Introduction

This document describes the communication protocol between Marlin and a host system. The protocol has evolved over several iterations:

- v1 Is the protocol as used by many RepRaps, the Ultimaker Original and Ultimaker2 printers. This protocol is fully ASCII and allows people to enter commands on a terminal. For computer based communications there is an optional CRC.
- The v2 protocol was created because v1 was not robust enough in the situation of communication errors and re-sends. This version is not backwards compatible as it introduces binary data like a sequence number and checksum which make it impossible for a user to enter commands directly.
- V3 changes the response commands from Marlin to the host, these are made more compact.
- V3.1 added the ‘q’ info response for reduced CPU load on the host (no more status polling required).

The communication protocol is not mirrored, meaning that the host to slave communication is not the same protocol as the slave to host communication. This is because the slave is limited in memory and CPU and doesn't have the facilities to keep buffers for re-sending. Also, the current slave implementation has no concept of a "message" towards the host but sends data in separate pieces. Changing this would be a serious effort due to the errors being sent asynchronously and thus can be in-between of a normal message.

Specification(s)

Host to Slave

v2 introduced a packet format (see SerialProtocol.h) for sending commands/data from host to Marlin. This packet contains defined sections for protocol management, data, and validation:

Start (index)	Description	Size (bytes)	Remarks
0	Start of Packet	2	Always the value 0xFFFF which should also fix framing errors. Unusual is to use 2 SOP bytes. That's to match the 2 CRC bytes, the only fields that are also allowed to contain the 0xFF value. Other protocols use an escape token which reduces the communication overhead a bit.
2	Sequence Number	1	Round-Robin value 0 .. 254 to be used for error detection and (resend) recovery. A size of 255 is disallowed as this is the same value as the Start of Packet
3	Payload Length	1	This will be >= 1 and <= MAX_CMD_SIZE (defined in Marlin as 96). This defines how many bytes there will be in the payload sequence.
4	Payload	1 .. MAX_CMD_SIZE	The actual data which may only contain ASCII values, defined as >= 0x20 and < 0x80.
4 + Payload Length	CRC-16	2	A CRC calculated over the fields "Sequence Number", "Payload Length" and "Payload" to detect errors. For error correction the whole packet will have to be resent.

Choosing a CRC algorithm

A lot of technical jumble can be thrown in here but it comes down to confidence levels based on the expected number of introduced errors.

A CRC8 check can detect single bit errors in a stream up to 511 bits. With a maximum message length of 784 bits (sequence nr + payload length + 96 bytes message) it was deemed that a CRC8 would not be safe and a CRC16 was chosen. The chosen polynomial for the CRC16 is the CCITT 0x1021 variant.

Hindsight info: The CRC16 is overkill, there are almost no communication errors anymore. The early UM3 models had a lot of communication errors in the communication to Marlin caused by a wrong configuration of the serial port driver on the Olimex board. The IO pins didn't get a power supply and the serial port was only working by some leak currents.

Slave to Host

The Marlin responses are separated into 3 different data sets:

- framing packets which are used to give feedback to the Host about the packets being received
- out of band packets which are used to indicate some kind of situation Marlin wants to report back (not linked to a packet from the Host)
- response data which is actual data being sent back as the result of a package sent by the Host

