**Mestra**

Communication Between Devices

Michel Keijzers, © 2017

# History

Table 1: History

|  |  |
| --- | --- |
| **Date/period** | **Actions** |
| Nov 8, 2017 | Initial Version |

# Table of Contents

Contents

[History 1](#_Toc498098983)

[Table of Contents 2](#_Toc498098984)

[List of Tables 3](#_Toc498098985)

[List of Figures 3](#_Toc498098986)

[1 Introduction 4](#_Toc498098987)

[2 Alternatives 5](#_Toc498098988)

[3 Aspects 6](#_Toc498098989)

[3.1 Network Topology 6](#_Toc498098990)

[3.2 Slave Presence 6](#_Toc498098991)

[3.3 Loop 6](#_Toc498098992)

[4 RF 7](#_Toc498098993)

[4.1 Proof of Concept 7](#_Toc498098994)

[4.2 NRF24L01 7](#_Toc498098995)

[4.3 One or Two RFs per Device 7](#_Toc498098996)

[4.4 Best Channel Initialization 7](#_Toc498098997)

[4.5 Normal Operation 8](#_Toc498098998)

[4.6 Acknowledgements 8](#_Toc498098999)

[4.7 Reducing Messages 8](#_Toc498099000)

[4.8 Multiple Messages 9](#_Toc498099001)

[4.9 Retries 9](#_Toc498099002)

[4.10 Payload 9](#_Toc498099003)

[4.11 Alive Message 9](#_Toc498099004)

[4.12 Diagnostic LEDs 9](#_Toc498099005)

# List of Tables

[Table 1: History 1](#_Toc498099006)

[Table 2: Communication Comparison 5](#_Toc498099007)

[Table 3: Generic Diagnostics LEDs 10](#_Toc498099008)

# List of Figures

**No table of figures entries found.**

# Introduction

This document describes the communication between different Mestra devices (both from Controller to slaves and vice versa).

For the combined device which has the Controller and one or more slaves inside, the communication will be handled in the specific document.

# Alternatives

Many options are available for the communication between the controller and slaves. The most important is REQ Gen1.

Table 2: Communication Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| **Item \ Type** | **Cables** | **WIFI** | **RF** |
| Cost | - | ++ | +++ |
| Setup Time | --- | +++ | +++ |
| Speed | +++ | + | ?? |
| Reliability | ++ | - | + |
| Software | + | -- | -- |
| Extra connections | --- | +++ | +++ |
| Soldering work | --- | ++ | ++ |

Cables have as highest advantage, that these are reliable. However, all devices need to communicate with the Controller, so it will mean one cable per slave. And either two connections per slave to make some kind of network of many connections on the Controller. When multiple slaves of the same type should be connected, a network is the only possibility.

TODO: additional checks are needed:

* For cables:
  + What kind of protocol: SPI (already used by SRAM/SD) or I2C?
  + What connectors (Dsub?)
* For WIFI:
  + Needs for a router or can a local network be created?
  + Is speed high enough?
  + ESP8266 need to be programmed probably
* RF:
  + Check in live situations with lots of noise/other RF sources
  + Is speed high enough?
  + Free channel checking need to be implemented

# Aspects

## Network Topology

The following requirements are needed for the network:

* The Controller sends messages to each slave device.
* Each slave device sends messages to the Controller.
* Slave devices do not send messages to other slave devices.

## Slave Presence

Initially, the Controller needs to know which slave devices are present. Therefore, each device has its own predefined channel. TODO: how to define this / (auto) changing of channels in case of wireless communication.

During the initialization the following actions take place:

* The Controller polls for used devices/channels
* Each device returns its type (e.g. MIDI, Pedals/Switches etc)
* The Controller sends information to each device for what messages to send data back. This depends per slave.

## Loop

After initialization, again the Controller always takes initiative. The reason is that important messages (like from MIDI) should be handled before lower priority messages (like a message from a pedal/switch) and important messages should not be interfered by possibly colliding messages from less priority slave devices.

