Course Project:
Time Synchronization and Periodic Data Collection

Low-power Wireless Networking for the Internet of Things

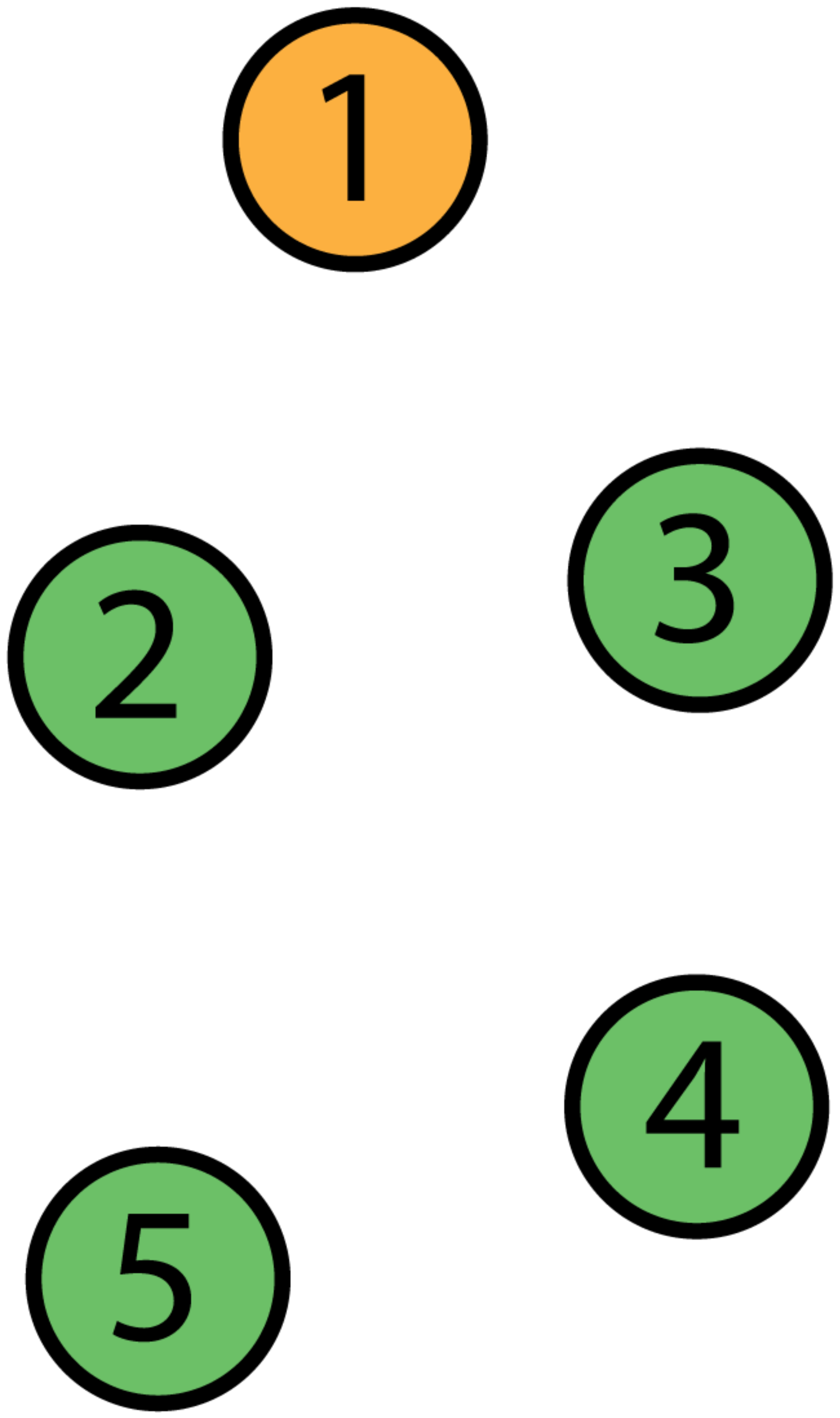
University of Trento, Italy

2020-2021

Lab 6 - 7: Data Collection Recap

Step 0:

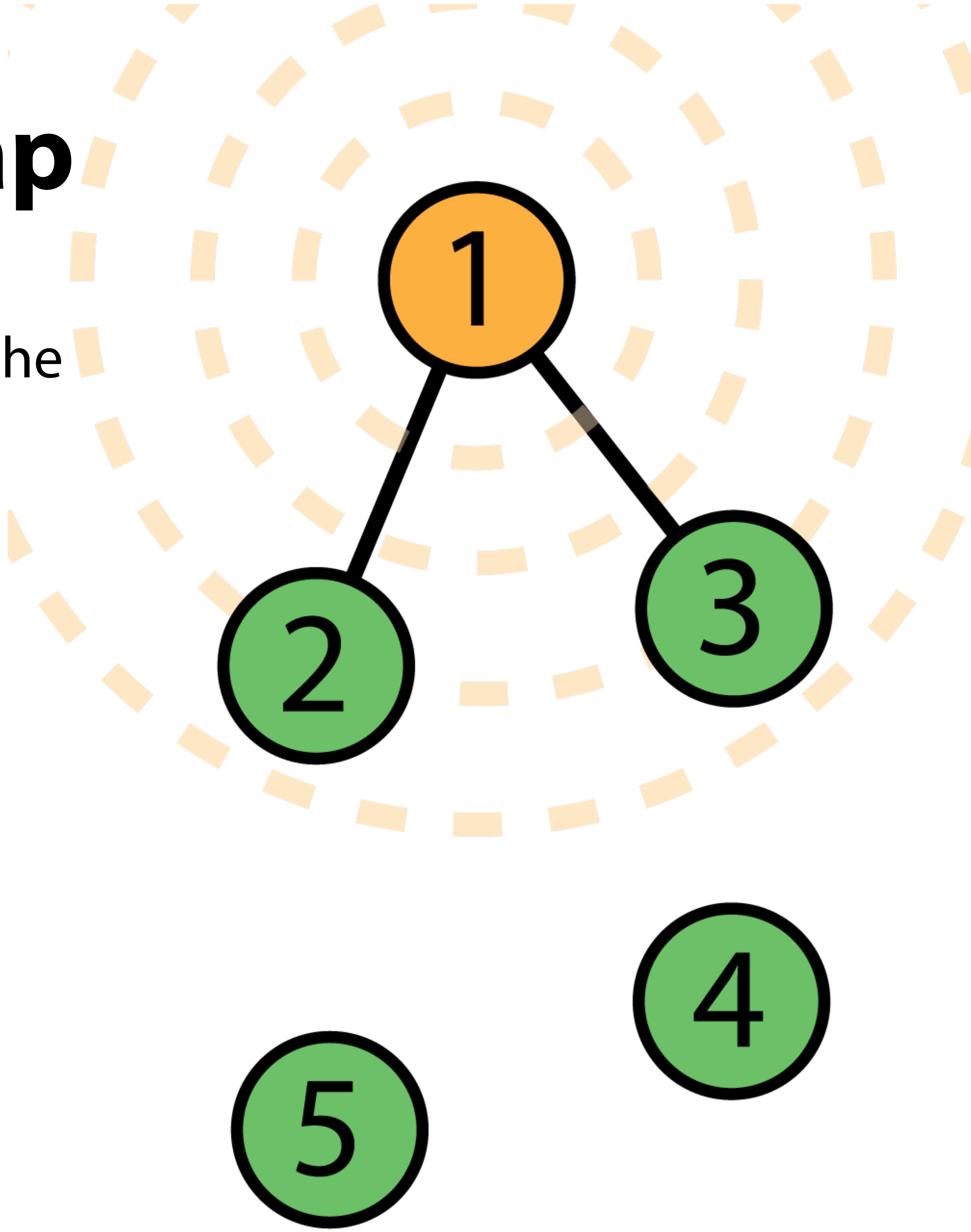
Initially we have a disconnected nework. We need to build the tree with node 1 as the root (data collection sink).



Lab 6 - 7: Data Collection Recap

Step 0:
Initially we have a disconnected nework. We need to build the tree with node 1 as the root (data collection sink).

Step 1:
The root sends a broadcast packet with **seqn = 0** and metric **h = 0**. Nodes 2 and 3 join the network with **h = 1**, selecting node 1 (the root) as parent.

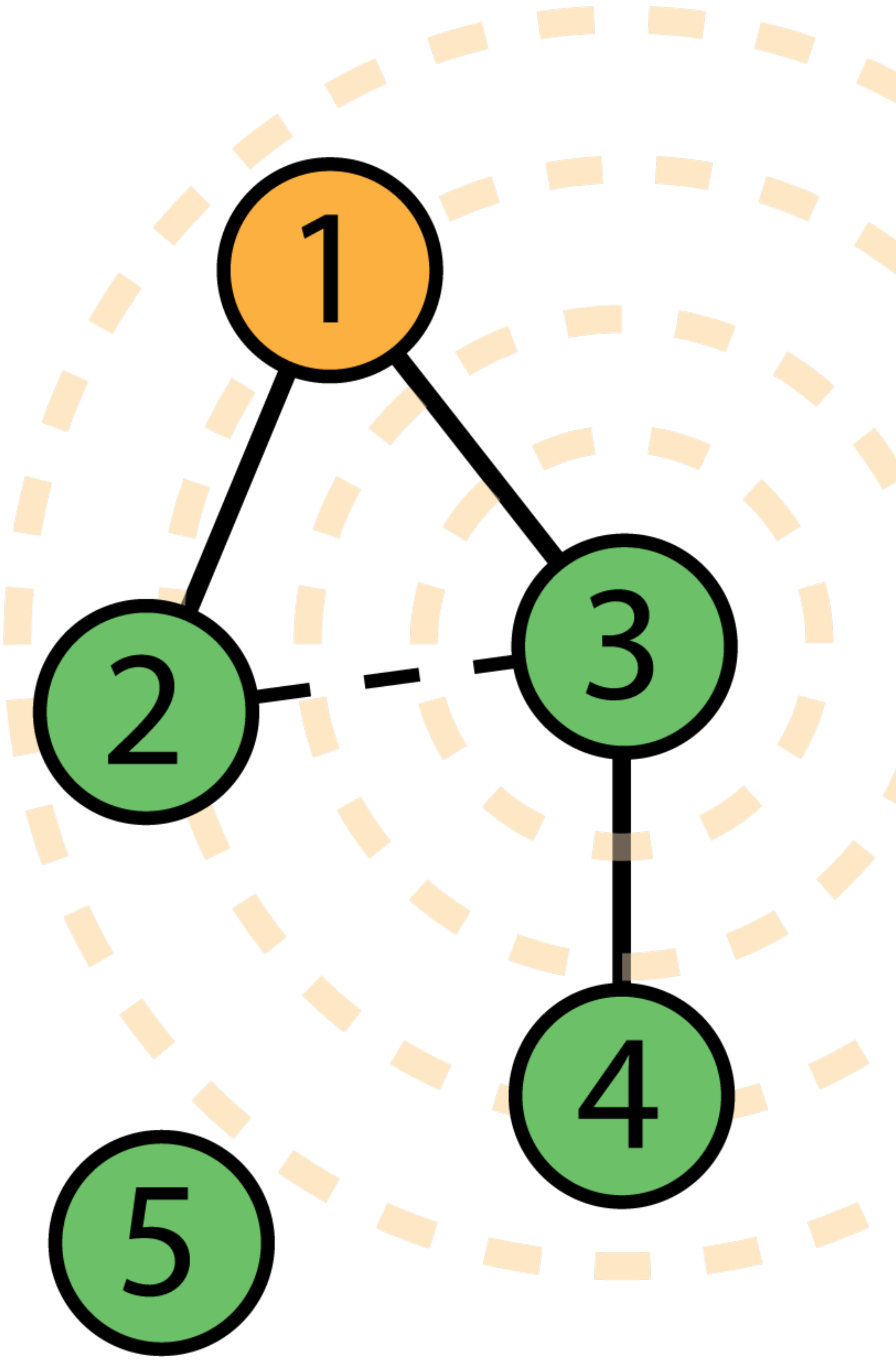


Lab 6 - 7: Data Collection Recap

Step 0:
Initially we have a disconnected nework. We need to build the tree with node 1 as the root (data collection sink).

