Wireless Sensor Networks project: Source Routing for Download Data Traffic

July 11, 2017

Ardino Pierfrancesco 189159

The aim of this project is to implement a multi-hop source routing protocol. The routing protocol will support both many-to-one and one-to-many traffic patterns. In the first one all nodes will send data packets up to the sink, while in the second one the root has the possibility to send unicast data packets to other nodes in the collection tree. The project consists of these files:

- `MyCollection.h`: defines the structs needed for the project;
- `RoutingC.nc`: defines the Routing interface that provides the creation of the collection tree and wires the modules used and provided by RoutingP;
- `RoutingP.nc`: contains the implementation of the Routing interface;
- `OneToManyC.nc`: defines the OneToMany traffic pattern interface that provides the sending of unicast packet from the sink to a specific node in the network, moreover it wires the modules used and provided by OneToManyP;
- `OneToManyP.nc`: contains the implementation of the OneToMany interface;
- `ManyToOneC.nc`: defines the ManyToOne traffic pattern interface that provides the sending of packets from all the nodes in the networks to the sink, moreover it wires the modules used and provided by ManyToOneP;
- `ManyToOneP.nc`: contains the implementation of the ManyToOne interface;
- `AppP.nc, Test`: contain the implementation of the application that uses the Routing protocol with the two traffic patter interfaces.

1 Implementation

1.1 Initialization

At the beginning all nodes boot with the node with ID 1 set as the Root. In addition to that they all start *StartTimer*. Once the timer fired, the Root will start the **Build Tree** procedure and start the *PeriodicTimer* timer, which is started also by the other nodes, that periodically start a *OneToMany* or *ManyToOne* communication based on the type of node.

1.2 BuildTree

The sink is responsible for the building of the routing tree. When it receives the `buildTree` command from the application, it sets the `i_am_sink` variable to **TRUE**, starts the *RefreshTimer* timer that when fired will restart the **Build Tree** procedure and posts the `send_beacon` task that will broadcast a beacon. The payload of the beacon will contain the sequence number of the beacon and the current hops to the sink, which will be used in combination with the **RSSI**

as metric for choosing the parent of a node.
When a node has to send a beacon, it first check if the `sending_beacon` variable, if this variable is **TRUE** it means that the node is already sending another beacon and so it can not send the current one, while if the variable is **FALSE** it will fill the beacon payload and send the beacon, if there are some problems during the sending of a beacon the message will be rescheduled using the *NotificationTimer* timer.
When a node receives a beacon it will first check if it is the root, if so the beacon is discarded. Otherwise it will compute the **RSSI** between him and the sender of the beacon. If the **RSSI** is lower than a threshold the beacon is discarded, otherwise the node will compare the sequence number of the beacon, if the sequence number is older than the previous beacon than the packet is dropped otherwise if it is newer it means that the root started to rebuild the tree and so the parent of the node is set to the sender of the beacon. Lastly, if the sequence number is the same the node will check the hops to the root. If the number of hops to the root contained in the beacon is lower than the one stored by the node than the parent is updated to the sender of the beacon, otherwise if the number of hop is equal, the node will compare the **RSSI** and then update the parent. If the parent is updated the node will broadcast a new beacon with the newer distance from the root.

1.3 Routing Table creation

A routing Table is maintained by the root, each row of the routing table contains information about a node and its parent. When a node updates its parent, it sends a message to the root containing the information about the parent. If the node is already sending a **InfoBeacon** to the root, it will put the message in a temporary variable and reschedule the sending using the *InfoTimerRescheduling* timer that when fired, will fetch the information of the packet from the temporary variable.
When the root receives the message it will update its Routing table, adding a new entry or updating a existing entry.

1.4 Many-To-One traffic pattern

Using Locks in the functions that modify the money of a branch guarantees that a branch can not send and receive money in the same time. The same happens if a branch wants to start a new snapshot. It will acquire the lock, saves the current amount of money of the branch and then releases the lock. Tcp guarantees also FIFO ordering and the automatic retransmission of a packet and the management of acknowledgments. The money are withdrawn from the account of a branch only if the Tcp socket has successfully sent the packet. Using the option `MSG_WAITALL` on the receiver side, guarantees that the receiving socket waits until all the bytes will be received.

2 Testing

2.1 Network failures

Without network failures, the snapshot will eventually finish. In presence of network failures during a snapshot, it never terminate since a branch will wait forever for the delivery of a token from the others branch. The snapshot will continue when the failures will be fixed.
Network failures were simulated by removing the Ethernet for a couple of seconds and then reconnecting the cable and finally checking the log of the snapshot in order to check that no money had been lost.

2.2 Crash Failures

In the context processes crashes, the application was not designed to tolerate them. In presence of crash failures the snapshot will never terminate. The software on each branch has to restarted and unless there are snapshots already completed the money will be lost.

3 Launching example

3.1 Single machine

In this execution each branch is running on the same machine, the content of `host.list` is the following:

```
127.0.0.1:8000:S
127.0.0.1:8001
127.0.0.1:8002
127.0.0.1:8003
```

Thus four branch with the first that will start the snapshot.
The application is launched with the following command:

```
./start_bank.py 127.0.0.1
```

3.2 Multiple machines

Following there is an example with two machines, one with IP 10.0.0.1, the other with 10.0.0.2, the content of `host.list` is the following:

```
10.0.0.1:8000:S
10.0.0.1:8001
10.0.0.1:8002
10.0.0.2:8003
10.0.0.2:8004
10.0.0.2:8005
```

The application is launched on the first host with the following commands:

```
./start_bank.py 10.0.0.1
```

On the second host with:

```
./start_bank.py 10.0.0.2
```