Purdue University

RF-Draw Network Specification

Anthony Goeckner and Krutarth Rao
CS489 - Embedded Systems

December 7, 2017

Contents

1 Overview

The purpose of RF-Draw's network stack is to provide easy, efficient communication between devices, either one-to-one or one-to-many. It accomplishes these goals by using a very minimal but robust network protocol, along with an automatic code generation system to allow for easy transmission and reception of commands from user code.

The protocol is command-based, meaning that all messages sent must match the format of a pre-defined command. This allows for simple, dynamic parsing of messages.

1.1 Network Topology & Routing

The network uses an ad-hoc mesh topology, with no central coordinator. This allows devices to be added or removed at will and eliminates the need for a central "hub" device. While the XBee proprietary protocol is capable of complex mesh routing for use when at least two nodes in the network are not connected to each other, it is expected that all nodes will be operating in the same room and that every node will be connected to every other.

2 Link Protocol

The link protocol facilitates communication between nodes in the network, both one-to-one and one-to-many. RF-Draw uses the proprietary link protocol supported by Digi XBee products. The radios must operate with their 'AP' parameter set to '1'.

2.1 Addressing

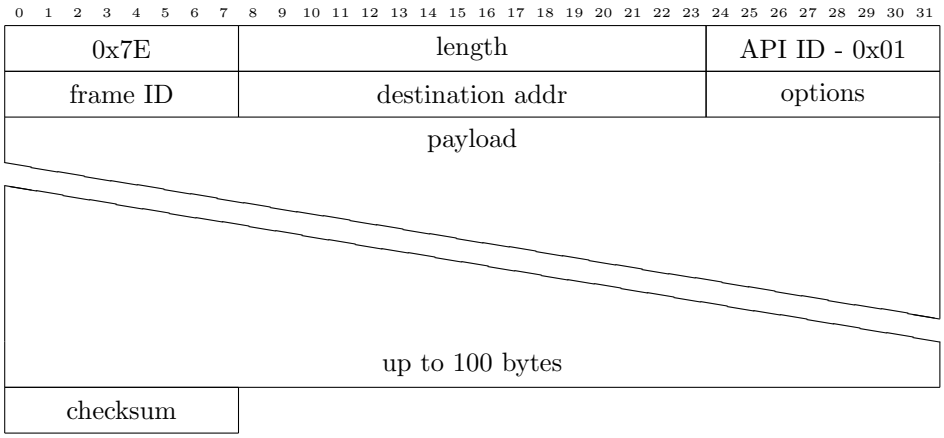
Frames can be sent with either a unicast or a broadcast address. Before operation, each node must be assigned a unique 16-bit address in the range [0x0000, 0xFFFE) using the 'MY' parameter. Unicast addresses must be less than 0xFFFE, and the broadcast address is 0xFFFF. Frame acknowledgement and retransmission is only supported in unicast mode.

2.2 Frame Retransmission

When a unicast frame is sent, the sender begins a timer. If this timer expires before the receiver sends an acknowledgement of frame reception, the frame is sent again. This occurs up to three times by default, or until the frame and acknowledgement are successfully received.

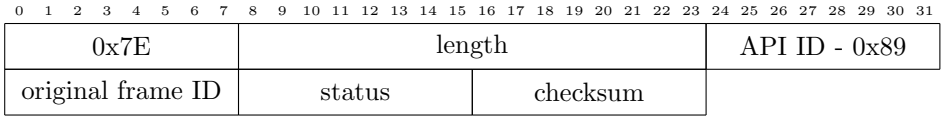
2.3 Tx Data Frame

A frame of the following structure is sent via UART to the XBee radio to transmit a message.



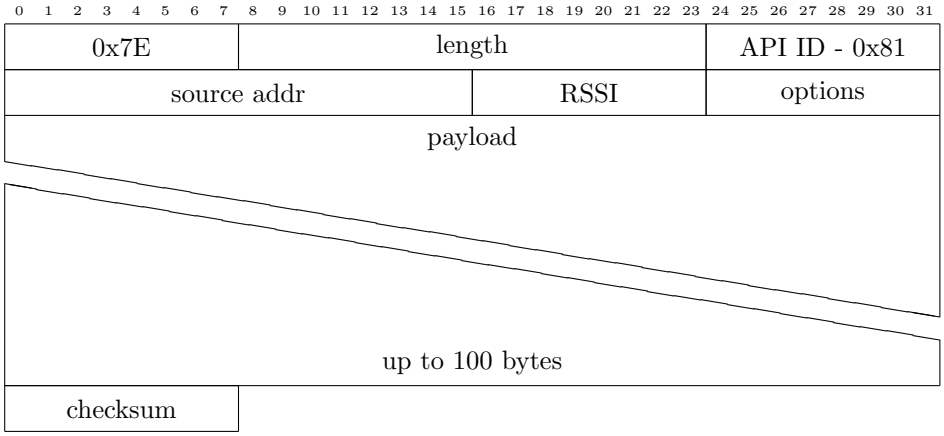
2.4 Tx Status Frame

When a Tx Data frame has been sent to the radio for transmission, the transmitting radio will respond via UART with the following status frame.



2.5 Rx Data Frame

When a radio receives a transmission from another radio, it will send the following frame via UART.



2.6 Description of Fields

All fields listed in the above frames are detailed in XBee/XBeeP-RORFModules802.15.4v1.xEx[2009.09.23].

3 Application Protocol

RF-Draw uses a custom protocol called PLink (Purdue Link), which is based on the MAVLink protocol. This is a command-based protocol, meaning that each message must fit its data to that of a predefined command.

3.1 Node Discovery & Removal

Upon power-on, nodes will alert other similarly configured nodes of their presence via a broadcast message. When such a message is received, each other node will add the new node to its list of known devices and then send a response to the new node, notifying the new node of itself.