Type	V2	V3	Description	Remarks
Framing packet	"\n\rok N[0-254] P[0-16] *CRC16\n\r"	"\noNNPPCC\n"	Message received and handled with sequence number **NN** (0-254) as hexadecimal representation, and **PP** (0-16) as hexadecimal representation for the the remaining room in the plan buffer. Last part **CC** is the CRC8 over the ***oNNPP*** data.	"Ok"
Framing packet	"\n\rrej N[0-254] *CRC16\n\r"	"\nrNNPPCC\n"	Message with sequence number **NN** (0-254) as hexadecimal representation is rejected, and **PP** (0-16) as hexadecimal representation for the the remaining room in the plan buffer. Last part **CC** is the CRC8 over the ***rNNPP*** data. The reason for rejection is sent prior to this message with the out of band "ProtoErr" message.	"Resend"
Framing packet		"\nqNNPPCC\n"	This message is sent by Marlin autonomously to prevent polling from Opinicus, it is not a response to any request. This polling causes way too much CPU usage, it was introduced in firmware 5.1.x (after 5.0.19) The sequence number is always 0.	"info"
Out of band	"ProtoErr: [ascii]\n\r"	"ProtoErr: [ascii]\n"	Actual error message to indicate the type of protocol error, for debugging.	For consistency sake, this should become PROTOCOL_ERROR
Out of band	"Error: [ascii]\n\r"	"Error:[ascii]\n"	System error.	For consistency sake, this should become ERROR
Out of band	"WARNING: [ascii]\n\r"	"WARNING: [ascii]\n"	System warning.	
Out of band	"LOG: [ascii]\n\r"	"LOG: [ascii]\n"	Log the message in the system log.	

Response data	Anything else (ASCII), including \n\r	No more "\r" being sent, rest remains the same.	Any byte that is sent to the serial output, not being part of the previous packet types is to be considered data to report back to Opinicus. This data will be handled by Opinicus whenever a Framing packet is received.	

With Jira | EM-2346 this reply will change into a more compressed V3 format:

Packet #	Start	Description	Size (bytes)	Remarks
1	0	\n	1	Start of the response data - makes sure it begins on a new line. Might be changed with an STX in the future(?)
2	0	Framing command	1	This will be either an 'o' (lowercase) to indicate a previous sent Opinicus packet is handled (note: handled is not the same as received correctly) or an 'r' for reject indicating the packet received is invalid. The next 2 bytes will identify the Opinicus packet.
2	1	Opinicus packet number	2	This will be the hexadecimal representation of the packet number of Opinicus (0-254).
2	3	Plan buffer size	2	This will be the hexadecimal representation of the remaining plan buffer size (0-16). This will now also be returned with the reject response!
2	5	CRC8	2	Instead of a 4 bytes CRC16 this is now a 2 byte CRC8, again in hexadecimal format.
2	7	\n	1	End of Packet indicator. Might be changed into an ETX in the future(?)

The response is made out of 2 packets, basically separating the previous data (which can be response data) and the actual framing packet so it can be easier recognized on the Opinicus side.

Implementation

Host to Slave

The current implementation of the Slave (Marlin) side to handle communications is as follows:

1. Wait for the first header byte (0xFF) which indicates a start of a packet.
2. Wait for the second header byte. If this is not a valid byte (0xFF) then we wait for the first again
3. When a header byte (0xFF) byte is received when it's not expected (waiting for the sequence number, payload length or payload) this byte will be considered to be the start of a new packet and the protocol will wait for the second header byte.
4. The next number will be the sequence number. If this is not equal to the number being expected, a "Resend" will be communicated to the Host (Opinicus), and waiting for the Start of Packet will be in order.
5. Next, the payload length is expected. If this is received and higher than MAX_CMD_SIZE, a "Resend" will be communicated to the Host, and waiting for the start of the packet will be in order.
6. While the payload length is not yet matched, verify the character to be in bounds. If it's not, a "Resend" will be communicated to the Host, and waiting for the start of the packet will be in order.
7. Next 2 bytes for the CRC will be received and compared to the CRC16 calculated over the Sequence Number, Payload Length, and Payload. If the values are not equal, a "Resend" will be communicated to the Host, and waiting for the start of the packet will be in order.
8. The packet will be handled for further processing. If the processing fails, a "Resend" will be communicated to the Host, and waiting for the start of a packet will be in order.

In all cases, before a resend, the Slave will communicate back to the Host the reason for the "Resend" request.

A request to Resend will be sent every time it's needed.

All "Ok" and "Resend" framing packets will be coming on a new line and are appended with a CRC8.

