RadioControl-Protocol

Rev.1.3

Paul Nykiel

March 29, 2019

Contents

1 Abstract

Fast and extensible communication over various serial-interfaces. Primarily designed for radio-controlled models, mesh capable.
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.1 Example Application

1.1.1 Trainer

Transmitter → Transmitter → Plane

1.1.2 Long Range

Long Range: Multiple Transmitters, Drone as Repeater. Network for Transmitters.

2 Protocol

The protocol is designed for serial interfaces, the data is binary encoded to optimize for slow connections. The protocol consists of packages, one package contains all channel data.

2.0.1 Package

A package is a unit in which all information at a certain time point is collected. The protocol is responsible for encoding and decoding the data into a package. Furthermore the package is responsible for error-detection (not correction), it is not responsible for retransmitting data or the order between packages.

2.0.2 Channel

The data of a package is accumulated in channels. Every package consists of a fixed number of channels with every channel having the same (fixed) resolution.

2.1 Timing

A new package should be transmitted by the sender at a rate of 50Hz (every 20ms) which is the rate a normal Servo or ESC gets signals at. The minimum time between each package must be large enough to allow every device to parse the data.

2.2 Mesh behavior

The sender of a message is the one deciding wheter a package should be mesh package or a normal package. A normal package gets transmitted by the sender and every device in which is reachable via a direct link receives the message and can use the data.
In a mesh application the sender must set the routing length flag to a non zero value. Furthermore the sender must specify via the routing length field the maximum amount of nodes the package passes by. Each recipient should resend the message over every interface but the receiving one and decreases the routing length value by one until the routing length reaches the value zero. To avoid that the same packages passes multiple nodes the same time every node should keep track of the packages it forwarded, this is done by having a sufficient sized ring buffer which saves the last UIDs of the packages forwarded, the size is determined by the amount of packages per time, the may be between 64 and 128.

2.2.1 Mesh initialization

If necessary a node can perform a recursive discover to find out which device can be reached via an interface with a maximum amount of hops (routing length). This method is not intended for networks where multiple paths can reach the same targets.
The node sends a discover message via all connected interfaces. The message gets forwarded to all nodes in the network with consideration of the routing length. When a device with only one interface is reached or the routing length field is zero the node sends a discover-response with its transmitter-ID as payload via the receiving interface. Every node that forwarded the discover message waits for a discover-response from every interface and then sends a discover response containing all the transmitter-IDs it received and its own ID via the interface that was used for the discover-message. In the end the initial node receives a package with the ID of all transmitters that can be reached via a certain interface in a certain maximum amount of hops.

Discover Message

A mesh discover message consists only of the header, the configuration and the footer, there is no channel data. A discover message is identified by the discover-message bit in the mesh-configuration byte.

Discover Response

A mesh discover-response is a normal package with the channel resolution set to 8-bits (256 Steps). The channel count is as small as possible. Every channel value that is non zero represents a transmitter-ID.

2.3 Package structure

Every package consists of four major parts:

1. Header
2. Configuration
3. Data
4. Footer

2.3.1 Header

The header consists of three bytes:

1. Start byte
2. Unique-ID
3. Transmitter-ID

Start byte

The start byte is one byte which is always $C9_{\mathrm{HEX}}$.

Unique-ID

The unique-ID is generated by the initial sender. It is used to identify a message, especially in a mesh-application.

Algorithm for generation of the UID The easiest way of generation is having a counter shared by all devices, every time a device receives or transmits a package the counter gets incremented. If the upper limit of a byte (255) is reached the counter overflows. This implementation guarantees maximum time between reoccurring UIDs.

Transmitter-ID

A persistent ID which should be unique to the transmitter. Optimally it should be easily configurable by the user, for example via a configuration file, a settings menu or DIP-Switches on the module. The transmitter-ID zero is reserved for mesh discovery.

2.3.2 Configuration

The configuration consists of at least one byte. The 7-th bit of every configuration byte is used to keep track of how many configuration bytes there are.

First configuration byte (general)

The first byte consists of the following bits:

- Bit 0-2: Resolution
- Bit 3-5: Channel count
- Bit 6: Error
- Bit 7: Following

Resolution

Three bytes containing the resolution of each channel transmitted, possible values are listened in table 2.1.

Value	Resolution (Steps)	Bit per Channel
000_2	32	5
001_2	64	6
010_2	128	7
011_2	256	8
100_2	512	9
101_2	1024	10
110_2	2048	11
111_2	4096	12

Figure 2.1: Possible resolution values

Channel count

Three bytes containing the amount of channels used, possible values are listed in table 2.2.

Value	Channels
000_2	1
001_2	2
010_2	4
011_2	8
100_2	16
101_2	32
110_2	64
111_2	256

Figure 2.2: Possible channel count values

Error

If a device registers an error (checksum, timeout during package transmission) this flag is set for the next 4 packages send by this device, if another error occurs before all 4 packages are sent the number of packages with the error flag set gets increased by 4.

Following

This byte is a flag whether the following byte is still a configuration byte and not a data byte.

Second configuration byte (mesh)

The second byte consists of the following bits:

- Bit 0-3: Routing Length
- Bit 4: Discover Message
- Bit 5: Discover Response
- Bit 6: unused
- Bit 7: Following

Routing Length

A number counting the amounts of nodes this message has been passed by. The initial sender sets the value and every node decrements the value. If the value reaches zero, the package doesn't get forwarded.
The value zero means that this package is not a mesh package.

Discover Message

This flag denotes that the current message is a discover message. This implies that there is no channel data.

Discover Response

This flag denotes that the current message is a discover response. This implies that each channel consists of 8-bits.

Following

This byte is a flag whether the following byte is still a configuration byte and not a data byte. At the moment this byte needs to be false.

2.3.3 Data

The size (in bytes) of the data section is determined by the resolution and the channel count:

$$\text{size} = \left\lceil \frac{\text{resolution} \cdot \text{Channel count}}{8} \right\rceil \tag{2.1}$$

The data is just queued together so the first r (with r being the channel resolution) bits are the first channel, bits r+1 to r*2 are the second until the last channel. If there are unused bits left at the end they shall be zero.

2.3.4 Footer

The footer consists of two bytes Checksum End byte

Checksum

For the checksum think of the transmitted data as list starting with the unique-ID and ending with the last data byte. The start-byte, end-byte and checksum are not used for the calculation. The checksum is once calculated by the sender and once by the recipient to verify the data is still correct. A implementation may still accept a package even tho the checksum is incorrect.

Algorithm

The checksum is similar to a normal parity bit. To calculate the checksum take the list of bytes and count for every bit position how often it is set across all bytes, if it is odd the corresponding bit in the checksum is 1 else 0. So the n-th bit of your checksum byte is the parity bit of all n-th bits of the byte-list.

End byte

The end byte is one byte which is always 93_{HEX}.

3 Revision-History

- Rev.1.1 Removed Mesh-Enabled Flag and replaced with Routing Length. Clarified forwarding.
- Rev.1.2 Added Discover-Message and Discover-Response
- Rev.1.3 Clarification for error count and checksum (#14 and #15), added reference to RFC 2119