Therefore the Controller polls each device for messages to be received. Maybe high priority devices will be polled more often than less priority devices.

Assuming all devices have same priority, the following flow will be typical for two slave devices:

1. The controller asks slave 1 if there are messages (Controller is transmitter)
2. The slave sends messages back (including a last (empty) message) (Slave 1 is transmitter)
3. Assuming these are high priority messages, messages are sent back. It can happen that some messages are discarded (e.g. MIDI CC messages with the same CC value).  
   It should be prevented that switching the transmitter happens too often).
4. Next slave is being handled (note that high priority slaves might be polled more often).

# RF

## Proof of Concept

There are different types of RF, like below 1 GHz and above 1 GHz frequencies. Normally, 2.4 GHz is used. The default module to be used is NRF24L01+, however also the cheaper SE8R01 is available. Both will be used for the proof of concept.

* Which RF
* Used communication method: SPI
* Speed: Highest reliable / highest possible (cost more energy, but since an adapter will be used, this is not a problem). Default speeds are 250 mbps, 1 gbps, 2 gbps.
* Switching between receiving and transmitting.

## NRF24L01

These are the typical device used for 2.4 GHz wireless communication. Since the latter is cheaper and compatible, the latter is chosen. Also, it has a special network topology, using a 6-1 network for 6 transmitters and 1 receiver. The idea is to use this to:

* Let each slave be a transmitter
* The controller is the (only) receiver
* Each slave sends messages as soon as an input signal is received that has to be sent to the controller.
* After each message sent by a slave, the slave

## One or Two RFs per Device

After a short check it turned out that sending a message can take from 0,6 ms (16 byte payload) to several ms when retries are needed. Without retries sometimes messages are lost. However, several ms is too much to delay a message. Therefor an own protocol will be used.

Also to prevent switching (which takes 250 to 350 us excluding overhead) it is best to use 2 RF’s per device, one for sending. This means that devices can both transmit and receive without switching.

Also this makes it easier to send whenever needed, in case of collisions do a (smart) resend.

## Best Channel Initialization

The 2.4 GHz band used by the nRF24L01+ radios is from 2.4 GHz and has 125 possible channels. From these channels until channel with frequency 2.484 GHz is used by WIFI and therefore unusable by Mestra (also because WIFI will be heavily used by changing audience and thus very unpredictable).

The controller will check which frequency is best and send it to all slaves which can send a message back when received to move to the new frequency.

TODO: Check if the used frequency needs to be changed during operation.

This way the controller automatically knows which devices are present (the slave devices return their type).

## Normal Operation

Devices will send messages when available. This means it can happen that messages collide. If so, there is a default scheme how long each device waits until trying to retransmit (varying from 10 us to 100 us).

The time to send a 8 byte payload message cost (see datasheet, page 38, fragment below).

Since ACKs will not be used, the time will be T\_UL + 2 \* T\_stdby2a + T\_IRQ = PL / SPI\_data\_rate + 2 \* 0,130 us + 0,006 us =113 / 16 us + 2 \* 0,136 us = around 7 us. However when tested, the shortest time was around 136 us (so let’s assume that).

Where PL (payload length) = 8 \*( 1 + 3 (address) + 8 + 1 (CRC) )+ 9 (bits) = 113 bits.

The SPI rate is assumed 16Mbps

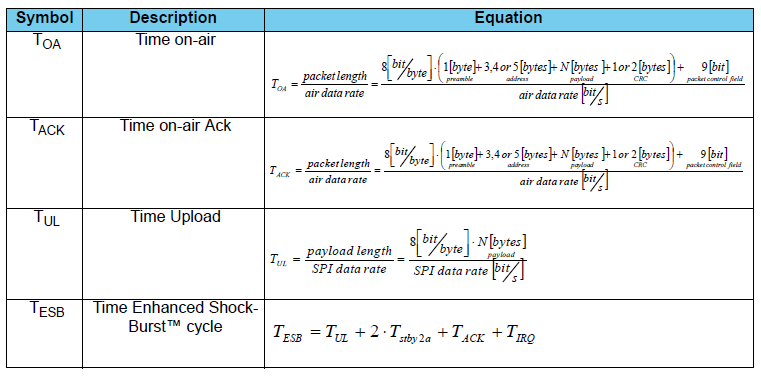


Figure 1: Time Equations

* This means per second 1000000 / 136 / 2 = 3676 messages can be sent. The division by 2 is because for each message an acknowledgement message is sent. Retries are not taken into account here.