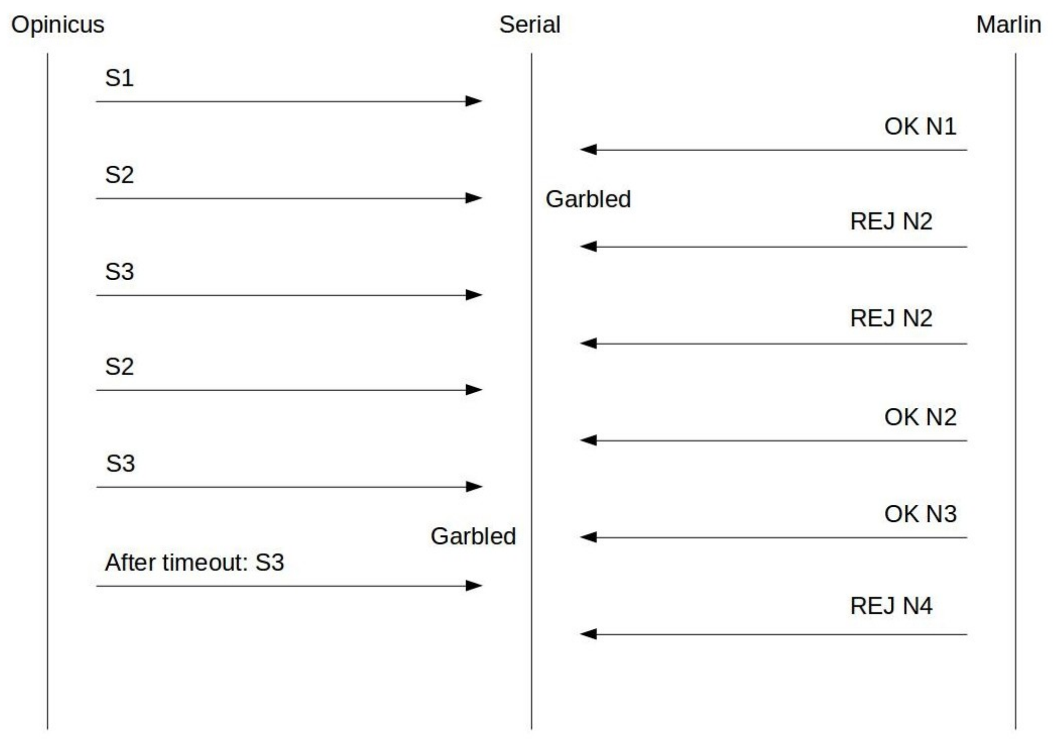
Slave to Host

The current implementation of the Host (Opinicus) side to handle communication is as follows:

When receiving an ACK or REJ, the CRC is verified. If the line is deemed to be corrupted, it will be ignored.