Step 1:
The root sends a broadcast packet with **seqn = 0** and metric **h = 0**. Nodes 2 and 3 join the network with **h = 1**, selecting node 1 (the root) as parent.

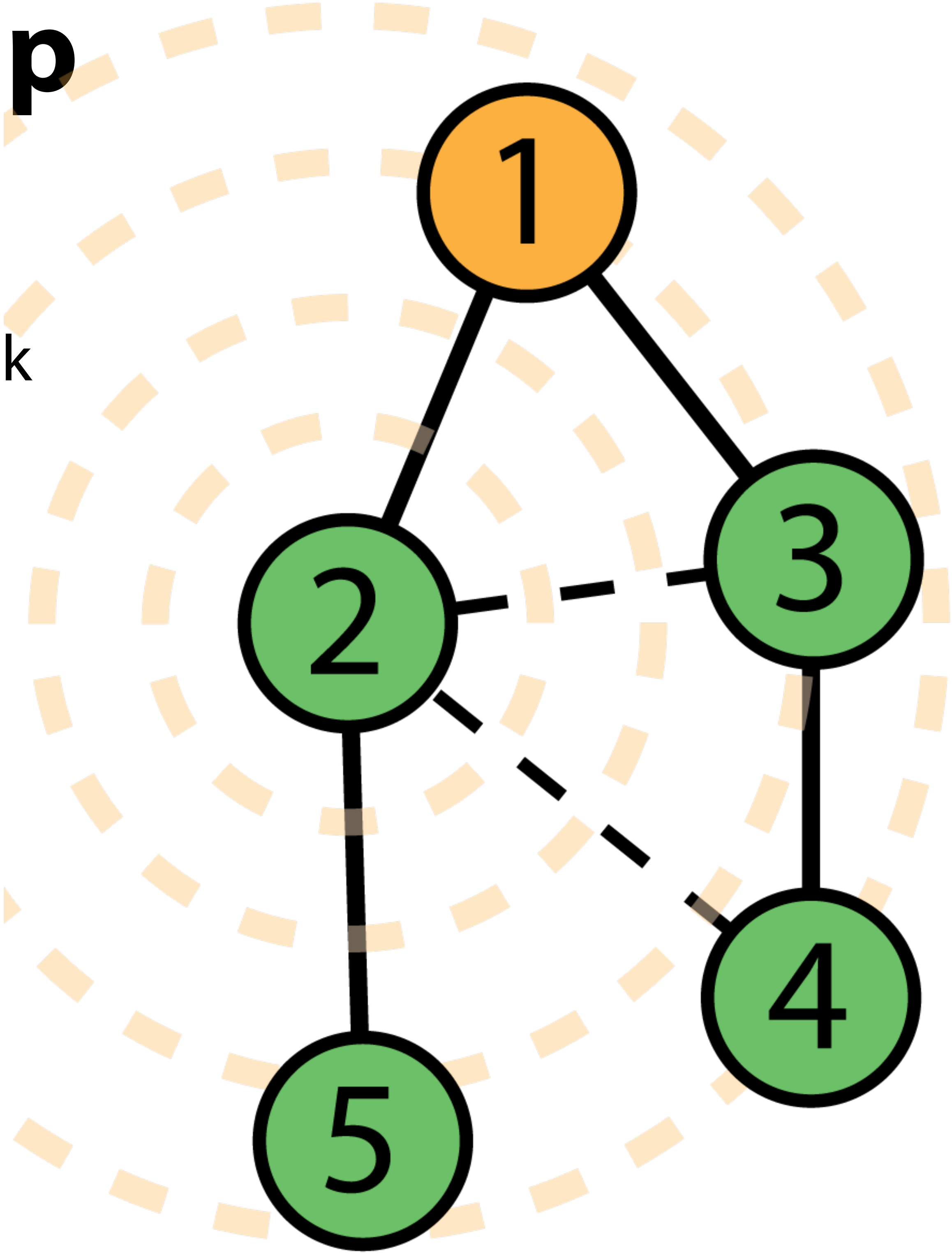
Step 2:
Node 3 sends a broadcast packet with **seqn = 0** and metric **h = 1** after **a random delay** from the reception of the packet from 1. Node 4 joins the network with metric **h = 2** and selects node 3 as parent.



Lab 6 - 7: Data Collection Recap

Step 3:

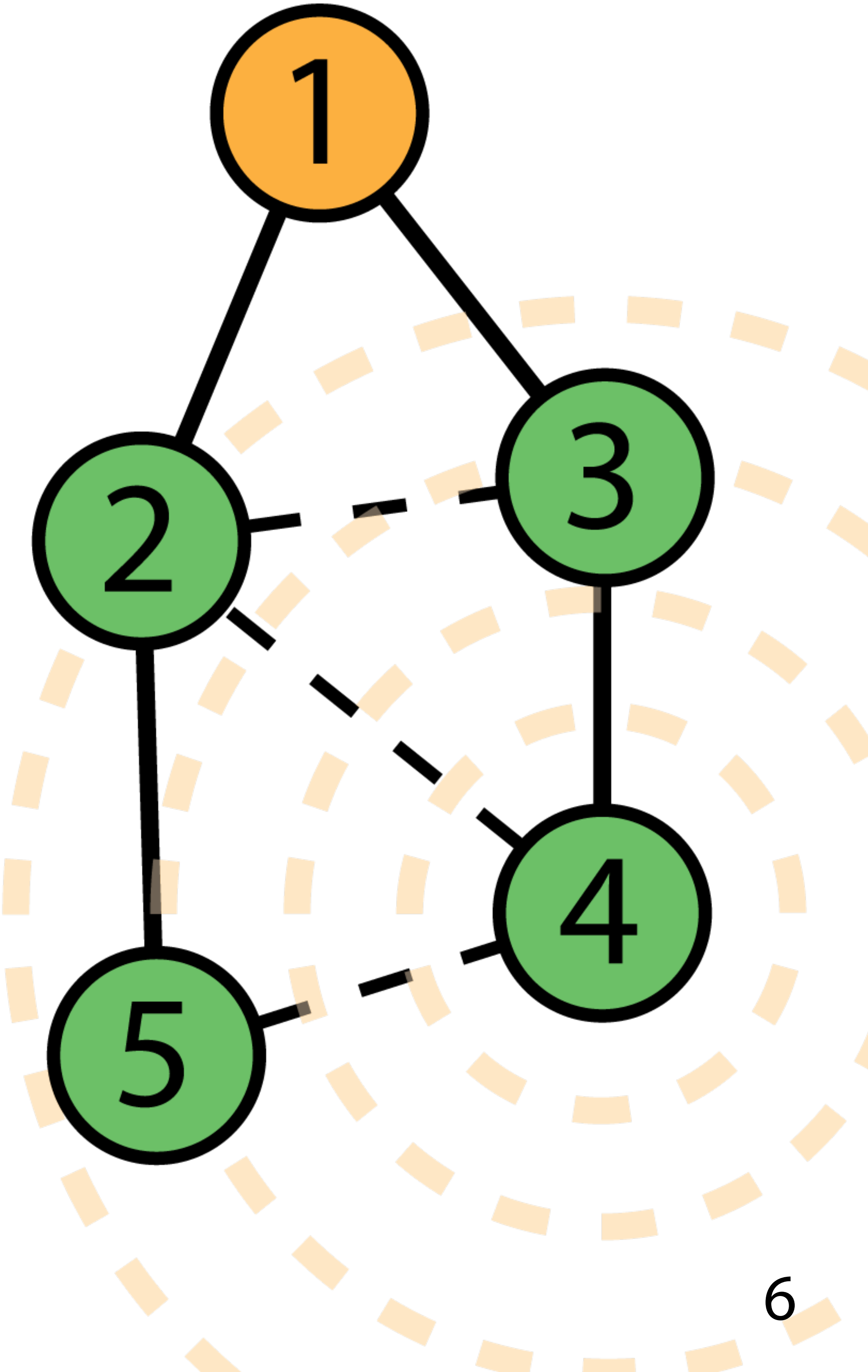
Node 2 sends a broadcast packet with **seqn = 0** and metric **h = 1** after **a random delay**. Node 5 joins the network with metric **h = 2** and selects node 2 as parent.



Lab 6 - 7: Data Collection Recap

Step 3:
Node 2 sends a broadcast packet with **seqn = 0** and metric **h = 1** after **a random delay**. Node 5 joins the network with metric **h = 2** and selects node 2 as parent.

Step 4:
Node 4 sends a broadcast packet with **seqn = 0** and metric **h = 2** after **a random delay**.

