Course Project:
Time Synchronization and Periodic Data Collection

Low-power Wireless Networking for the Internet of Things
2020–2021

In this project, you will implement a protocol for periodic data collection. In contrast with the collection protocol seen during the labs, in which nodes transmit with random time delays, you are asked to schedule transmitters to reduce the interference between them. You are provided with an application template that contains the basic structure of the requested implementation. You will run both simulations and testbed experiments to assess the performance of your implementation, then quantitatively compare the scheduling approach to the random-delay one. You will follow a research methodology similar to the one commonly adopted by the low-power wireless research community.

Periodic data collection. The goal is to forward packets reliably and efficiently across the network towards a collection point, the sink node. Data collection in this project is done periodically, with interval T. First, nodes must be all synchronized to the sink. Then, data collection is performed. During data collection, all nodes must keep their radio on (i.e., without duty cycling) to send their data packets and to relay packets from neighboring nodes to the sink. Finally, after a fixed duration, all nodes must turn off their radio to save energy. Exploiting time synchronization, the radio of the node is activated again shortly before the next synchronization, which is followed by data collection, and so on. Each repetition of this cycle is called an *epoch*, depicted in Figure 1.

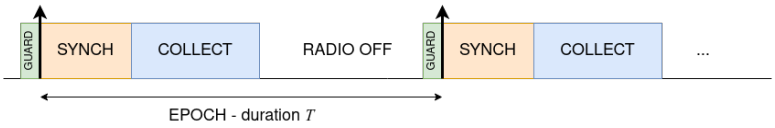

Figure 1: Epoch structure of the collection protocol.

Time synchronization. To synchronize nodes to the sink, you will reuse the network flood used in Lab 6 and Lab 7 to build the routing topology. At the beginning of each epoch, the sink must send a broadcast packet to build the topology and also to achieve time synchronization. All nodes that receive this packet can directly synchronize to the sink. Then, they should re-broadcast the packet with a random delay as in the labs, allowing nodes farther away to also synchronize and join the network. Of course, the synchronization error due to random delays would accumulate as the packets propagate hop after hop (Figure 2). To account for this, packets must embed the random delay used in their transmission. Receivers should timestamp the reception of packets (e.g., using `clock_time()`) and correct this timestamp by subtracting the delay found in the payload. With this simple mechanism, they can align to the sink with sufficient accuracy even if they are multiple hops away. Processing delays can also contribute to de-synchronization. You can estimate them to improve the synchronization accuracy achieved.

Routing topology. To build the routing topology you can reuse the logic implemented in Lab 6 and 7. The topology should be rebuilt at the beginning of each epoch to refresh the routes (from the sources to the sink) used in data collection, to adapt the topology to failures and potential changes. A minimal-cost spanning tree must be constructed as a result of the flood, using hop count as the routing metric as in the labs. During collection, nodes should forward packets to their parent in the tree, eventually reaching the sink.

Scheduling transmissions and wake-ups. After synchronization, if performed correctly, all nodes will enter the data collection phase roughly at the same time. Within the data collection phase, nodes schedule the transmission of their packet based on *i)* their node ID and *ii)* the expected time for a packet from the farthest hop to reach the sink. Assume that the data collection phase begins at time t, and that it takes at most a time interval p for a packet to reach the sink. With node 1 being the sink, a node with a given ID will transmit its packet at $t + (ID - 2) \times p$. For example, consider node IDs that go from 1 to 20. Node 2 will schedule transmission at time $t + (2 - 2) \times p$, or simply t. It will be the first to transmit in the epoch. Node 20 will

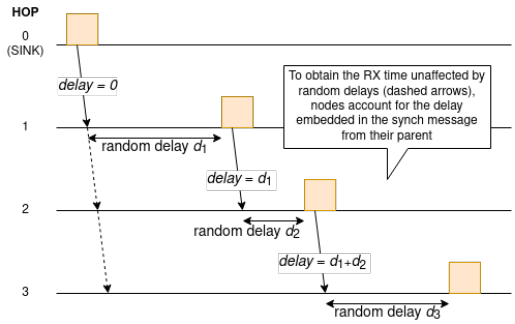

Figure 2: Synchronization phase. Random delays accumulate across hops. The total is embedded in the re-broadcasted packet, allowing the next node to apply a correction for the random delays.

