Wireless Sensor Networks project: Source Routing for Download Data Traffic

July 10, 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, the branch is launched by the `start_banks.py` script with the IP address of the machine as argument. This script create a new process that instantiates a new `Bank` class for each different branch on the machine. After a period of initialization each branch is connected to all the others branches. Each branch will attempt to connect with the others branches every two seconds, until all the network is connected.

1.2 Money exchange

After the initialization procedure, each branch, after a period of time governed by a uniform variable between 2 and 5 seconds, spawns a thread which contains an infinite loop. In this loop it tries to acquire a lock on the amount of money owned by the branch. After having successfully acquire the lock it chooses randomly an amount of money and a branch, then it tries to send

this amount to the other branch through the `bankinterfaceout` class. If the operation finishes successfully the thread withdraws the money from the total amount otherwise it does not and finally releases the lock.

1.3 Global snapshot

The branch that has to start the Global Snapshot, spawns also another special thread. This threads periodically starts a new Global Snapshot. It tries to acquire the lock on the total amount and once it succeed, it records the local state and it sends to every branch a snapshot token. All the other branches process this token following the Global Snapshot protocol described during the lectures.

1.4 Safety guarantees

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