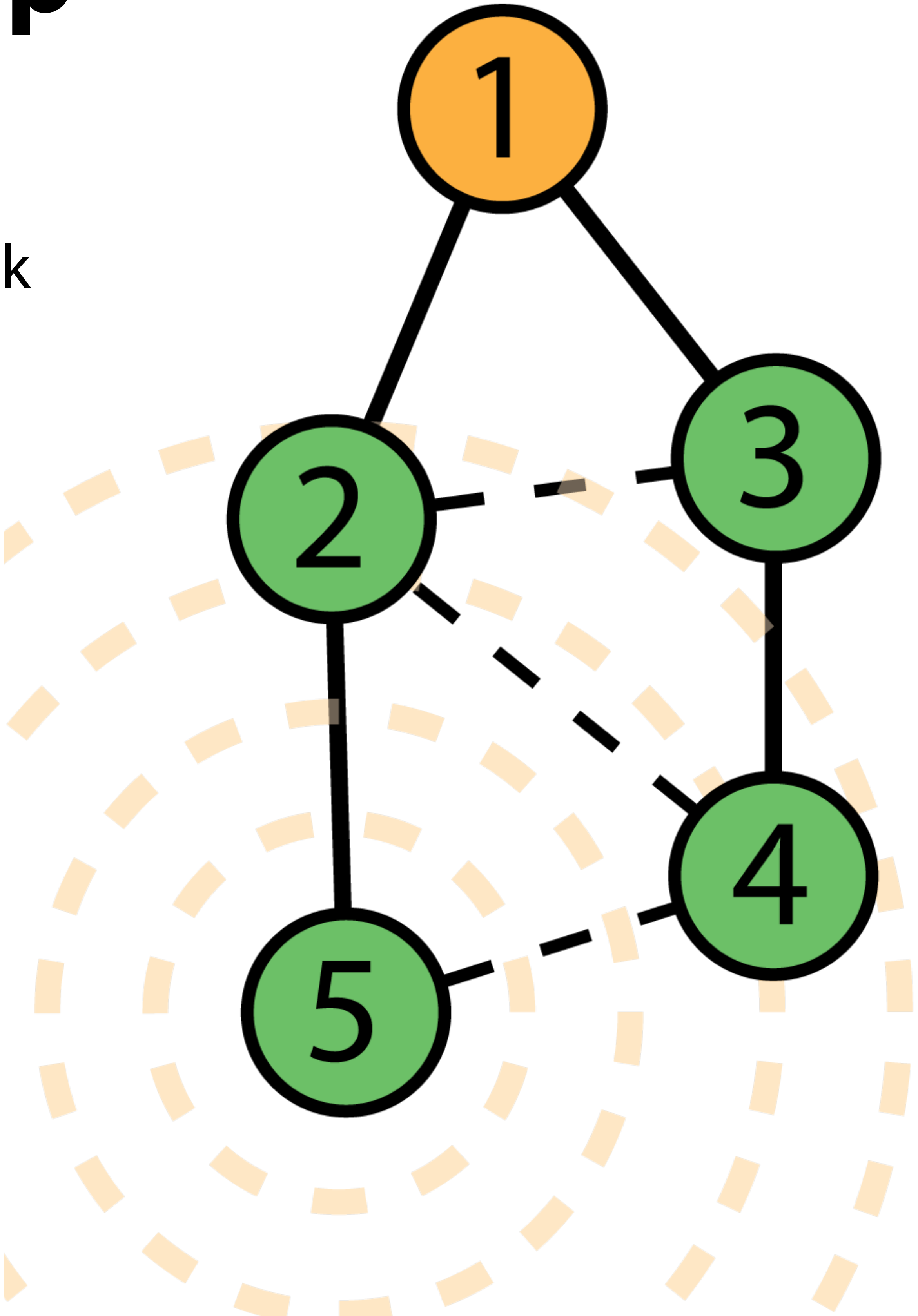


Lab 6 - 7: Data Collection Recap

Step 3:
Node 2 sends a broadcast packet with **seqn = 0** and metric **h = 1** after **a random delay**. Node 5 joins the network with metric **h = 2** and selects node 2 as parent.

Step 4:
Node 4 sends a broadcast packet with **seqn = 0** and metric **h = 2** after **a random delay**.

Step 5:
Node 5 sends a broadcast packet with **seqn = 0** and metric **h = 2** after **a random delay**.



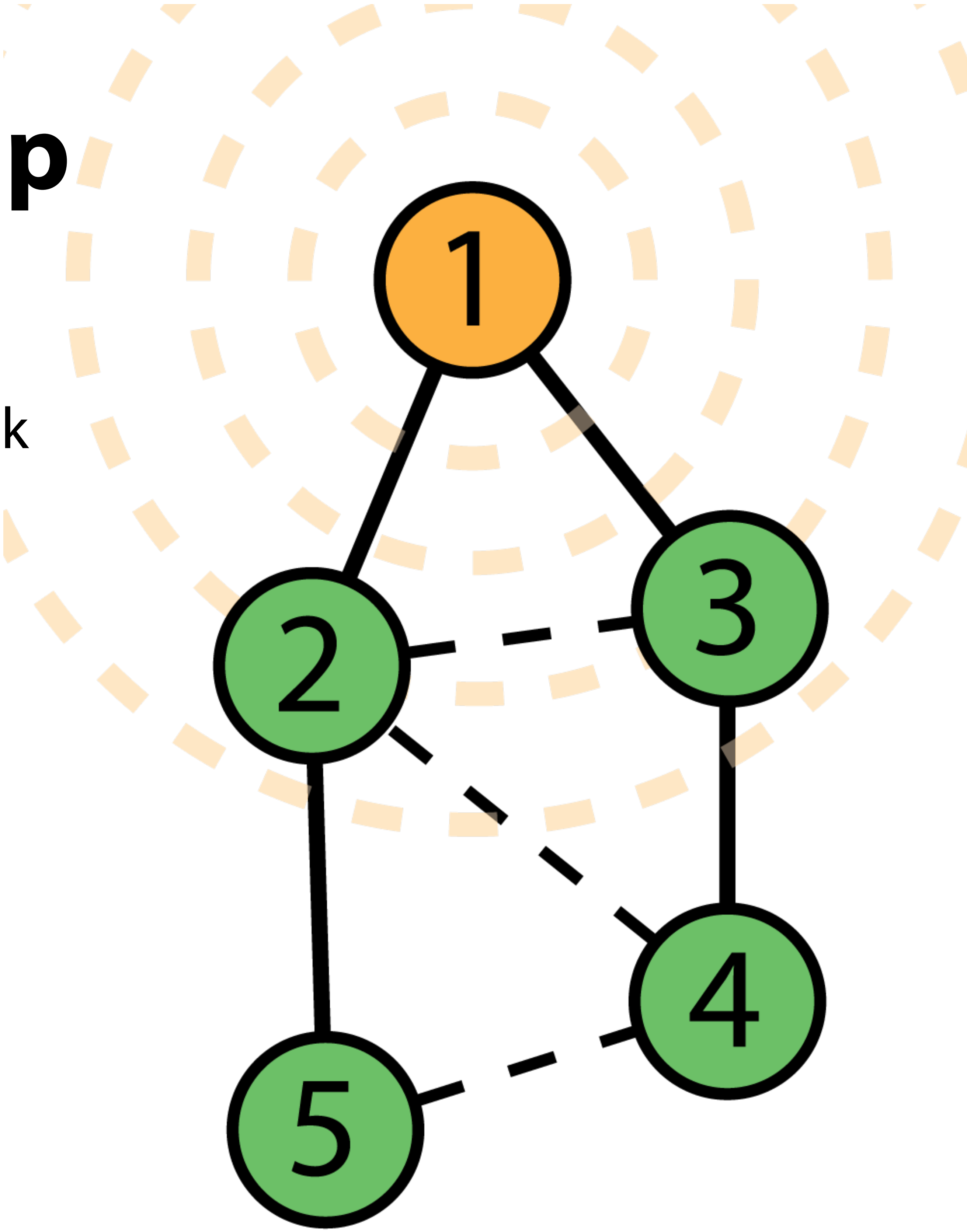
Lab 6 - 7: Data Collection Recap

Step 3:
Node 2 sends a broadcast packet with **seqn = 0** and metric **h = 1** after **a random delay**. Node 5 joins the network with metric **h = 2** and selects node 2 as parent.

Step 4:
Node 4 sends a broadcast packet with **seqn = 0** and metric **h = 2** after **a random delay**.

Step 5:
Node 5 sends a broadcast packet with **seqn = 0** and metric **h = 2** after **a random delay**.

Afterwards
The sink increases the sequence number and a new flood is started to rebuild the routing topology and cope with network changes.



Time Synch + Periodic Data Collection

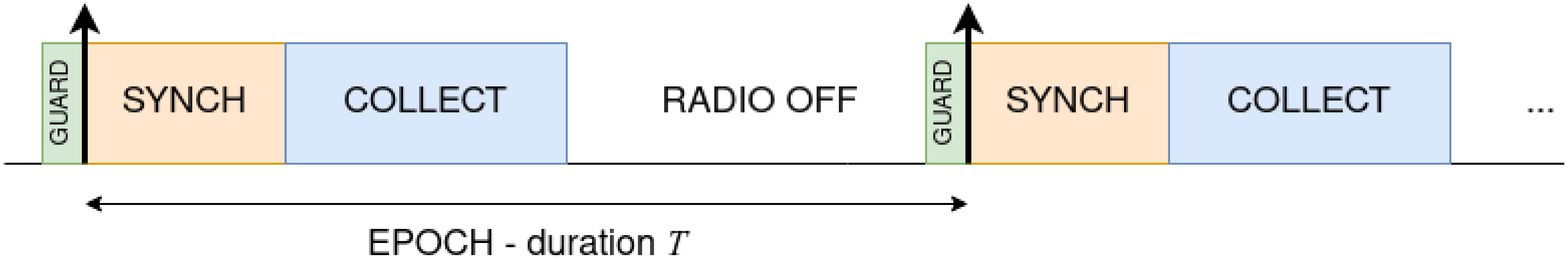
Time Synch + Periodic Data Collection

Epoch Structure

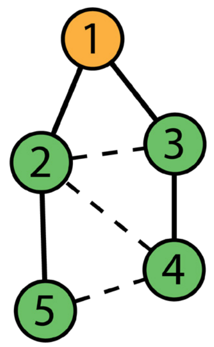
Periodic sequence of three phases:

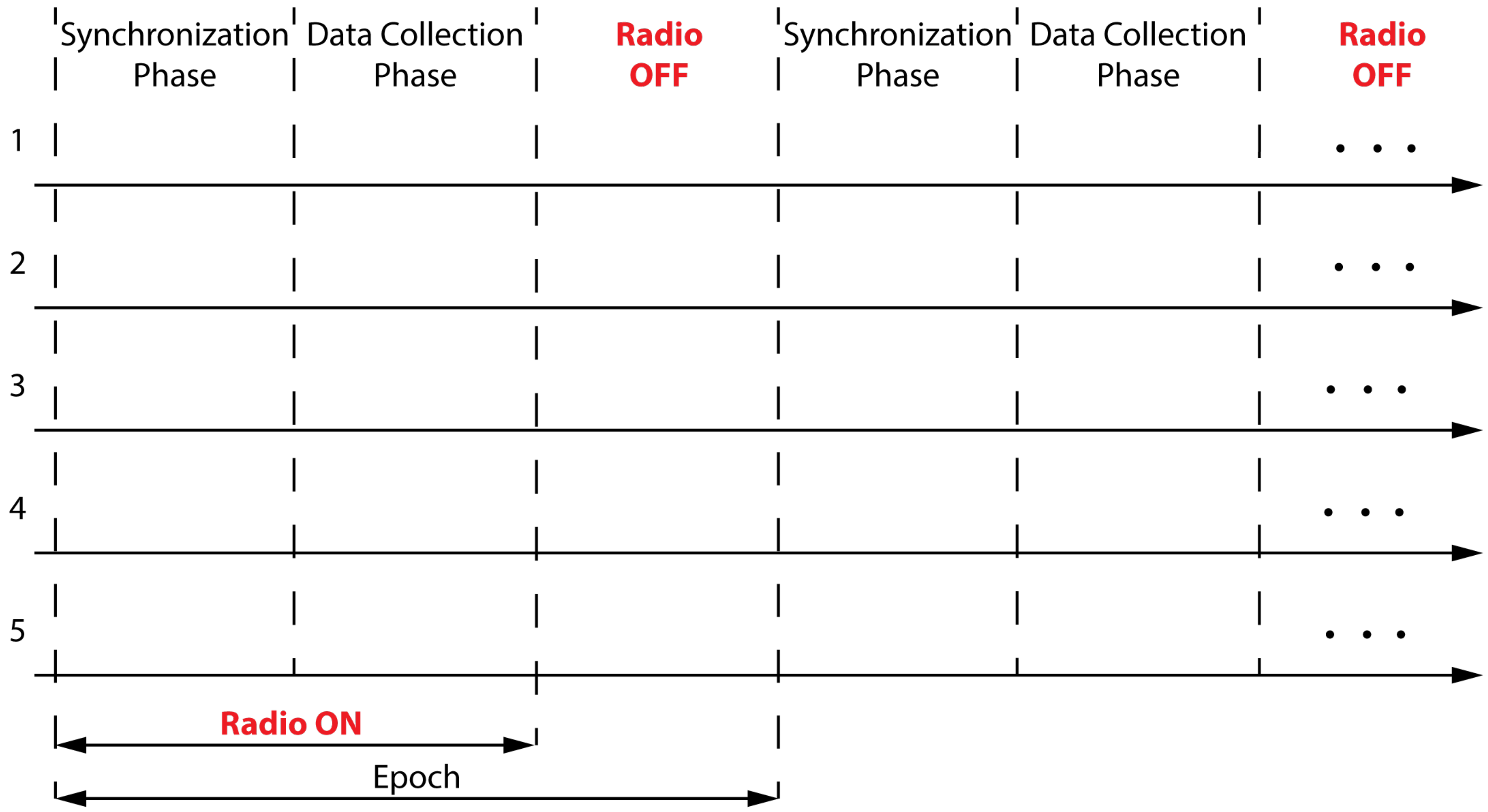
- Synchronization phase
- Scheduled data collection
- Radio OFF

You need to use a guard time to account for time synchronization misalignments.