After the initial discovery broadcast, the node will broadcast a heartbeat message every 5 seconds.

When a node is cleanly powered down, it will first send a message to all other nodes to immediately alert them to its removal. However, if a node is powered-off abruptly or if some other interruption occurs, all other nodes shall remove the node from their list of devices after 10 seconds pass without receiving a heartbeat message from the dead node.

3.2 Packet Retransmission

Each packet contains a checksum of the header and payload. If upon reception the CRC fails to match, the receiver will send a negative acknowledgement containing the sequence number of the last packet successfully received. If the CRC matches, the receiver will respond with a positive acknowledgement.

Upon transmitting a packet, the sender will begin a timer. If the timer expires before an acknowledgement (negative or positive) is received, the sender will retransmit the dropped packet, along with all packets sent after it.

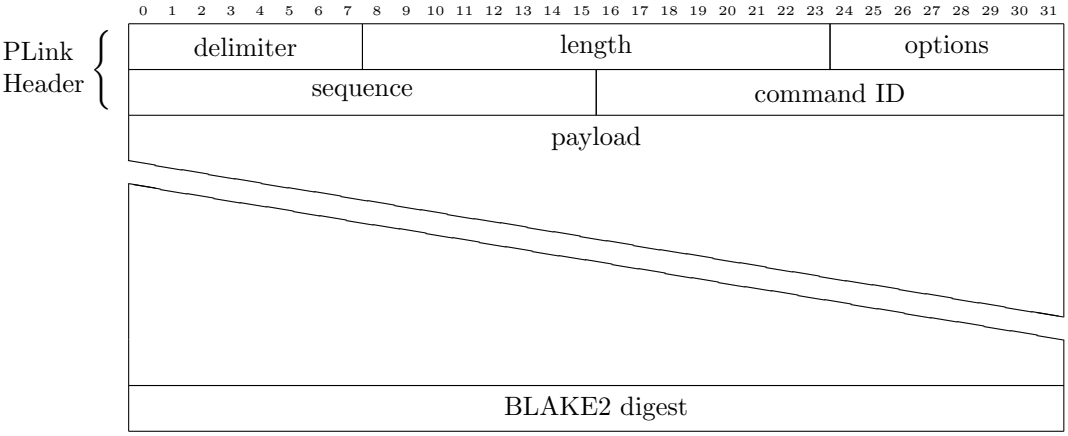
If a packet is received out-of-order or with a duplicate sequence number, the packet will be dropped.

Reception of a negative acknowledgement will cause the device to retransmit all packets sent since the last packet that was successfully acknowledged.

Reception of a positive acknowledgement will remove the packet from the potential retransmission list if all previous packets have been successfully acknowledged.

3.3 Packet Structure

The overall format of the packet is as follows:



Header Fields

The following is a brief description of each header field.

Field	Description
delimiter	This must be set to 0xEB, and marks the start of a packet.
length	The length of the packet, including the header, payload, and CRC.
options	A field for future option bits.
sequence	The packet sequence number.
command ID	The length of the packet, including the header, payload, and CRC.
payload	Arbitrary data corresponding to the specified command.
crc	The checksum.

Error Checking

The specific CRC or other checksum algorithm has yet to be determined as of November 11, 2017.

4 Programming Interface (API)

Before usage, commands must be defined in a "command definition file". The definition file is then passed to a code generator, which creates auto-generated code for easy command transmission, reception, and parsing.

4.1 Command Definition

Commands are defined in an XML file using the following format:

"Command Definition File"

```
<?xml version='1.0'?>
<mavlink>
    <version>1</version>
    <devices>
        <device name="typeA" lang="python">
            <description>A example device type.</description>
        </device>
        <device name="typeB" lang="c">
            <description>A different class of device.</
                description>
        </device>
        ...
    </devices>
    <enums>
        <enum name="example_enum">
            <entry value="0" name="example_value">
                <description>This is one of the enum's possible
                    values.</description>
            </entry>
            ...
        </enum>
        ...
    </enums>
    <messages>
        <message id="0" name="EXAMPLE_COMMAND">
            <description>This is a command.</description>
            <field type="uint8_t" name="some_var">This is an 8-
                bit unsigned integer.</field>
            <field type="double" name="another">This is a double
                </field>
            ...
        </message>
        ...
    </messages>
</mavlink>
```

The above example demonstrates most functions of the definition file, including device definition, which allows for multiple classes of devices to be defined. Each class of device can require code generated in either C or Python, and commands can specify that they are only

to be sent or received on certain classes of devices (for example, a display device need not transmit data). This also shows enumeration definition, which allows for the same enumerations to easily be shared between devices, along with a command definition that has several parameters.

4.2 Code Generation

The command definition file is passed to the code generator, which uses it to generate C or Python code for each class of device. These generated code files can then be imported to the RF-Draw program itself.

The code generator creates three basic types of functions: SEND_⟨command-name⟩, PARSE_⟨command-name⟩, and RECV_⟨command-name⟩.

The SEND_* commands generated for each function are public and accept as arguments all of the fields listed in the definition file. The command takes in each field as an individual argument, packs them into the network format, and places the completed packet on the transmit queue.

A PARSE_* function is also created for each command. This is used to break down the payload of incoming packets into individual fields. It then calls the RECV_* function, passing all fields as individual arguments.

The RECV_* functions are used to perform actions when a specific command has been received. While these functions are *called* by the auto-generated PARSE_* functions, they are not defined automatically. Instead, they must be defined by the user. For each command that a device can receive, that device must have a function called "RECV_⟨command_name⟩", which takes each command field as an argument (of the same name), and returns null.