reason 1/3: Adapting to Traffic
  - A node <u>always</u> has at least one cell to preferred parent
  - Keep counters to preferred parent:
    - NumCellsPassed
    - NumCellsUsed
  - When NumCellsPassed reaches 16:
    - If NumCellsUsed>12, add a cell
    - If NumCellsUsed<4, remove a cell

Dynamic Scheduling (3/4)

- Reason 2/3: Switching parents
  - Count number of cells to old parent
  - Schedule the same number to new parent
  - Remove cells from old parent

Dynamic Scheduling (4/4)

- Reason 3/3: Handling schedule collisions
  - Counter for each cell to preferred parent:
    - NumTx
    - NumTxAck
  - When NumTx==256:
    - NumTx>>1
    - NumTxAck>>1
  - Periodically, compare numbers for all cells to parent
    - If no roll over yet, abort
    - If PDR of one cell <50% of cell with max PDR, relocate

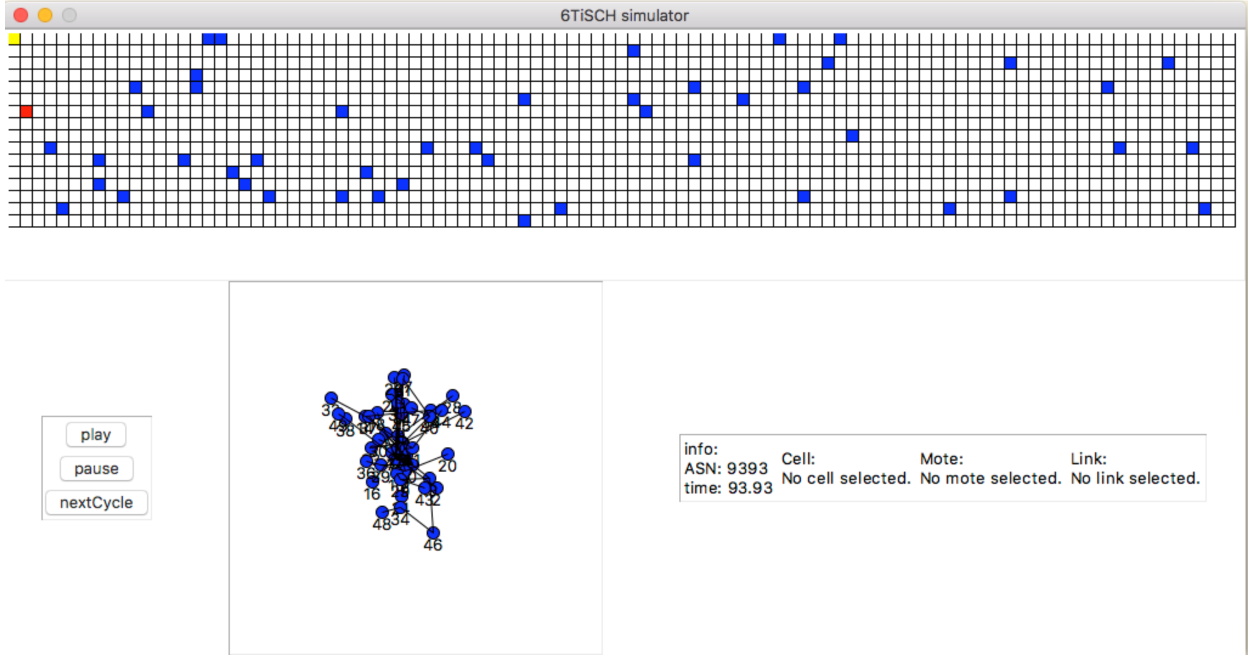
Other “details”

- 6P SIGNAL command
- Rules for CellList
- 6P Timeout Value
- Rule for Ordering Cells
- Meaning of the Metadata Field
- 6P Error Handling
- Schedule Inconsistency Handling

6TiSCH Simulator Implementation

https://bitbucket.org/6tisch/simulator

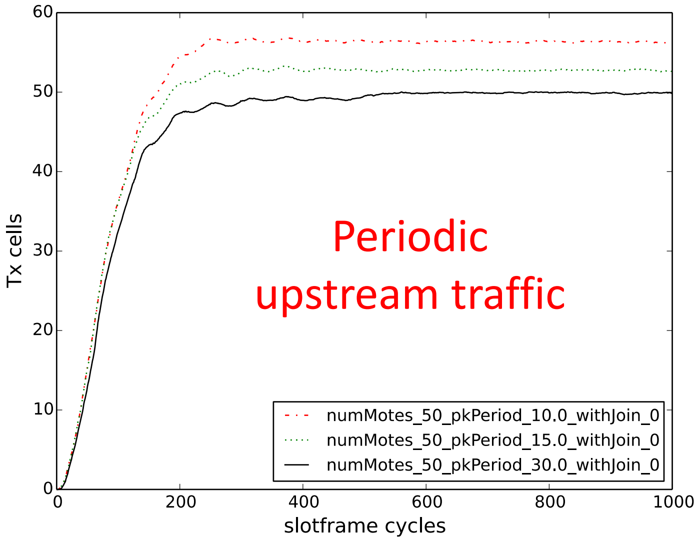
Over 100 000 slotframe cycles simulated!