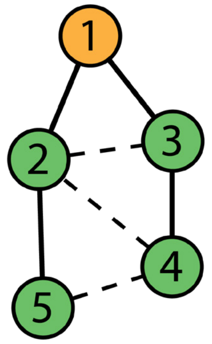


Time Synch + Periodic Data Collection



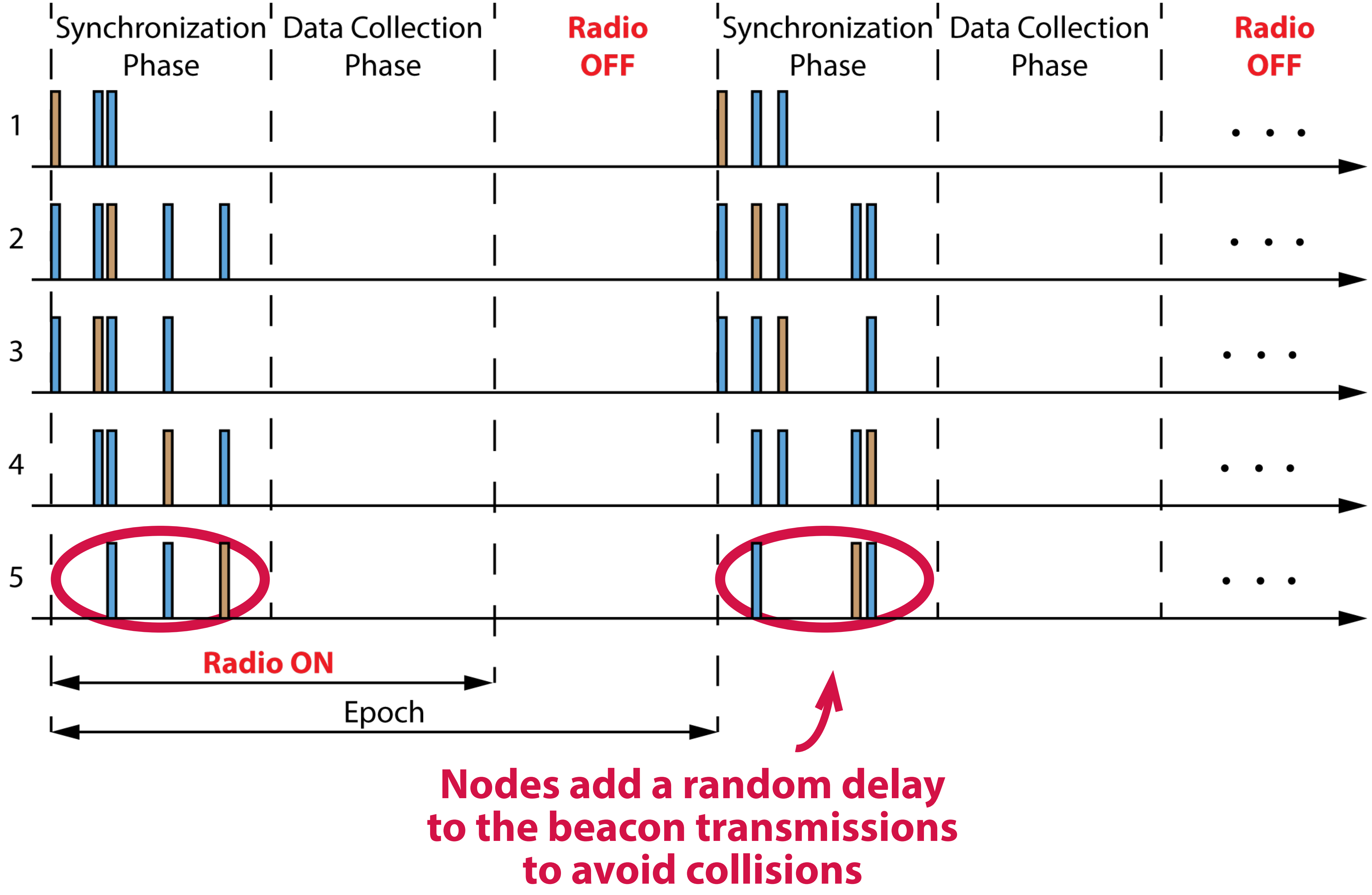


Time Synch + Periodic Data Collection

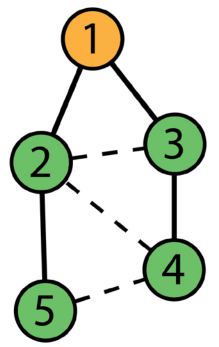


Synchronization Phase

- Same flood as in the data collection protocol of Lab 6-7 to build the routing topology.



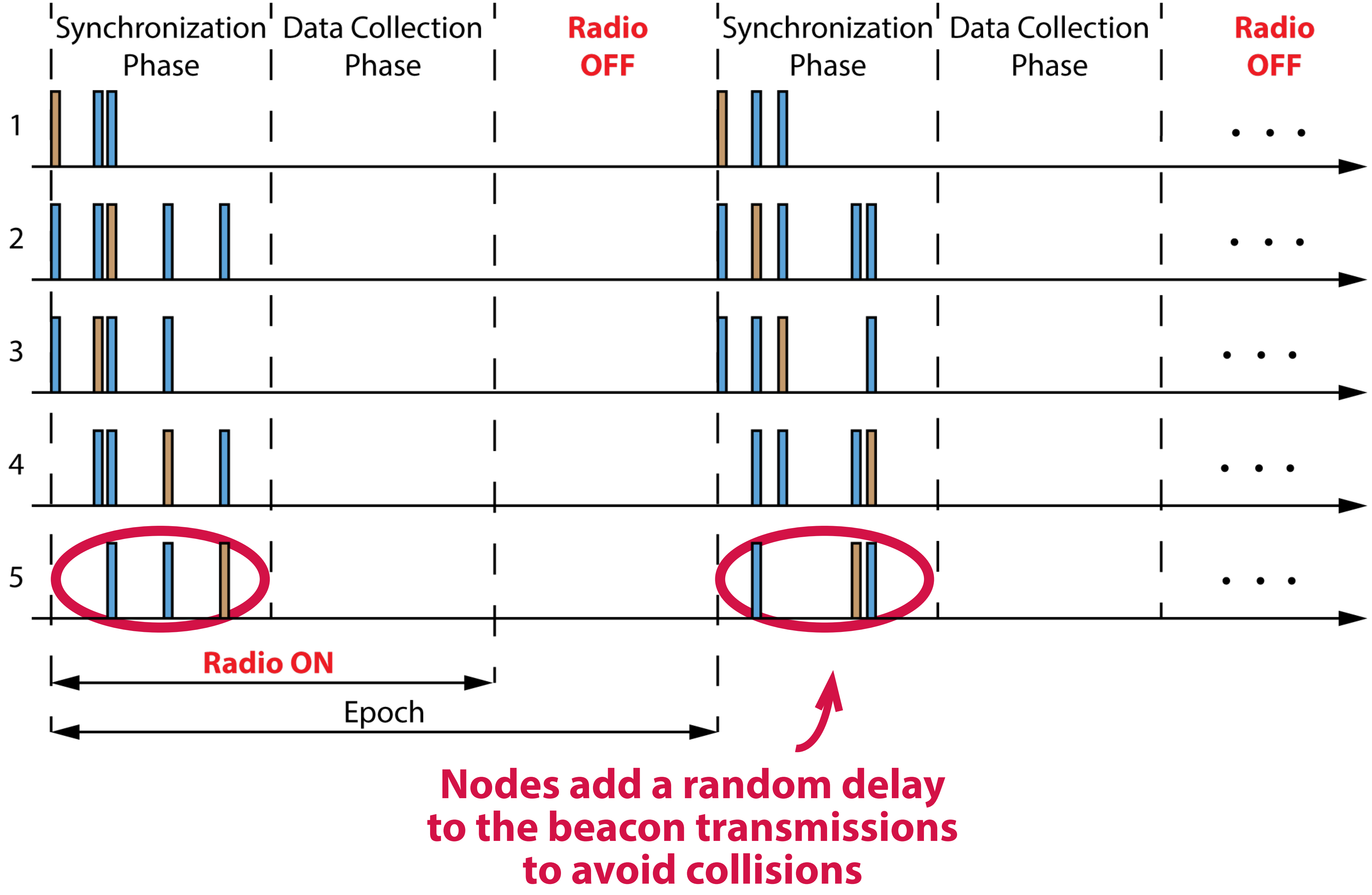
Time Synch + Periodic Data Collection



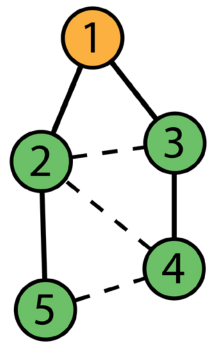
Synchronization Phase

- Same flood as in the data collection protocol of Lab 6-7 to build the routing topology.

- Nodes embed in their packet payload the **accummulated delay** from the transmission of the first packet from the sink.



