ROCARSYS

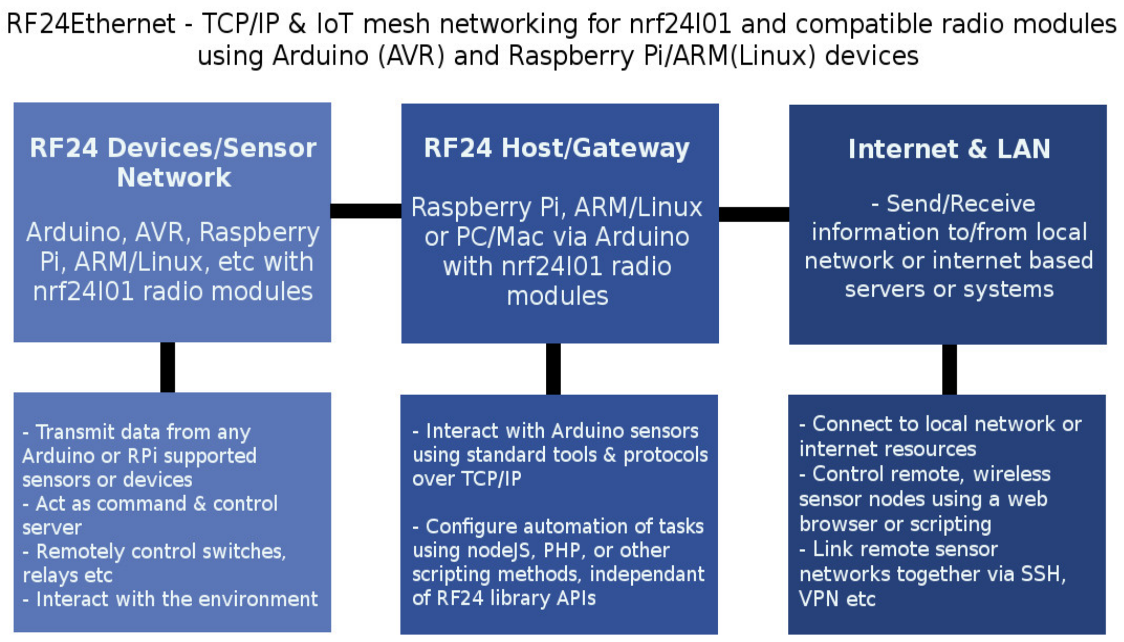
Rocarsys controls model railroads.

Rocarsys is built from different components. Those components are glued together according to the design schematics.

The transmission is wireless. NRF24 modules establish the radiolinks.
TCP/IP over RF24 network assembles the network using the radio links.
The RF24 Host/Gateway runs on RASPI. The related client software runs on Arduino Mega.
The messaging system uses the MQTT protocol over TCP/IP.
The MQTT broker, Mosquitto, runs on Raspi.
The MQTT client software runs on Arduino Mega.
The Rocduino Master runs on Raspi.
Rocduino Master subscribes to sensor data on Mosuitto.
Rocduino Master publishes control data on Mosquitto.
Rocduino Master publishes status data on Mosquitto.
The Rocduino decoder runs on Arduino Mega.
The Rocduino sensor decoder publishes sensor data on Mosquitto.
The Rocduino turnout decoder subscribes to control data on Mosquitto.
Rocduino Master exchanges data between Rocrail and Mosquitto via UDP and IP.
Rocarsys webserver runs on Raspi. It publishes status data to webbrowsers.

Available software components:

TCP/IP over RF24 network.


RF24Ethernet - TCP/IP & IoT mesh networking for nrf24l01 and compatible radio modules using Arduino (AVR) and Raspberry Pi/ARM(Linux) devices
RF24 Devices/Sensor Network
Arduino, AVR, Raspberry Pi, ARM/Linux, etc with nrf24l01 radio modules
RF24 Host/Gateway
Raspberry Pi, ARM/Linux or PC/Mac via Arduino with nrf24l01 radio modules
Internet & LAN
- Send/Receive information to/from local network or internet based servers or systems
- Transmit data from any Arduino or RPi supported sensors or devices
- Act as command & control server
- Remotely control switches, relays etc
- Interact with the environment
- Interact with Arduino sensors using standard tools & protocols over TCP/IP
- Configure automation of tasks using nodeJS, PHP, or other scripting methods, independant of RF24 library APIs
- Connect to local network or internet resources
- Control remote, wireless sensor nodes using a web browser or scripting
- Link remote sensor networks together via SSH, VPN etc

The messaging system.

Client

When talking about a client it almost always means an MQTT client. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That's why MQTT is ideally suitable for small devices. **MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET.**

Broker

The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. **The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients.** It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. All in all the broker is the central hub, which every message needs to pass.

MQTT Connection

The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack.

The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. **The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK** and a status code. Once the connection is established, the broker will keep it open as long as the client doesn't send a disconnect command or it looses the connection.

Eclipse Mosquitto™ is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 3.1 and 3.1.1. MQTT provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for "Internet of Things" messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers like the Arduino.

Raspberry Pi

Mosquitto is available through the main repository.

There are also Debian repositories provided by the mosquitto project, as described at http://mosquitto.org/2013/01/mosquitto-debian-repository/

The Rocduino sensor decoder software is tested. Publishing to Mosquitto has to be added.
The Rocduino turnout decoder software is tested. Subscribing to Mosquitto has to be added.

Rocduino Master sensor data collection and control data distribution is tested. Status data is available.
Publishing and subscription to Mosquitto has to be added.
Transferring data between Rocrail and Mosquitto has to be developped.

Rocarsys website has to be developed.

Way ahead.

Deploy the TCP/IP over RF24 network.
Required components:

- RASPI
- Two Arduino Mega
- Three NRF24 modules

Implement the MQTT client on both MEGA's.
Implement MOSQUITTO on RASPI.

Adapt the Rocduino decoders on Arduino MEGA to MOSQUITTO.

Port Arduino Master to RASPI. Enhance Rocduino master with the lacking functionality.

Rocrail is able to communicate over UDP via the Rocnet protocol. Investigation is required if a more direct connection between Rocrail and MOSQUITTO via TCP/IP is beneficial.

The status website has to be developed.