Project 1

**Overview**

The goal of the project is to create a system for wireless contacts tracing inside a given location. The system must be able to collect contacts and persist them into the application backend. Also, it must provide to users the ability to trigger an event of interest, which is then notified to all users who has been in contact with the user who just triggered the event.

The application client is implemented in Contiki-NG, and it will be deployed (in production) in small low power microcontrollers (held by customers) with wireless communication capabilities (we used Cooja for simulating real world environment). The application backend is implemented in Node-RED, which exchanges messages with Contiki nodes using a Contiki RPL border router, which acts as a bridge between the two subsystems.

**Contiki subsystem**

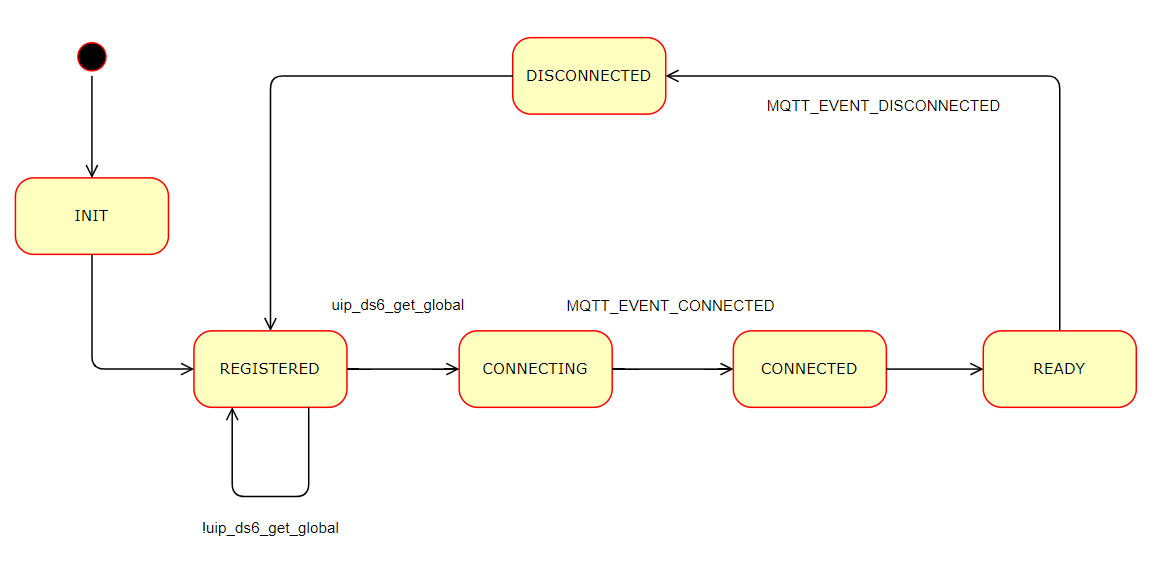
As mentioned above, the Contiki subsystem includes one standard RPL border router, which is basically using RPL transport protocol to handle application MQTT messages between user nodes and an MQTT broker, which is a Mosquitto broker process running in the Cooja virtual machine.

Contiki user nodes are considered in contact when they are in the wireless communication range; each personal device identifier is realized by a function of the link address of the node.

To capture most of the contacts, user nodes signal their identity (and receive others) with a higher frequency (n time higher) with respect to backend update; contacts are saved in an array list (size of n contacts) which is then flushed when sending to the backend; node already perceived are not appended; therefore, each node can detect up to n different other nodes for each backend update (in the implementation n is set to 2).

User nodes are of two types: client and client-triggerer. They are basically the same program: they both detect contacts, send them to the backend, and receive an event of interest triggered by someone else; client-triggerer also have a periodic event of interest triggering function (simulates user intention of notify the event).

All the nodes are made by two cooperatively scheduled Contiki protothread:

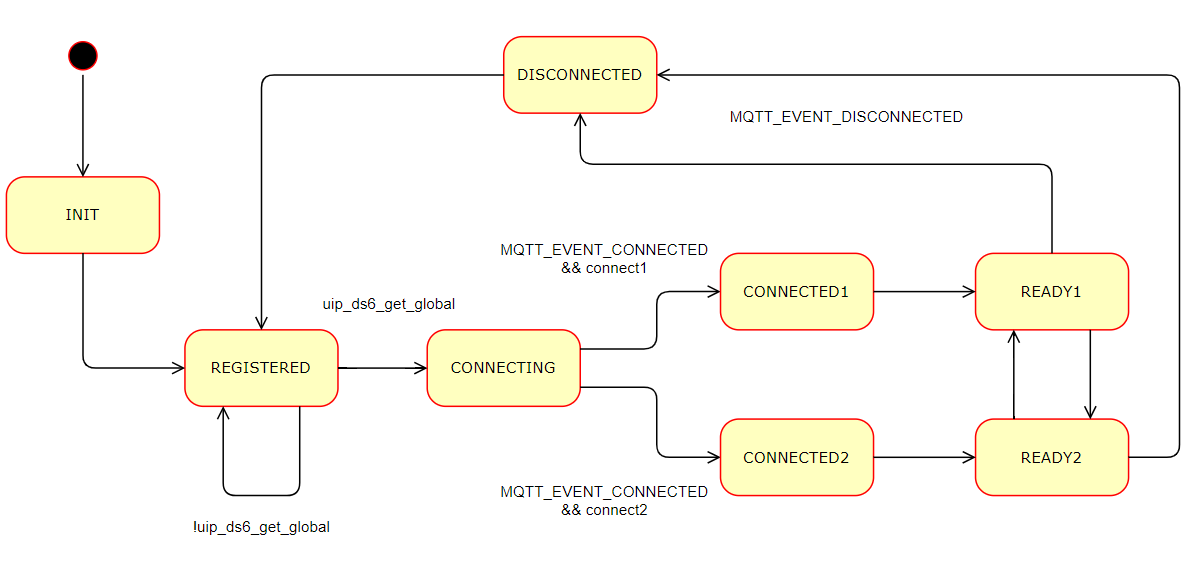
* **Broker process** handles the TCP connection with the MQTT broker, as well as MQTT message send/receive.
* **Broadcast process** handles contact tracing, by signaling itself to neighbor nodes using a UDP multicast transmission and listening for neighbors on the same connection.

State diagram of broker\_process of client.c

Client-triggerer node in addition:

* Have a **signal process** which triggers an event of interest to be sent.
* Use broker process in turns to flush contacts, then send (if signaled) the event.

State diagram of broker\_process of client-triggerer.c



QoS of sending contacts is set to 0; assuming contact dynamic last at least few seconds, and that detection has much higher frequency, we do not care losing few messages (we can detect again it later, while decreasing broker utilization).

QoS of sending events of interest is set to 1; we need to ensure that an event is actually sent to the backend; the event of interest is an important information, it can be a danger signal to other users (Immuni app analogy).

For the same reason, QoS of receiving an event is set to 2; we need to guarantee that the event is received, but we do not want to dispatch it multiple time (no fake event of interest).

(NODE-RED)