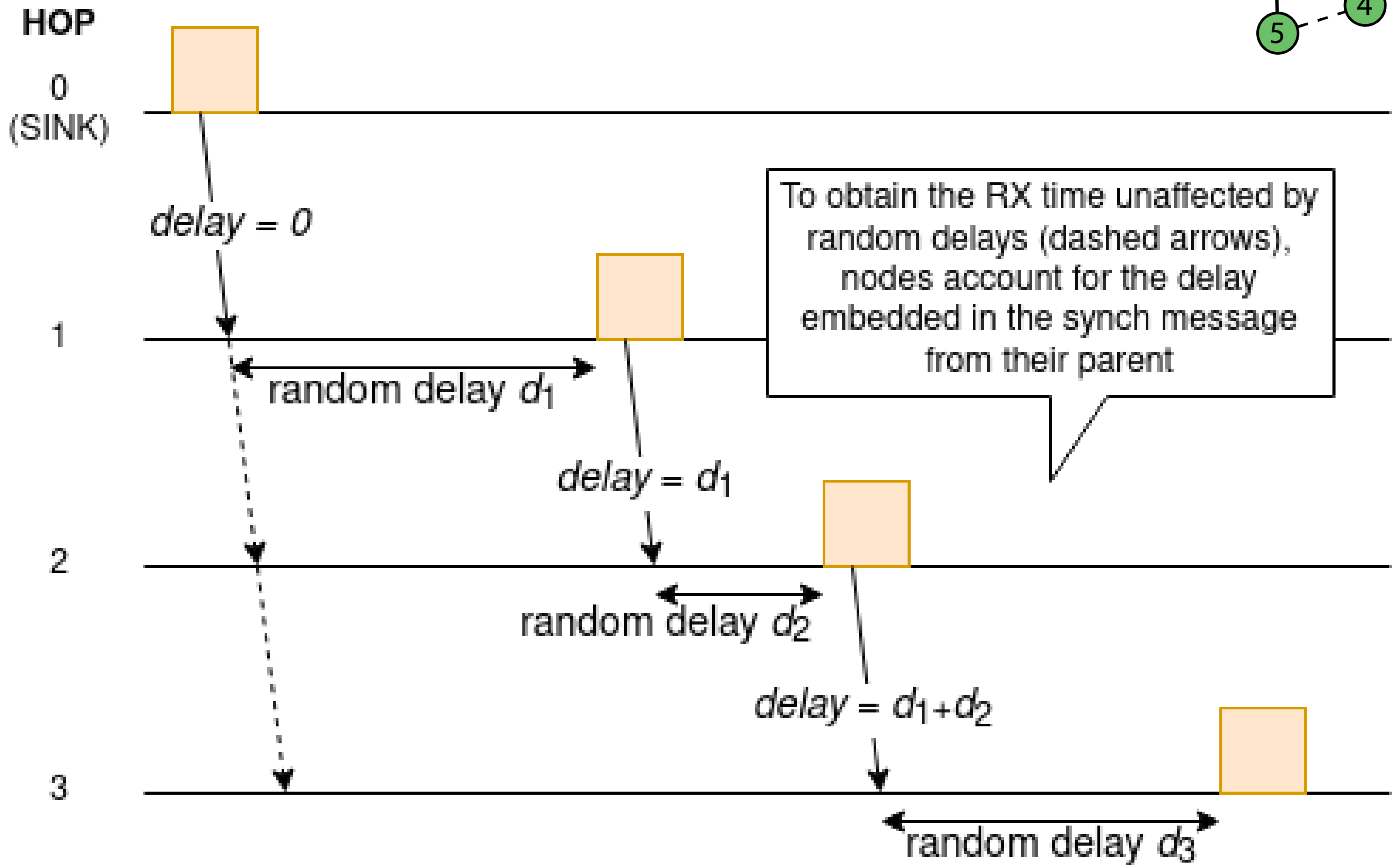
Time Synch + Periodic Data Collection



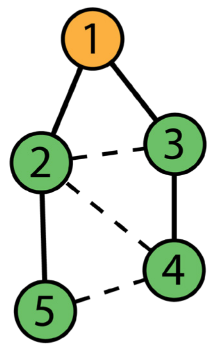
Synchronization Phase

- Same flood as in the data collection protocol of Lab 6-7 to build the routing topology.

- Nodes embed in their packet payload the **accummulated delay** from the transmission of the first packet from the sink.

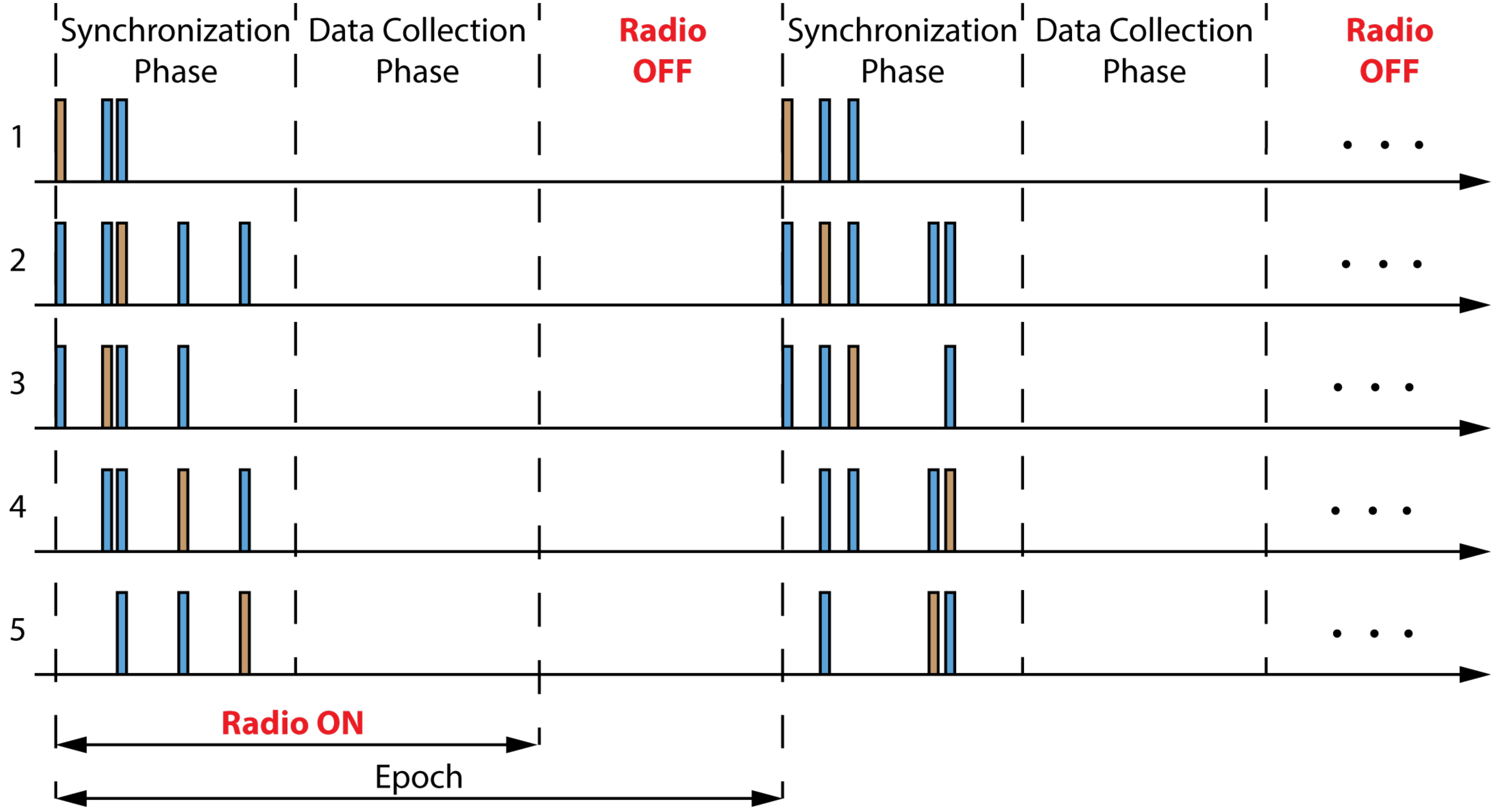


Time Synch + Periodic Data Collection

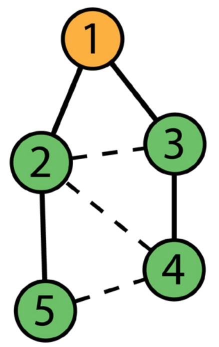


Data Collection Phase

- The phase starts at a "known" time **t** after the transmission of the first sink beacon assuming:
  - Max. random delay.
  - Max. number of hops.



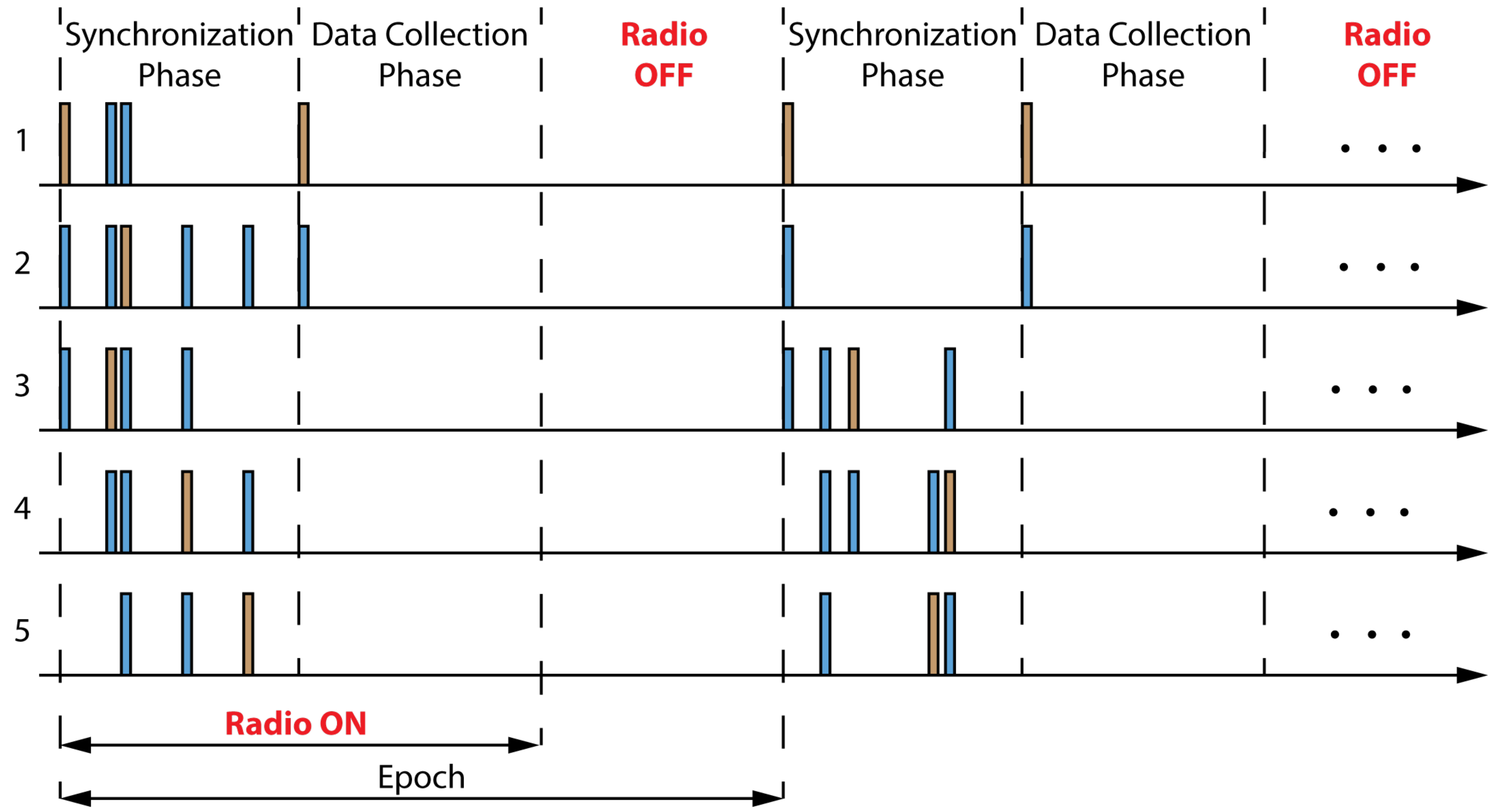
Time Synch + Periodic Data Collection



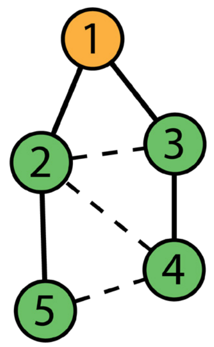
Data Collection Phase

- The phase starts at a “known” time **t** after the transmission of the first sink beacon assuming:
  - Max. random delay.
  - Max. number of hops.

- Nodes should transmit in an ordered sequence based on the node ID.



