Course Project: Source Routing for Downward Data Traffic

Laboratory of Wireless Sensor Networks, 2016–2017

In this project, you will extend the data collection routing layer that you have implemented in class with a simple multi-hop source routing protocol. As a result you should have a routing layer (a nesC component) that supports two traffic patterns: many-to-one, allowing network nodes to send data packets up to the sink (root) and one-to-many, which is the exact opposite of the first one, with the root sending unicast data packets to other network nodes down the collection tree.

Source routing. In source routing algorithm, each data packet contains complete routing information to reach the destination. These routes are computed by the source node (the root in our case) and piggybacked in the packet header. A key advantage of source routing is that intermediate nodes do not need to maintain routing tables in order to forward the packets, since the packets themselves already contain all necessary information. The protocol will be divided into two parts, a *routing* part and a *packet forwarding* part.

Routing. How routes are constructed?
For the root to be able to construct the routes it needs to know the network topology (connectivity graph). In the case of one-to-many routing a spanning tree would suffice instead of the full connectivity graph. You have already implemented a distributed algorithm for building the spanning tree for data collection and the same tree can be reused for the one-to-many traffic, provided that the sink collects enough information from the nodes to successfully reconstruct it.

To build a tree it is sufficient to know for every node which node is its parent. Therefore, whenever a node selects or changes the parent, it should send a packet containing its own ID and the ID of its parent to the root using the data collection protocol already implemented in the labs. The sink node keeps a table of those parent-child relations for each node in the network.

For example, let us consider the following network: When R2 joins the network and chooses R1 as the parent, it sends a packet to the root saying **R1 is the parent of R2**. Now the root (**S**) makes an entry for the R2 into the source route table with the previous hop address set as R1 as shown in Table 1 below. This indicates that to reach R2, the packet needs to be forwarded through R1. If an entry for R2 was already present, it should be updated with the newly received information.

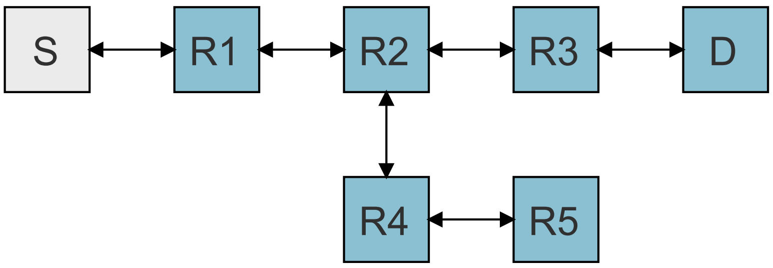


Table 1: Source Routing Table

Child	**Parent**
R1	S
R2	R1
R3	R2
D	R3
R5	R4
R4	R2

Packet Forwarding (one-to-many communication)
In addition to the many-to-one forwarding that you already have, your network layer should provide one-to-many delivery service to the application. This means that the application running on the root may request sending data to any node in the network and the application at an ordinary network node should be notified when a packet destined to the current node arrives. To implement this, the root should build the path (the

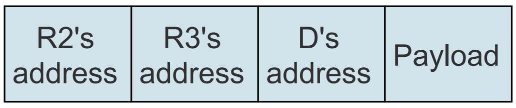
sequence of forwarders) to the destination and put this list to the packet header. The intermediate nodes (forwarders) when receiving a packet should forward it to the next forwarder in the list or deliver it to the application if the destination is the current node. More formally, the algorithms of the root and a forwarder are described in the following.

1. **At the root**: When the application at the root node **S** requests to send a packet to the destination node **D** the following algorithm should be executed:

    1. Assign N := D
    2. Search for node N in the routing table as constructed above to find N's parent P
    3. If N is not found or a loop is detected, drop the packet
    4. If P == root, transmit the packet to next-hop node N
    5. Else add N to the source routing list of the packet, assign N := P, go to step 2.

2. **At the forwarders**: When an intermediate node receives a packet to be forwarded, it modifies the routing header by removing the next hop node address from the list and transmits the reduced packet with the payload to the next hop node. If a node receives a packet with empty forwarding list, it delivers the packet to the application. Same procedure is repeated until the packet reaches the destination.

Packet Format. The data frame should contain the source routing list and the payload as shown below:



Miscellaneous Notes

- All the information necessary for forwarding should be contained in the data packet header. The number of address entries in the header is equal to the number of nodes on the path minus one.
- Source route entries **SHOULD NOT** repeat as this will cause unwanted loops. Any loops should be detected by the root while constructing the list of forwarders. If there is a loop, the packet should be dropped.
- Assume that the number of nodes in the network and the maximum path length are bounded. Use C defines to set these parameters using reasonable values (e.g., MAX_NODES=30, MAX_PATH_LENGTH=10).

Supported data flows. The system should support bidirectional communication: one-to-many unicast and many-to-one data collection traffic to and from a predefined node S.

Application interfaces. In addition to the data collection application interface, your networking layer should provide a new interface for one-to-many delivery. Similarly to the many-to-one data collection interface implemented in the labs, it should provide a **send** command and a **receive** event. The send command should accept the packet destination node ID as a parameter. The send command called at a non-sink node should return with an error immediately. The receive event should be signaled on the non-sink nodes only.

Rules of the game.

- The project is individual and the student is strongly suggested to deliver it before the beginning of the second semester, and in any case no later than September 2017.
- You should submit the TinyOS code and a short report and later demonstrate that the project works as expected using COOJA Simulator.
- The code MUST be properly formatted. Follow style guidelines (e.g. this one).
- You MUST contact through e-mail the instructor (gianpietro.picco@unitn.it) AND the teaching assistants (piyare@fbk.eu, timofei.istomin@unitn.it), well in advance, i.e. a couple of weeks, before the presentation.
- Both the code and documentation must be submitted in electronic format via email at least one day before the meeting. The documentation must be a single self-contained pdf/txt. All code must be sent in a single tarball consisting of a single folder (named after your surname) containing all your source files.
- The project will be evaluated for its technical content (algorithm correctness). *Do not* spend time implementing unrequested features - focus on doing the core functionality, and doing it well.
- The project is demonstrated in front of the instructor and/or assistant.

Plagiarism is not tolerated. An incomplete project will be considered more positively than one with parts of the code adopted from someone else.