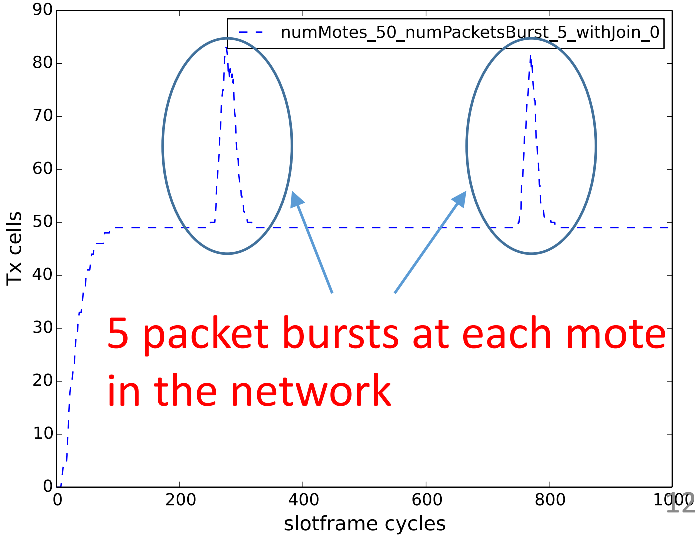


Code at: https://bitbucket.org/6tisch/simulator/pull-requests/7/implementation-of-msf-according-to-draft

50 motes, randomly deployed on 2x2 km area
Each mote has at least 3 neighbors
Simulated 6P signaling

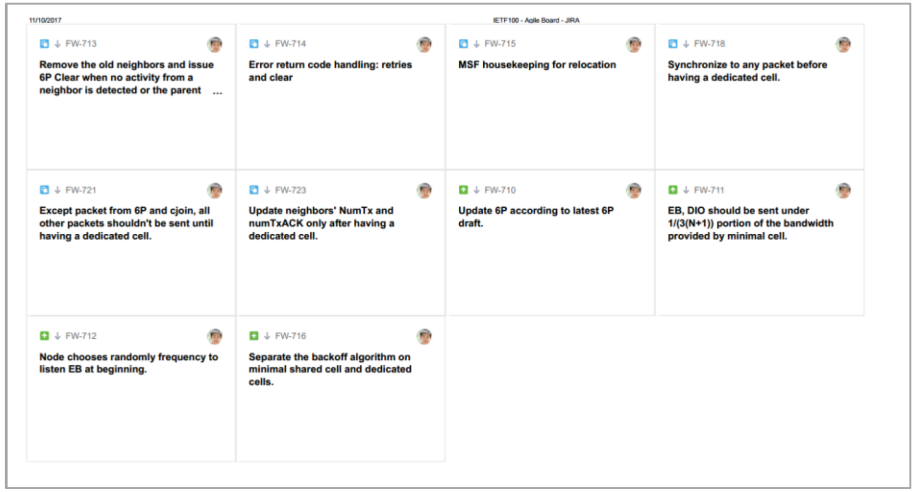
Study MSF convergence
(join phase not implemented yet)