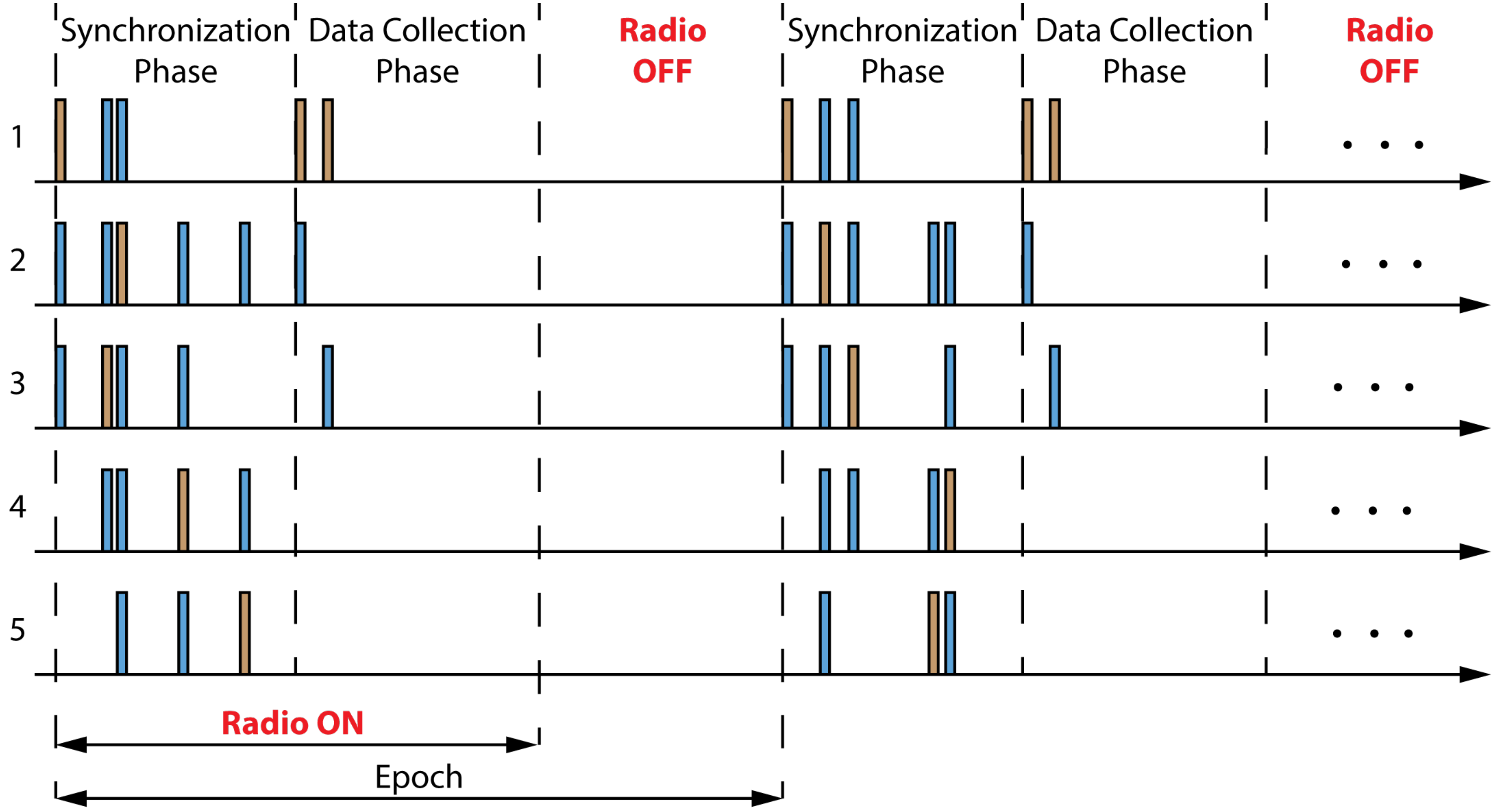
Time Synch + Periodic Data Collection



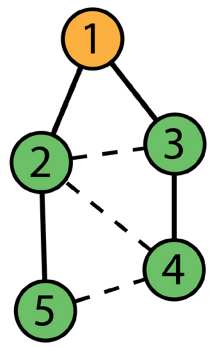
Data Collection Phase

- The phase starts at a "known" time **t** after the transmission of the first sink beacon assuming:
    - Max. random delay.
    - Max. number of hops.

- Nodes should transmit in an ordered sequence based on the node ID.



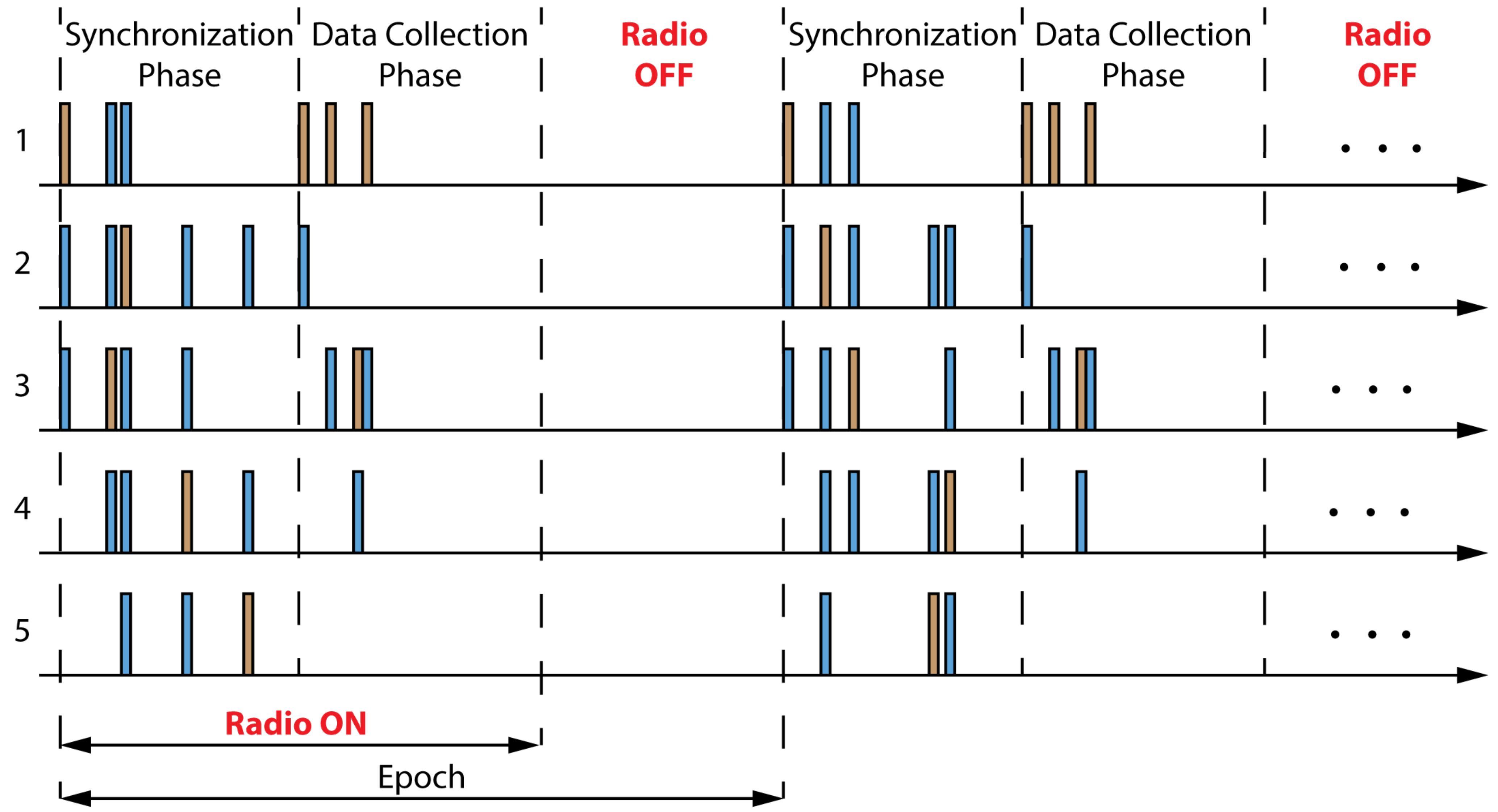
Time Synch + Periodic Data Collection



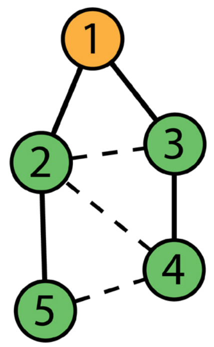
Data Collection Phase

- The phase starts at a "known" time **t** after the transmission of the first sink beacon assuming:
  - Max. random delay.
  - Max. number of hops.

- Nodes should transmit in an ordered sequence based on the node ID.

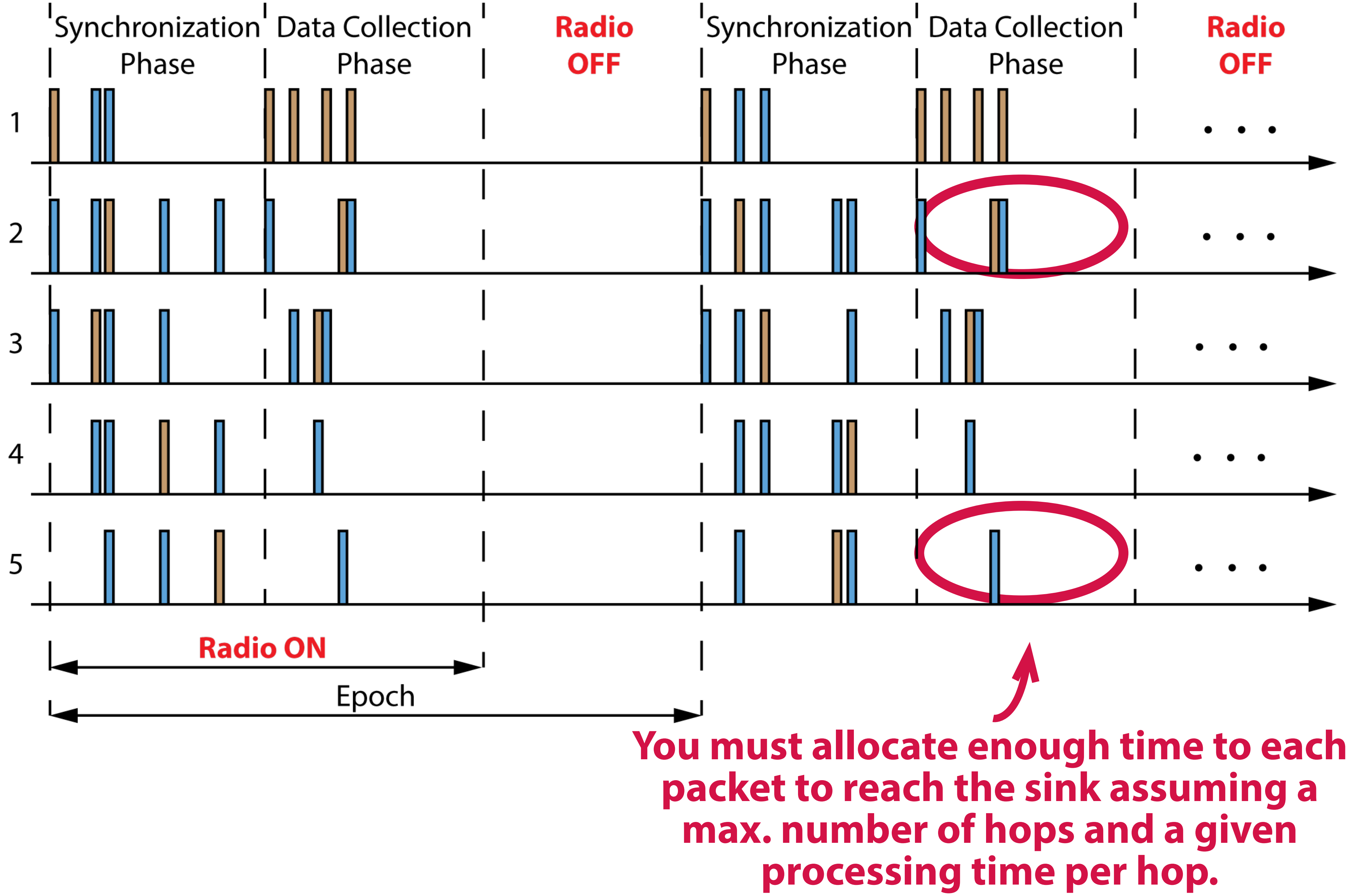


Time Synch + Periodic Data Collection



Data Collection Phase

- The phase starts at a "known" time **t** after the transmission of the first sink beacon assuming:
    - Max. random delay.
    - Max. number of hops.
- Nodes should transmit in an ordered sequence based on the node ID.