## Acknowledgements

Every message has to be acknowledged, both from controller and from slaves. An acknowledgement command has two bytes: one Ack ID and the sequence number to be acknowledged. The other 6 bytes can be used for normal messages which otherwise had to be sent normally.

Acknowledgements need to be sent less than 0.5 ms after receiving the message, to prevent the sender sending an equal message, thinking it has not arrived.

## Reducing Messages

To reduce messages multiple commands can be stored in one payload.

Also messages to be sent are not sent within a 2 ms from one device. Messages are ‘saved’ until ready to be sent after 2 ms.

## Multiple Messages

All commands either have a fixed length or their length inside the command as first byte(s) after the command ID. This means the receiving device knows if a message is unfinished and has to wait for more. In this case an ACK is sent only after the last one (last message). This prevents collisions and reduces messages.

## Retries

It can happen that retries are need. For example, when the receiving device does not get a message, or when an acknowledgement message is not received back. 2 ms after sending a message and not getting an acknowledgement message, the message is sent again. An acknowledgement message does not need to be acknowledged, since the sender of the original message will send its message again after 2 ms, and continues to do so until the message acknowledge has been received.

Note that it takes about 0.5 ms to send a message, the acknowledge message will be sent by the receiver after a very short time (< 0.5 ms) and sending takes another 0.5 ms. So 2 ms should be adequate to be sure the message has not been received.

## Payload

This depends on the type of device. All messages are 8 bytes. In case bigger messages need to be sent, they are split up (e.g. MIDI system exclusive messages). The partial messages need to be stored until complete.

Device IDs do not need to be sent separately, but a message ID is needed for the acknowledgement. Therefore each first byte of the payload is the sequence number for that device message counter.

The other bytes are e.g. for a MIDI device, Note on command: 2nd byte: NoteOn, 3th byte: Note number, 4th byte: velocity.

Multiple MIDI messages can be packed in one payload.

## Alive Message

If the controller has not been sending a message to a slave for 5 seconds, it will send an alive message. If there is no reaction, both the controller and slave will notify it with a LED.

## Diagnostic LEDs

All devices will have a LED to show the power. The reason is that almost all boxes will not have any direct notification if it has power or not.

Also, each box communicates through RF, so for this reason a LED will show the RF status. Note that when the controller polls each device, this will not be shown (since it happens every few ms), and slave devices receiving a poll request for packages and no messages are available (thus an ‘empty’ ACK package is sent, will also not result in a LED being on at the slave side, since this will happen many times per second.

Table 3: Generic Diagnostics LEDs

|  |  |  |
| --- | --- | --- |
| **Function** | **LED Color** | **Description** |
| Power | Blue (generic) | Off: Power off  On: Power on |
| RF | Yellow (generic) | Off: empty message transmitting/receiving  Slow blinking: contact with controller  Double fast blinking per second: no contact with other device(s).  Triple fast blinking per second: problem with RF  On: non empty message transmitting/receiving |

Note that if the GUI Device shows errors whenever possible.

Blinking details:

* Slow blinking means one blink of 20 ms, followed by a pause of 980 ms (totaling 1 s).
* Double fast blinking means two blinks of 20 ms with a 80 ms gap, followed by a pause of 800 ms (totaling 1 s).
* Triple fast blinking means three blinks of 20 ms with a 80 ms gap, followed by a pause of 700 ms (totaling 1 s).