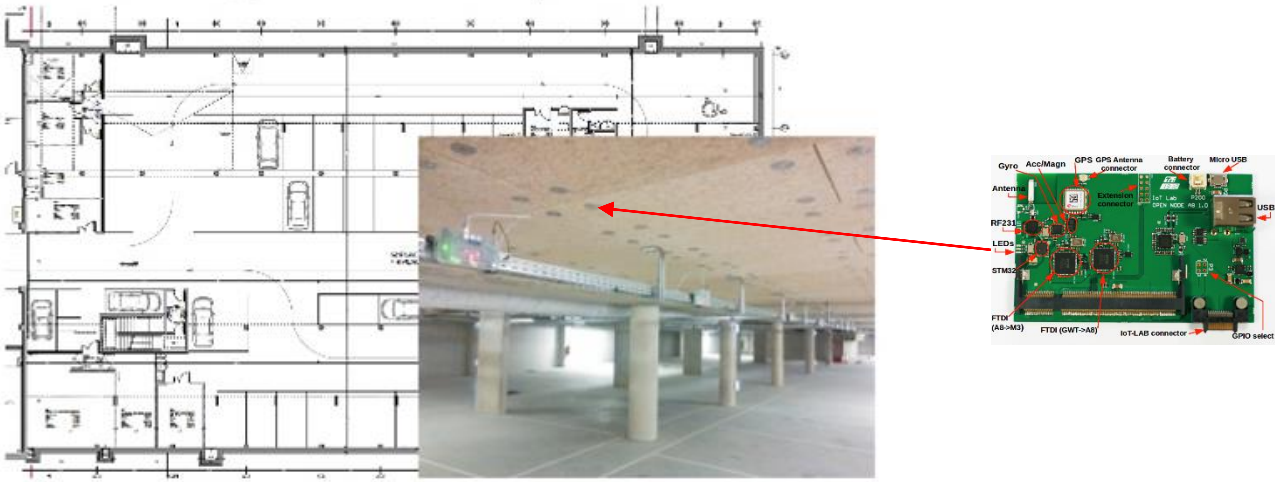
MSF OpenWSN Implementation

www.openwsn.org, https://github.com/openwsn-berkeley

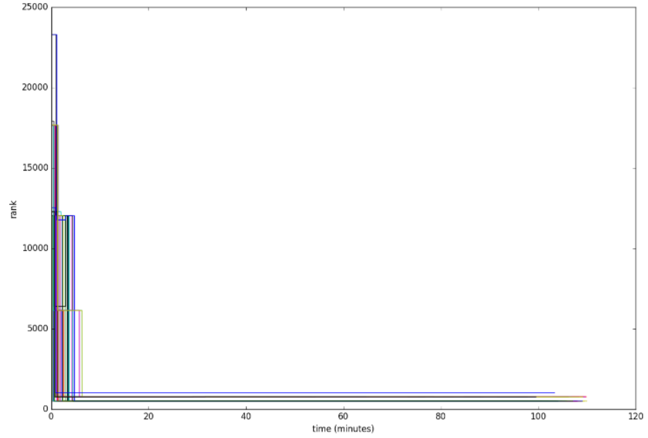


after 2-week MSF code sprint

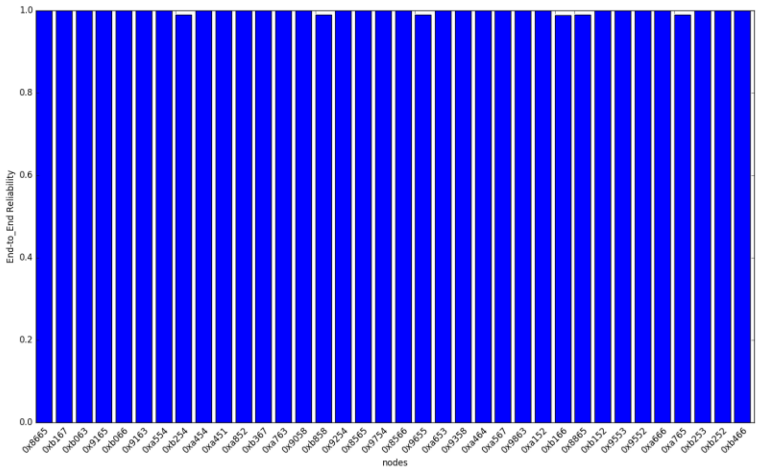
IoT-LAB @Inria-Saclay



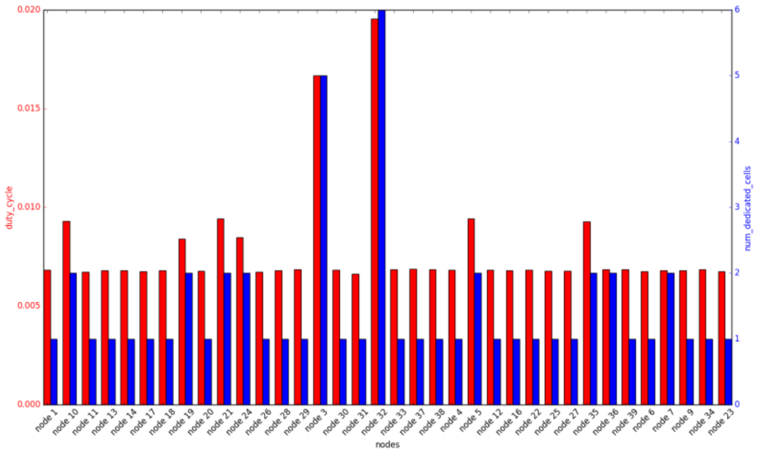
35-node deployment (Cortex-M3+AT86RF231)



node rank stable after join



100% end-to-end reliable on most nodes



~1% radio duty cycle

Conclusion and Future Steps

- Running code: It works!
- Simple to implement
- Broadcast strategy on the minimal cell critical for join phase
- How to set MSF parameters as a function of e.g. latency requirements, duty cycle?
- Lessons learnt from implementation
  - When a schedule inconsistency is detected, the 6P CLEAR Request and Reponse SHOULD be exchanged on the minimal cell.
  - Limit backoff exponent on dedicated cells as only 2 nodes discussing.
- Further experimental benchmarking based on application scenarios