Giovanni De Novellis – 10608672

Challenge 3 Report – TinyOS and TOSSIM

Introduction

The goal of the challenge was to implement in TinyOS and simulate in Tossim a network of nodes, where each one can send messages containing data, or route request and reply to the neighbors nodes to discover new paths. Every node has a routing table, which is a data structure with entries for each member of the topology along with the cost needed to reach it and the next hop of the path.

Components explanation

Timer0: This is a timer component, responsible for triggering the send of a message. It is set after a generate_send request by the program and, after firing, the sending of the message is executed. While sending, other send requests are blocked.

Timer1: This is a timer component, responsible for simulating the Node 1 sending a data message. It is triggered for the first time when the Node 1 is booted and, when fired, it triggers the first try of sending the data message, which is not possible since the routing table is empty. It is set the second time again from Node 1 when it receives a route reply message for the node 7, this time triggering the true send of the message.

AMSenderC: This is the Tiny Os packet's abstraction; it allows to edit the payload of the message we need to send through the Packet.getPayload function.

AMReceiverC: This is the Tiny Os component responsible for handling the receive of the message. When a Node receives a message, an event associated with this component is triggered.

LedsC: This is the Tiny Os Led's abstraction. Through this component it is possible to read the status of the Leds and to toggle them when needed.

ActiveMessageC: Component used to handle the start of each node and the stop.

Functions and Events explanation

generate_send: This function is used to trigger the send of a message in the node. It stores the packet and the address of it in a variable and triggers the timer 0, which, when firing, will perform the actual send of the data.

Timer0.fired: This is the event called when the timer 0 fires. It simply calls the actual_send function that will send the queued packet to the needed destination and set the variable locked to True, which will avoid any new sending until the current one is completed.

actual_send: This function first checks if the variable locked (that signals if the node is waiting for the completion of a previous send) and if it is false, then proceed to send the input packet to the input address.

Boot.booted: Event called when the node boots, it simply prints a debug message and then triggers the startDone event.

AMControl.startDone: This event first check for any error and eventually retries to start, then if it is the node 1 it will trigger the logic to try to send the data, otherwise nothing will happen, and the node will wait for messages.

AMControl.stopDone: A simple debug message is printed in case of stop.

Timer1.fired: This event, attached to Timer1, is used to simulate the first node sending the data message. The first time it is triggered the routing table is empty, so the node will send a route request message for the node 7, while the second time it will be possible to send the data message using the next hop present in the table.

handleLeds: This function is called every time a node receives a new message of any type, it works by increasing the message counter, selecting the correct digit of the person code, using it to toggle the correct led and then printing a debug message with the new led status.

Receive.receive: This event is called when a node receives a message. First, it updates the status of the leds by calling the handleLeds function. Then, depending on the type of message received it will behave differently:

- If the message received is a data message, it will check if he is the destination of the message and eventually just print a debug message. Otherwise, it will forward the message to the node saved in the next-hop field of the routing table entry corresponding to the destination id of the message.
- If the message received is a route request, there are 3 different cases:
    - If the node is the requested one, it will reply in broadcast with a route reply message with himself as the requested id and 1 as cost.
    - If the node requested is present in the routing table, it will reply in broadcast with a route reply message with the requested node as the requested id and the routing table cost + 1 as the cost.
    - If the node requested is not present, it will generate a route request for that node in broadcast.
- If the message received is a route reply, first it will check if it is the node contained in the reply and eventually do nothing. Otherwise, it will check if the node is already present in the routing table and update it and forward the message with the cost + 1 to the neighbors if the node is not present or the new cost is lower.

AMSend.sendDone: Event called after the sending of a message is finished, it will make an error check and set to false the variable locked to allow new sendings.