## Software

### Front-end Software

This section describes the firmwares that define tag, detector and proxy behaviors. The codes are adapted to Arduino platform, using Arduino and RF12 library (Jeelabs)

#### Basic communicating

##### Tag

A tag periodically broadcast its ID and battery level every *t* seconds, goes to sleep, then wakeup and continue another broadcast.

Initialize MCU & Radio

Broadcast

Sleep

Battery check

Notice that tag performs battery check at beginning of every cycle.

##### Detector

Detectors are responsible for:

* Pick up data from tag and determine tag’s signal strength and transmit
* Pick up data from other detectors and relay

Initialize MCU & Radio

Listening for packet

Battery check

Read RSSI

Construct New Packet

From tag?

Yes

Modify packet route

No

Transmit packet

##### Proxy

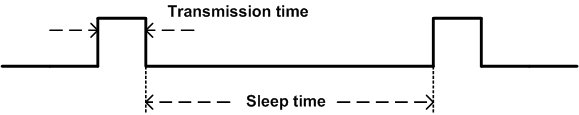
#### Power Management

##### Battery life

* C = capacity [mAh]
* I = active current [mA]
* T = transmit ratio = transmission time / transmission cycle

|  |
| --- |
| battery life (hours) = C/(I x T) |

We have this on the assumption that we *ignore the sleep current*.



For example, assuming that

* Transmission time: 100ms every 10 seconds
* Active current 100mA (RF transceiver + MCU)
* Negligible sleep current

Approximate battery life is

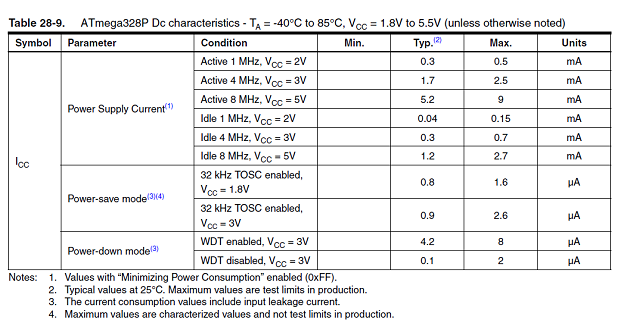
##### Transmission time

It is good to know the transmission time of the module so that we can calculate the transmission ratio. Also, transmission time place an important role in collision avoidance (the shorter transmission time, the smaller chance of collision).

|  |
| --- |
| transmission time = overhead + (number of transmitted byte x 8/baudrate) |

##### Sleep mode

The sleep mode (implemented on tags) exploits power-down mode in ATMega328P and uses watchdog timer to periodically wake up the chip.

The ATMega328P is Picopower version. Here's the DC characteristic from datasheet:

Tag with sleep mode program flow chart

Initialize MCU/Radio/WDT

Broadcast

Turn radio off

Battery check

Sleep

Turn radio on

WDT interrupt/Wakeup

#### Mesh protocol

For a detector to determine the RSSI of an asset tag, the asset tag must broadcast a message. The message has six fields, as shown in Figure 3-4. The way the fields are used depends on the origin of the message. There are two possible origins of a message. The first type of origin is an asset tag. A message can be identified as coming from an asset tag by examining the DID, SID, and RSSI value fields. If all three fields contain the value zero, then the message originates from an asset tag. Otherwise, the message originates from a detector.



Figure : Message format.

When receiving a message whose DID is not zero, a detector will do four things:

1. Replace the DID of the message with a DID contained in its routing table
2. Replace the SID of the message with its own ID
3. Replace the RSSI value with that inferred from the reception of the asset tag broadcast
4. Rebroadcast the message

An example of this message passing sequence is show in Figure 5. **(Need modified)**