instead transmit at $t + 18 \times p$. Note that, based on the previous assumptions, the collection phase has a fixed duration. At $t + 19 \times p$, nodes turn off their radio and schedule their wake up for re-synchronization in the next epoch. A guard time should be present so that nodes wake up slightly earlier than the expected start time of the epoch, accounting for synchronization misalignments. By improving synchronization accuracy, you can shorten the guard time and radio duty cycle, thereby reducing energy consumption.

Protocol evaluation. You should analyze the performance of your protocol in terms of packet delivery ratio (PDR) at the sink and average duty cycle (DC) for all nodes in the network. In all experiments, assume an epoch duration $T = 30$ s. On the other hand, the value of p is not fixed, as it depends on the number of hops in the network and the time it takes for a node to receive and re-transmit a packet. You are asked to explore different values of p to evaluate the trade-off between reliability and energy consumption, showing the results from COOJA.

Comparison with the duty-cycled approach. Once you identify the best p value for a given network topology, run the original data collection protocol developed during the labs using the same value of T. In this case, enable ContikiMAC and compare the results to those of your project implementation, using COOJA.

Testbed experiments. Simulations are a very useful tool for protocol design. Nonetheless, testbed experiments are key to understand the effect of the environment on our solutions. Once your implementation achieves satisfactory results in COOJA, you can run the same tests in the testbed, using Zolertia Firefly nodes. You will need to adapt the value of p. If you use testbed node 1 as the sink (DISI floor), the network should span no more than 4 hops. It is sufficient to show the results for one experiment using your implementation, and one experiment using the duty-cycled approach. In the presentation for this project you can find additional details about how to estimate duty cycle of Firefly nodes.

Testbed experiments are optional. However, the maximum mark for a project submitted with no testbed experiments is 27/30.

Report. After the performance evaluation, you should write a report with a brief description of the design decisions behind your protocol logic, your implementation details, the results of your performance evaluation, and a concise summary of the findings. Your report should clearly describe your experimental methodology (i.e., how you run your experiments) and how the metrics are computed. To show your results, we encourage you to add tables or make meaningful plots, e.g., showing the node duty cycle and PDR. If you use Python for your analysis, we recommend Matplotlib to plot your results.

Implementation Notes.

- **Turning on and off the radio.** Your protocol must switch on and off the radio appropriately to reduce radio duty cycle and therefore energy consumption. You can use `NETSTACK_MAC.on()` and `NETSTACK_MAC.off(false)` to turn on and off the radio, respectively.
- **Clock Timer.** In this project, you need to carefully timestamp received radio packets and schedule data transmissions. To this end, you shall use Contiki's clock module, whose API you can find in contiki/core/sys/clock.h. To improve the clock resolution w.r.t. the default value in Contiki, the code template provided sets the clock to tick 1024 times per second. Therefore, each tick corresponds to ≈ 1 ms.
- **Implementation Style.** To implement the project we recommend a mixed programming style that includes both processes or protothreads and function callbacks from Rime primitives or ctimers. You can check the process.h header file as well Contiki's Wiki page about processes.

- **Node IDs.** To schedule data transmissions you should use the `node_id` variable directly set by COOJA in the simulations or by the code template provided for testbed experiments. To use the `node_id` variable, you simply need to include the node_id.h header file.
- **Application Interface.** Your protocol implementation must provide a *i)* `sched_collect_open()` function to open the connection and start the data collection protocol and *ii)* a `sched_collect_send()` function for the application to set the data packets to be sent to the sink. Besides this, your protocol should inform the application of packets received at the sink via a callback function. You can find the definition of these functions in the file `sched-collect.h` provided.
- **Zolertia Firefly Clock Configuration.** To make your protocol work in the Zolertia Firefly platform available on the testbed you should change the `CLOCK_CONF_SECOND` value in contiki/platform/zoul/contiki-conf.h to `1024UL`. This value is set in line 62. Due to the way the Contiki configuration is set for this platform, we cannot directly redefine this value properly in the `project-conf.h` file as done for the Tmote Sky.