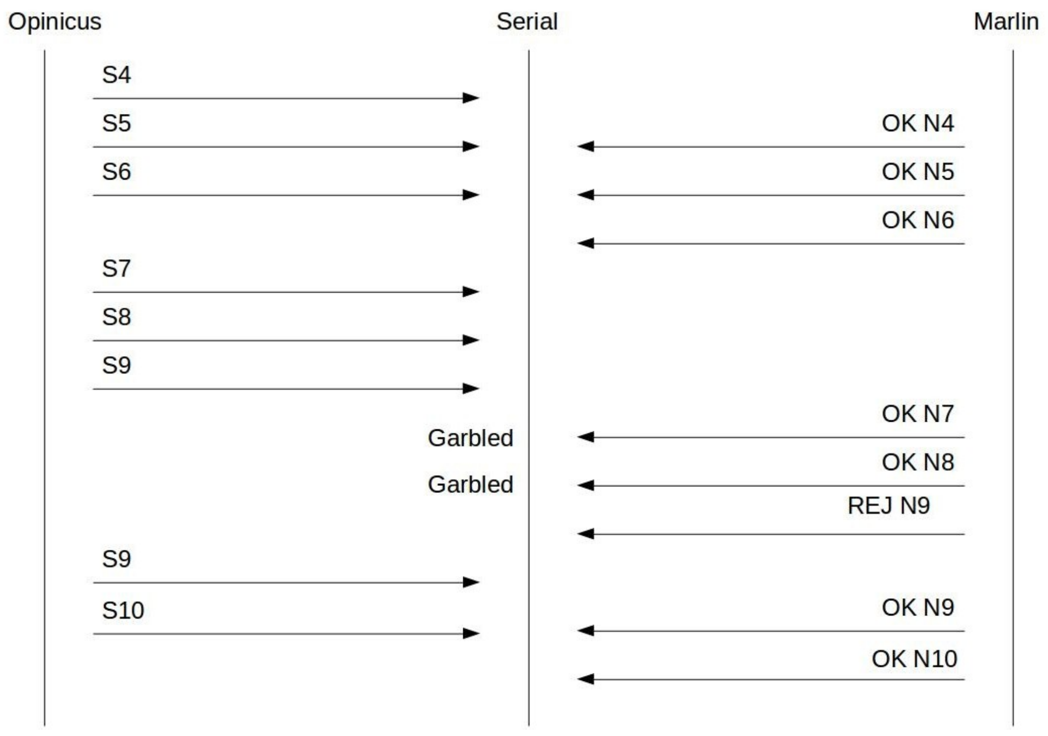
Network Protocol Diagram

Diagram 1: simple example(s)



1. Each sent packet by Opinicus must be met with either an OK or a REJ.
    a. In case of an OK it will contain the sequence number of the last packet accepted
    b. In case of a REJ, it will contain the sequence number of the packet being rejected
2. A REJ will be returned when:
    a. the received sequence number is not the expected one
    b. the payload size exceeds the maximum (MAX_CMD_LINE)
    c. the payload contains invalid data
    d. the calculated CRC over the data received does not match the received CRC
3. If a packet cannot be matched with an OK or REJ, it will be resent after a time out.

Diagram 2: slightly more complex example(s)



1. A (semi) optimal buffer for 3 packets is used. On the Marlin side, 3 is the maximum number of packets that can be received as a whole.
2. Packets will be implicitly marked as sent and received if its sequence number is lower than the sequence number received by either the OK or REJ.

Possible improvements

These suggestions help to improve the protocol, but situation 2020-10 is that the biggest problems are in different areas. There are huge delays in messages (30ms+) which pose a way larger limit on the throughput than small protocol optimizations.

1. Split the command receiver in Marlin in a low level communication handler and a high level command handler. This makes it possible to handle communication errors separately from how full the command queue is. Packet re-sends can be handled earlier, causing less stalling. For example add 1 response (a for Acknowledge) to indicate a packet has been received, but not handled yet. This will likely improve the communications. The o (Ok) response will then indicate the packet has been executed.
2. Instead of \n use the normal STX/ETX communication protocol control characters. (not sure if and how this is an improvement)
3. Handle data and response data differently on the Opinicus side. Everything sent now from Marlin is read as one big array of lines. And parsed. Keeping track of data being returned and put in a different buffer as the responses.
4. Compact the remaining responses: ProtoErr → p, Error → e, Warning → w, LOG → l. Small gain; it will make parsing on the host easier.
5. Replace the CRC16 in the host to Marlin messages by CRC8. Then the 2 Start of Header bytes can be reduced to 1 byte too. CRC16 is overkill with the number of communication errors we see. Gain: 6% less communication on average of 30 bytes payload.
6. Change the gcode commands where the distances are now in mm into steps, the conversion of steps/mm is then executed on the host. Communication volume remains the same. Gain: Marlin CPU performance can be improved since many floats can be replaced by integers.
7. Replace the 0xFF start indicator by another value. This is about the worst possible sync character, it equals an open line.
8. Use 7-bit data instead of 8 bit for reduced communication time (-12%), possible because we only communicate

ASCII in the range 0x20 - 0x7F, except for the 0xFF start character. Good for the transmit from Marlin, not sure if Marlin's receive can cope with the higher throughput.