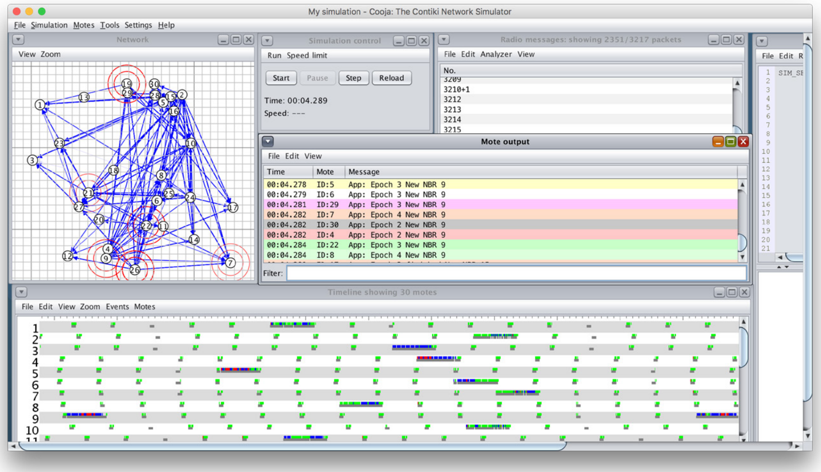


Performance Evaluation

1

Cooja Simulations

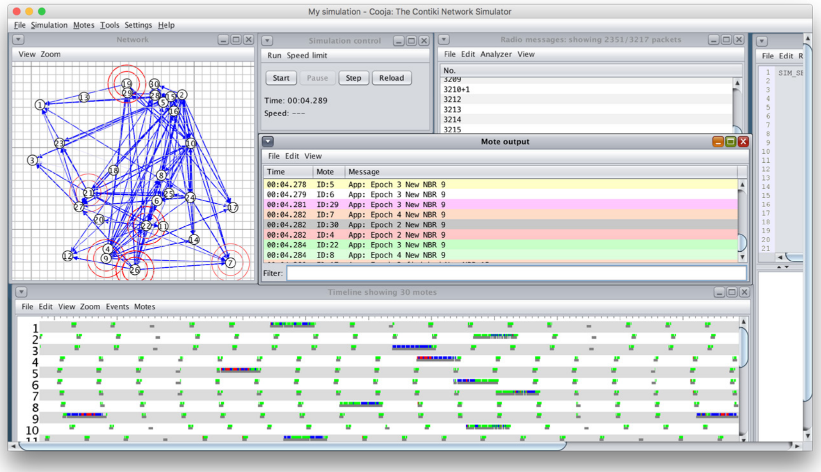
Testbed Experiments



Performance Evaluation

1 Cooja Simulations

Testbed Experiments



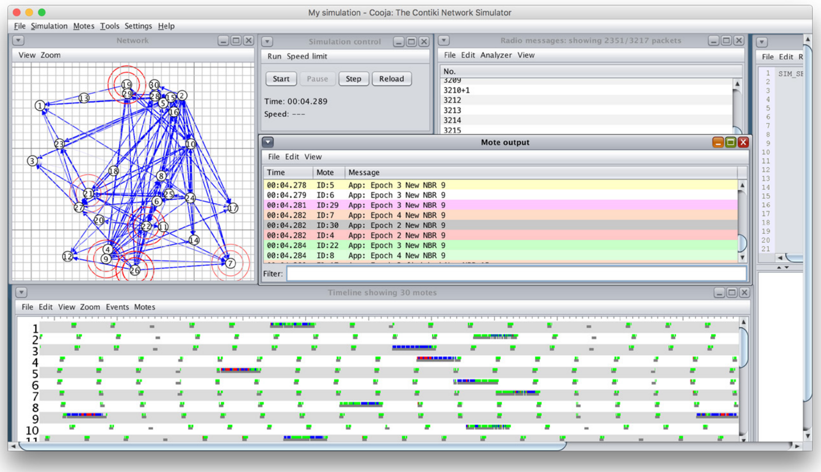
2 Log

Files

Performance Evaluation

1 Cooja Simulations

Testbed Experiments

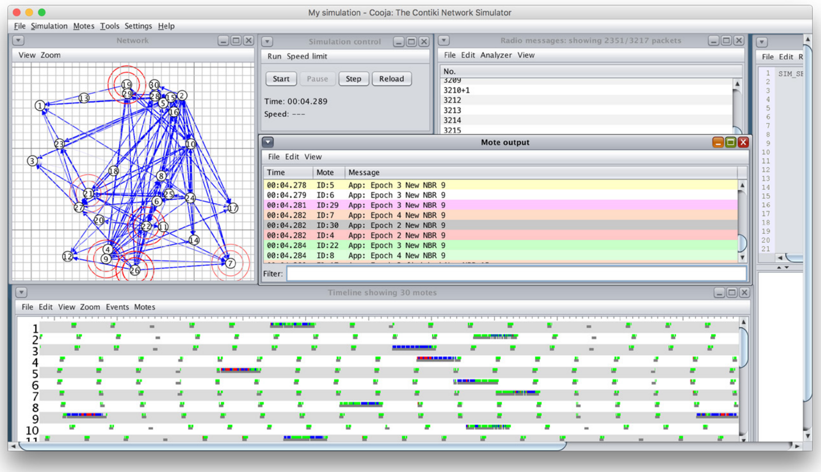


2 Log Files

3 Parser Script

Performance Evaluation

1 Cooja Simulations
Testbed Experiments



2 Log
Files

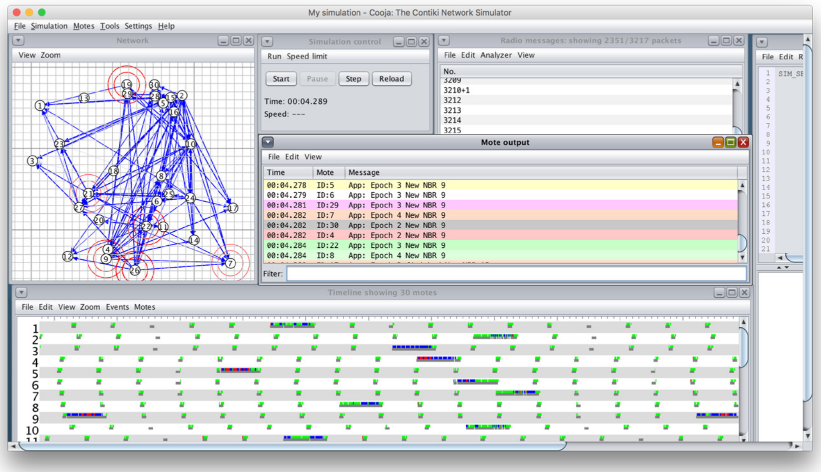
3 Parser
Script

4 Analysis
Scripts

Performance Evaluation

1 Cooja Simulations

Testbed Experiments



2 Log

Files

3 Parser

Script

4 Analysis

Scripts

5 Write

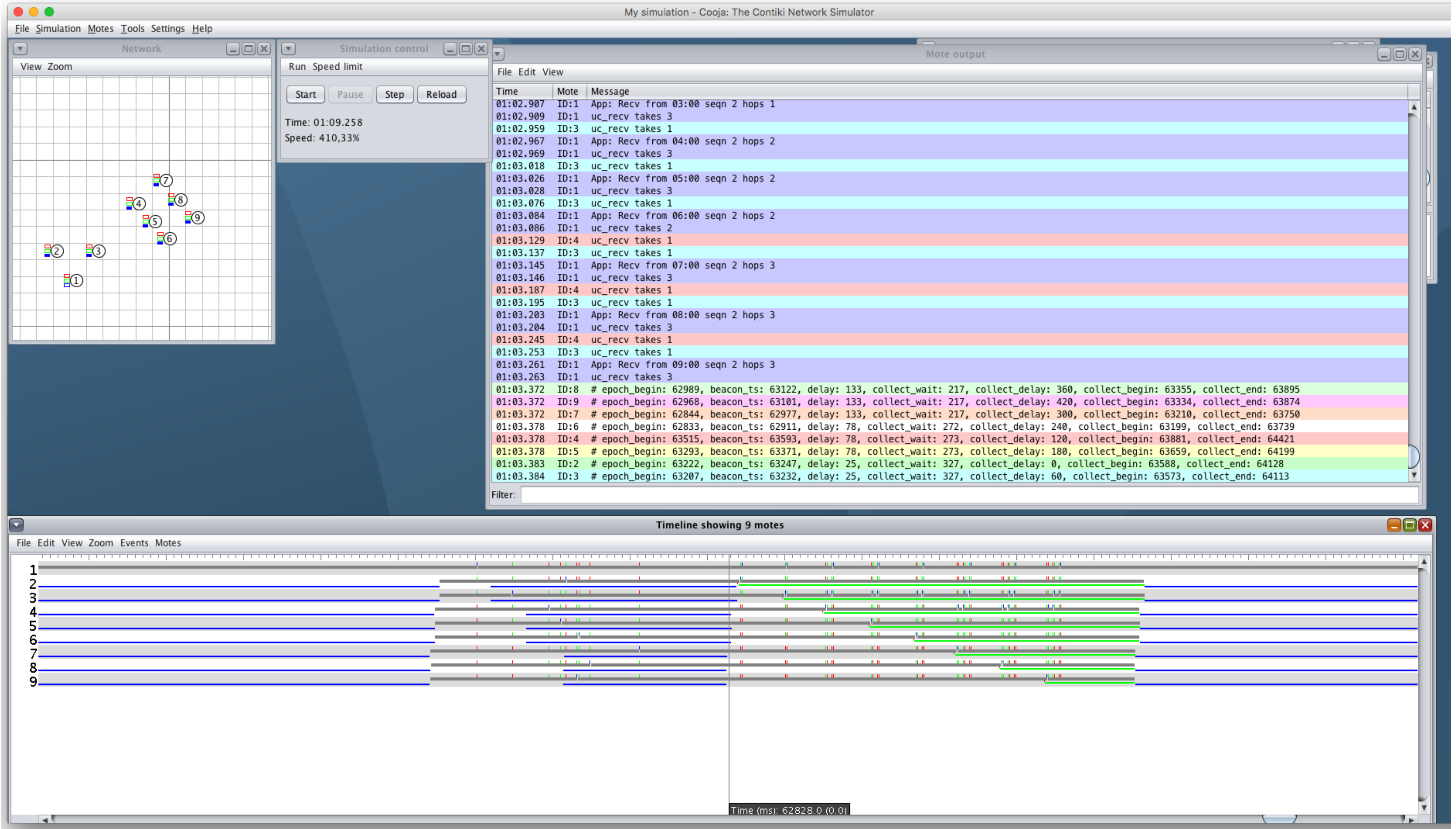
Report

Cooja Simulations

Files: test-nogui-*.csc

How to run simulations:

- $ cooja test-nogui-udgm.csc
- $ cooja_nogui test-nogui-udgm.csc



Log files:

- test.log ⟶ PDR & Duty Cycle
- test_dc.log ⟶ Duty Cycle

Approach:

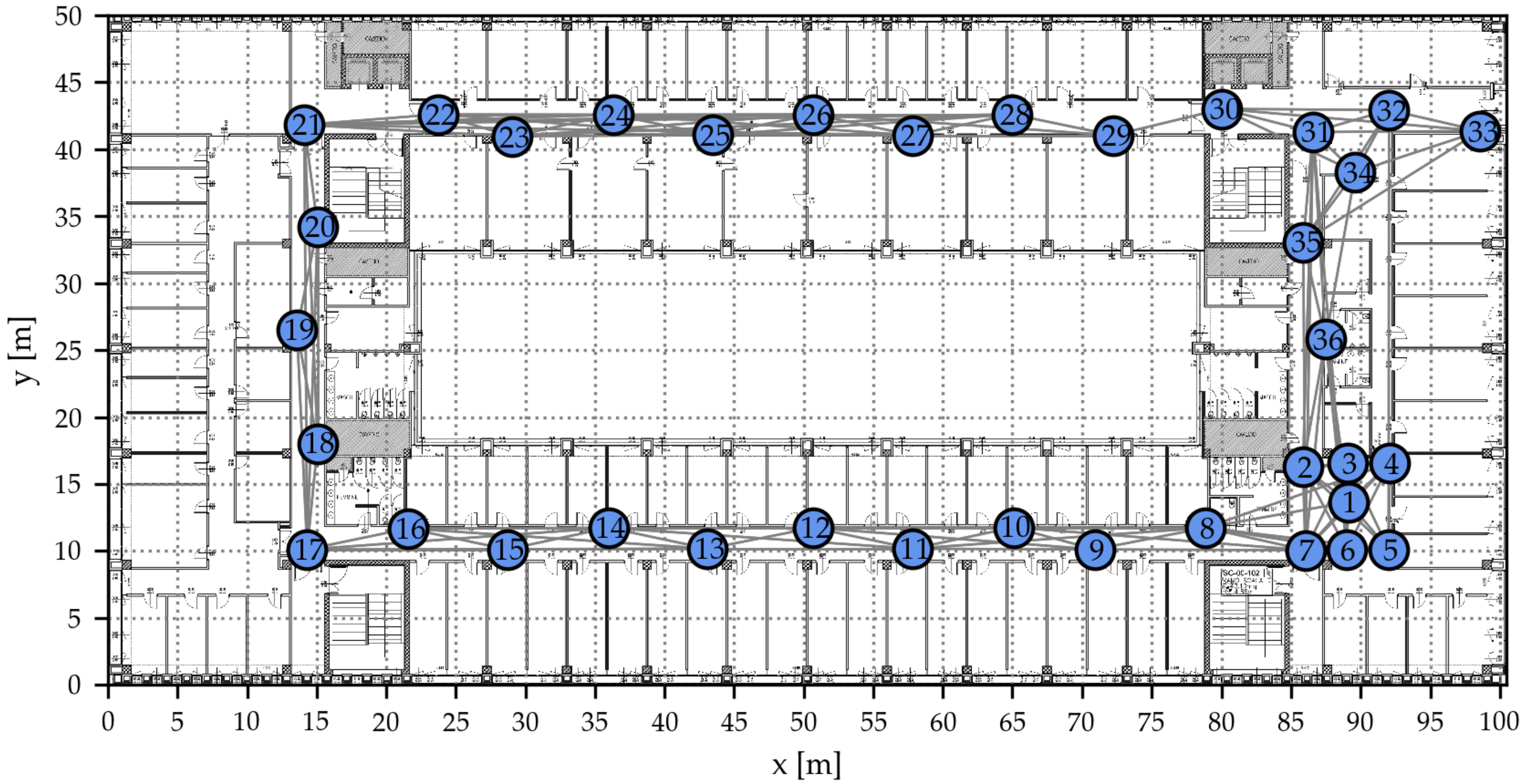
Run multiple simulations per network topology changing the random seed, radio model, mote start delay, etc.

Analysis: $ python parse-stats.py test_nogui_udgm.log

Testbed Experiments

How to run experiments:

- Connect to UNITN VPN
- $ make TARGET=zoul
- $ cp app.bin testbed/app.bin
- $ cd testbed
- $ bash run-test.sh
- $ bash get-test.sh job_id_number



Log files:

- test.log ⟶ PDR & Duty Cycle

Analysis:

$ python parse-stats.py testbed/job_X/test.log --testbed

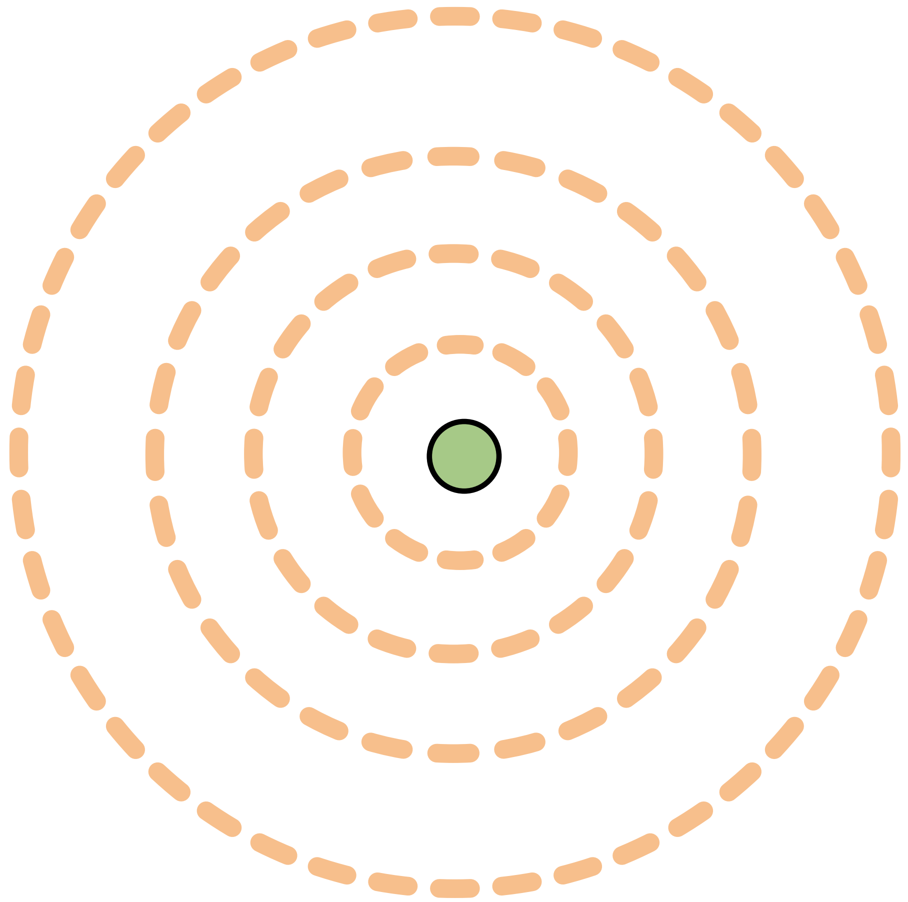
IMPORTANT NOTE: testbed experiments are optional.

However, the maximum mark without testbed experiments is 27/30.

Implementation Notes:

How to turn ON and OFF the radio:

- Turn the radio on
  NETSTACK_MAC.on();
- Turn the radio off
  NETSTACK_MAC.off(false);



Implementation Notes: Clock Time Module

API Description: contiki/core/sys/clock.h

- (Redefined) Frequency: 1024 Hz $\longrightarrow$ ~1ms
- Clock timestamp declaration:
  clock_time_t ts;
- Important function to timestamp RX packets:
  clock_time();
- Number of ticks in a second:
  CLOCK_SECOND

Implementation Notes

Application Interface:

Inform the application about received packets at the sink via a callback function.
Very similar to the callback in Lab 6-7.

void (* **recv**)(const linkaddr_t *originator, uint8_t hops);

Open a collection connection:

void **sched_collect_open**(struct sched_collect_conn* conn,
uint16_t channels, bool is_sink,
const struct sched_collect_callbacks *callbacks);

Allow application to schedule a data packet to be sent:

int **sched_collect_send**(struct sched_collect_conn *c, uint8_t *data, uint8_t len);

The data packet must be stored in a local buffer until sent in the data collection phase. If the packet cannot be stored (e.g., because there is a pending packet to be sent), then the function should return 0 to inform of failure. Otherwise, it should return 1 or non-zero.

Implementation Notes

Scheduling data packets

- Use **node_id** variable
- Assume maximum number of hops (as per the description) and study the processing time per hop.

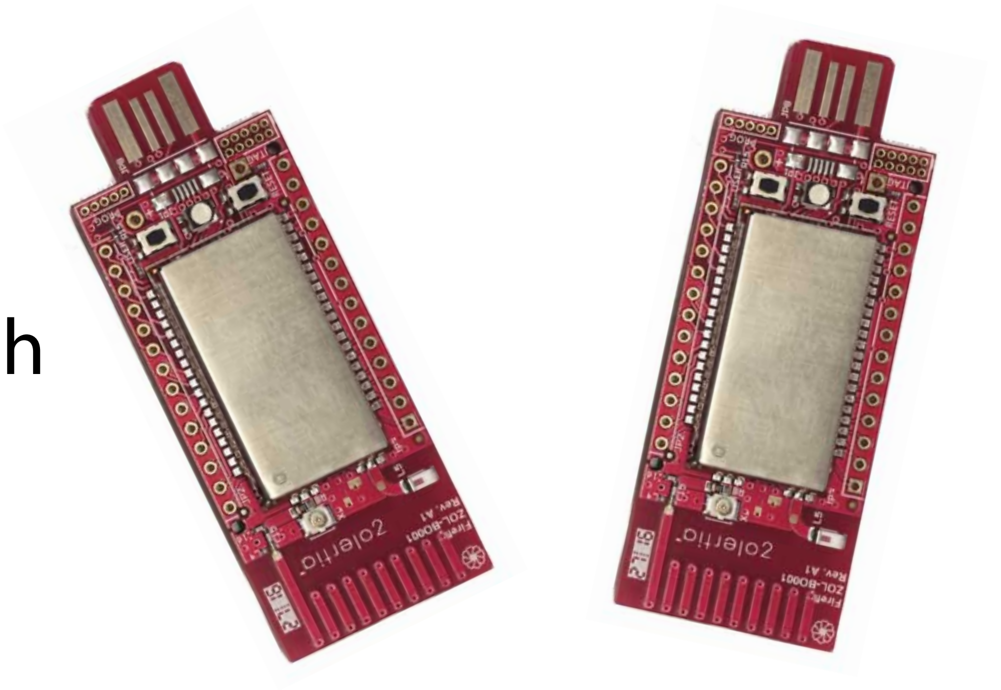
Implementation Style

- Processes or Protothreads
- Callbacks

We recommend a mixed programming style

Zolertia Firefly (testbed)

Modify CLOCK_CONF_SECOND in contiki/platform/zoul/contiki-conf.h
#define CLOCK_CONF_SECOND 1024UL



Project
Code Template

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

Q

Any Questions?