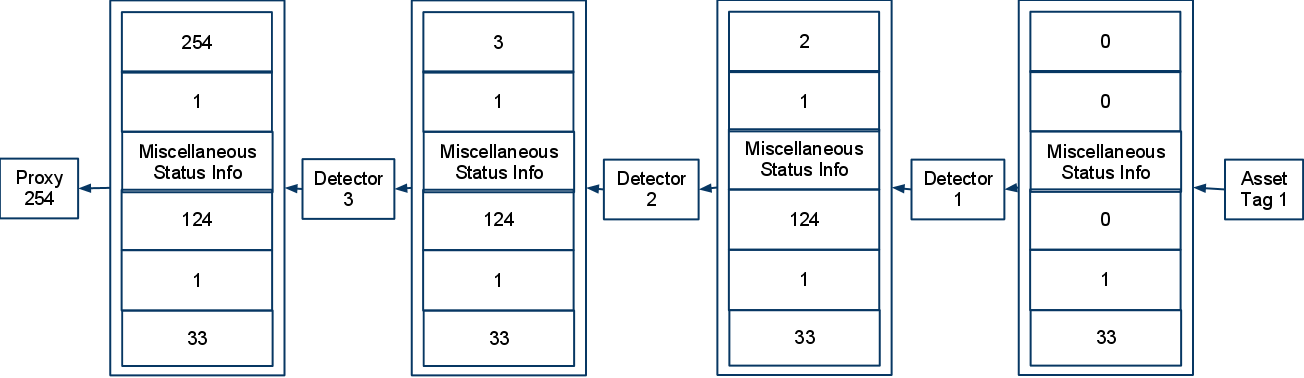


Figure : A example of a message originating from a asset tag, and being relayed to a proxy.

### Back-end Software

#### SQL Database

#### Web application

#### Controller

#### Location Engine

The location engine receives RSSI values from the controller and uses those values to calculate the locations of the asset tags. The location engine employs fingerprint algorithm which compares tag RSSI to sets of calibrated RSSI to find a best matching set. This is based on the fact that each location has a unique and consistent RSSI pattern. Since the calibrated RSSI set is associated with a calibrated location, the engine can use this result to provide location of the tag.

Data recorded from calibrating are RSSIs from a tag placed at the calibrating location (these signals measured by the surrounding detectors).

##### Euclidean distance

Fingerprint matching uses Euclidean distance calculated as follow:



Where:

* d = Euclidean distance
* N = Number of detectors
* RSSIli = RSSI value from detector i in locating phase
* RSSIci = RSSI value from detector i in calibrating phase

The minimum Euclidean distance from the set of calculated distances indicated a best matching result.

##### Aliasing

Aliasing is an issue of the fingerprint approach since two arbitrary locations can accidentally have a similar RSSI pattern, resulting in similar Euclidian distances. This causes incorrect or unstable output results.

To address aliasing, the algorithm needs to determine whether there are possible aliasing results in the calculated Euclidean distance set. If there is such case:

* If the locations are close to each others, then the algorithm chooses to interpolate their locations to provide location of tag.
* If the two locations are not close to each others, the algorithm will decide to referencing the nearest detector using it strongest RSSI in calibration set before determine location of tag

##### (n-1) supporting model

Locating algorithm uses a so-called (n-1) model to support its decisions. The (n-1) model performs n Euclidean distance calculations on n-1 RSSI values of the incoming RSSI set and at each time, an RSSI value from a detector is left out. This will give n results. Since there are less RSSI in used, prediction from this model will likely to:

* support the main prediction when it is unsure about a decision
* Eliminate effect of bad data from single detector

Core algorithm flow diagram:

Calculate Euclidean distances

Sort distance list

Aliasing

Yes

Referencing (n-1) model

No

Adjacent locations

Interpolating

Referencing nearest detector

No

Yes

Give Tag location

##### Input & output filter

Input filter use previous input to predict noises on current input packet. If the filter finds that input is unstable, it will ignore the packet and return the previous location of tag.

The output filter’s purpose is to provide stabilized output when decision of tag location is on unstable period.

Data flow diagram:

Extract input packet from controller

Compare to previous packet

Is stable

Yes

Retrieve previous tag location

No

Send to fingerprint engine

Send Tag location to output filter

Store tag location (engine output)