Code Template. To implement the project, we provide several files to aid your development.

- **app.c** a simple application that starts the data collection protocol and that implements the callback functions. We will use these functions to automatically test your implementation. The application uses the API your protocol should provide. This application SHOULD NOT be changed.
- **sched-collect.h** the header file of your periodic data collection protocol with the minimum API you should provide. This API is used by the top-level application (`app.c`). The function prototypes to start, send packets, and the callbacks SHOULD NOT be changed. You may, however, extend the header file with other functions or data structures as required.
- **sched-collect.c** the source file of your data collection protocol with stub functions. You should build your logic in this file, although you may use additional source files as you may need.
- **tools** we provide two tools to set the node IDs in the testbed and to enable duty cycle estimation in both simulated Tmote Sky nodes in COOJA and Zolertia Firefly nodes in the testbed. The files within the `tools` folder MUST NOT be changed.
- **parse-stats.py** a Python script that parses your Cooja or testbed `.log` files and computes the PDR and duty cycle of each node. You may use this script to analyze the performance of your protocol implementation and compare it with the one achieved with the data collection protocol implemented in Lab 6 and 7. For the script to work, you will need Python's NumPy and Pandas modules. You should be able to install them in the terminal as follows: `$ pip install pandas numpy`
- **testbed** we provide some bash scripts and a `experiment.json` file to facilitate the evaluation of your protocol in the Firefly testbed.
- **Makefile** to compile your code. You may add new source code files to the Makefile.
- **project-conf.h** to configure your application and protocol.
- **Cooja simulation files** to test and evaluate your protocol. You are encouraged to define your own simulation files to assess your protocol performance under other conditions, topologies, or radio models.

Rules of the game: How to (and how not to) work on the project

- The project is individual. Due to the structure of the course, the student is strongly encouraged to deliver it before the beginning of the second semester. However, students that are unable to meet this deadline are encouraged to contact the instructor well in advance, especially if they have already taken the written exam successfully.
- You should submit *(i)* the Contiki source code and *(ii)* a brief report with the description of your solution, your evaluation results, and your conclusions. The report must be in English.
- You should demonstrate that the project works as expected using the COOJA simulator and/or the testbed in front of the teaching assistants and/or the instructor.
- In the view of the current pandemic, you may be required to present your project via Zoom instead of discussing it in presence with the instructors. These logistics aspects will be clarified when setting up the date and time of your presentation.
- The code MUST be properly formatted. Follow style guidelines (e.g., Contiki code style).

- You MUST contact through e-mail the instructor (gianpietro.picco@unitn.it) AND the teaching assistants (p.corbalanpelegrin@unitn.it, davide.vecchia@unitn.it) well in advance, i.e., at least a couple of weeks before the presentation. If you have time constraints, it is up to you to make them known to the instructors in due time. Remember that the instructors have their own availability constraints and may not be able to accommodate exactly the date/time you need.
- Both the code and the documentation must be submitted in electronic format via email at least three days before the meeting. The documentation must be a single self-contained PDF. All code must be sent in a single tarball consisting of a single folder (named after your surname) containing all your source files and analysis scripts.
- The code SHOULD NOT be published in GitHub or any other online service/tool. While we encourage students to become part of the open source community, we believe sharing your project code can help other students develop their implementation, which can be unfair to fellow students and result in plagiarism (see below).
- The project will be evaluated based on the technical implementation (correctness and efficiency) and the report quality, which should demonstrate the student understanding of the problem at hand.

Plagiarism is not tolerated. Students whose project is partially copied/adapted from other students' projects will undergo disciplinary measures. Depending on the gravity of the plagiarism, these may involve reporting the student to the highest disciplinary bodies of the university, therefore possibly jeopardizing the offending student's academic career. If you are afraid you may not complete your project, get in touch with the instructors and remember: an incomplete project will be considered more positively than one with parts of the code adapted from someone else.