Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 8, 2020

C. Bormann
Universitaet Bremen TZI
T. Kaupat
Lobaro UG

November 05, 2019

Slipmux: Using an UART interface for diagnostics, configuration, and packet transfer
draft-bormann-t2trg-slipmux-03

Abstract

Many research and maker platforms for Internet of Things experimentation offer a serial interface. This is often used for programming, diagnostic output, as well as a crude command interface ("AT interface"). Alternatively, it is often used with SLIP (RFC1055) to transfer IP packets only.

The present report describes how to use a single serial interface for diagnostics, configuration commands and state readback, as well as packet transfer.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 8, 2020.

Table of Contents

1. Introduction

Many research and maker hardware modules for Internet of Things experimentation ("platforms") offer a serial ("UART") interface. This is often used for programming, diagnostic output, as well as a crude command interface ("AT interface"). Alternatively, it is used with SLIP (RFC1055) to transfer IP packets only; this may require the use of another serial interface for diagnostics and configuration.

The present report describes how to use a single serial interface for diagnostics, configuration commands and state readback, as well as packet transfer.

1.1. Terminology

The term "byte" is used in its now customary sense as a synonym for "octet". Where bit arithmetic is explained, this document uses the notation familiar from the programming language C (including C++14's

0bnnn binary literals), except that the operator "**" stands for exponentiation.

2. Using a UART interface

The serial interfaces provided by today's platforms often do not actually use EIA-232 ("RS-232") levels, but some form of logic levels (TTL or more likely 3.3 V CMOS). The present report does not discuss physical interfacing, but assumes that a TXD (transmit data) pin, a RXD (receive data) pin, and a GND (common ground) pin are all that is available. To interface laptops and similar devices to these serial interfaces, inexpensive ($2) USB to UART adapters based on chips such as PL2303, CP2102 or CH340 are easily obtainable. (The usual care needs to be taken when mixing 3.3 V and 5 V platforms; this is well understood but beyond the scope of the present report.)

The general assumption is that a serial port configuration of 8N1 (8 bits per character, no parity, 1 stop bit) and a bit rate of 115200 bit/s is used. As with the logic levels, alternative arrangements are possible, but a 3.3 V CMOS, 115200 bit/s interface is most likely to provide the best interoperability.

While it would be possible to run relatively complex and versatile protocols such as PPP [RFC1661] on such serial interfaces, this goes against a need for simplicity and ease of setup. In today's systems, either weird ad-hoc protocols based on "AT commands" are used that are not interoperable at all, or a simple encapsulation such as SLIP [RFC1055] is used for packet transfer only.

For the purposes of the present report, on top of the serial UART protocol, the frame format defined by [RFC1055] is indeed employed. The detailed descriptions below generally describe the frame data before applying SLIP escaping in the transmitter or after removing it in the receiver.

The approach described here is informally referred to as "slipmux".

3. Packet Transfer

Packet transfer uses the definitions of [RFC1055]. However, contrary to the statement in section DEFICIENCIES of [RFC1055], multiplexing is very well possible. A frame used for packet transfer is detected by an initial byte of one of the two forms:

- 0x45 to 0x4f: IPv4 packet

- 0x60 to 0x6f: IPv6 packet

This initial byte forms part of the packet; it is not removed from the payload as with the other formats defined below.

There are no changes to the formats defined by [RFC1055], so there should be immediate interoperability with tools such as tunslip.

4. Diagnostics Transfer

While not transferring a frame bearing a packet, the platform can alternatively transmit a diagnostic frame. These are encoded (and escaped) using SLIP framing exactly like packet frames, but start with the byte 0x0a (ASCII newline) and contain UTF-8 encoded characters after that byte. There is no semantics attached to the diagnostics message, except that it is intended as a human-readable debug or diagnostic message from the platform code. It is generally preferable to end the payload of a diagnostics message in another newline (0x0a, which on the wire is then followed by 0xc0 due to the SLIP framing). Note that, as long as only ASCII characters are used, there is no need to actually perform escaping on the diagnostic message.

Since diagnostic messages are intended for humans, they are only defined for the direction from platform to host (e.g., laptop); for robustness when connecting two platform modules, they should be ignored by platform modules.

5. Configuration

Configuration is performed by sending CoAP messages [RFC7252] in SLIP framing. The encapsulation of a CoAP message starts with an additional byte 0xA9, with the bytes of the CoAP message following (which, as for all data in frames, are escaped as necessary as per [RFC1055]).

In contrast to the packet and diagnostics frames defined above, CoAP frames benefit from a frame check mechanism. After the CoAP message, the last two bytes of a CoAP frame therefore contain a 16-bit CRC FCS computed over the byte 0xA9 followed by the (unescaped) bytes of the CoAP message, computed as specified in [RFC1662]. (Note that the two bytes of the CRC are escaped, as necessary, by the SLIP framing, as are all other bytes of the CoAP message.)

CoAP messages with incorrect CRCs are silently discarded.

Where a local URI needs to be formed for the configuration messages, the URI scheme "coap+uart" is used; the authority part of that URI might be used to refer to local interface names as needed, as in:

coap+uart://ttyUSB0/APlist

The client could also be using a local mapping table to provide some indirection in translating the authority part to a local identifier of the serial port(e.g., COM0 to ttyUSB0).

Using an empty URI authority allows the client to use a default port, as in:

coap+uart:///APlist

A later version of this report might define some common CoAP resources that research or maker platforms might want to provide, e.g. to cover the configuration and status checking often done by "AT commands" today.

6. Framing considerations

To make SLIP framing robust, it is important to send SLIP frame delimiters (0xc0) before and after each SLIP frame (maybe unless frames follow each other back to back). This means that empty frames need to be silently ignored by a receiver.

If a platform starts to send a packet or message, but then decides it should not complete the message before having sent the rest of the frame, it can send the SLIP ESC (0xdb) followed by SLIP END (0xc0) to abort the frame. Note that this goes beyond the error handling suggested by the section "SLIP DRIVERS" in [RFC1055] and might therefore be of limited interoperability at first.

Messages in slipmux are strictly sequential; there is no [RFC2687] style suspension. In particular, this means that diagnostic messages that are generated while another message is in progress may have to be buffered (unless they are important enough to abort the frame as described above).

7. Discussion

7.1. Why no shell?

The present report is somewhat radical in that it does not provide a common staple of interactive computer access: A command line interface (CLI), or "shell".

This would be easy to add, but distracts from the use of the platform as a "thing" - it should not have to carry an (even primitive) user interface; instead it should provide what would have been "shell commands" as CoAP resources.

As a transition aid, existing shell commands can first be converted to just accept their parameters via CoAP but continue to provide their output as ASCII text over the diagnostic channel.

However, in order to aid script-driven use of the platform, the next step should then be to also provide the response to the command in a CoAP response, possibly structured for better use by the script. Often code that was designed to format the data for human consumption can be simplified to just ship the raw data, e.g. in a CBOR data item [RFC7049].

7.2. Frame aborts

Implementing frame aborts as described in Section 6 requires a receiver to receive the entire frame before acting on it. For diagnostic information, this is somewhat moot - the information is there independent of whether its frame was aborted or not. For packets, it is usually necessary to check a UDP or TCP header checksum before acting on it, anyway. For CoAP requests, similarly, the CRC needs to be checked. So implementing frame aborts should not be an undue burden.

7.3. Unknown initial bytes

Frames with unknown initial bytes should be silently ignored.

The same is true for frames with initial bytes that are unimplemented. However, there is an expectation that true slipmux implementations do implement CoAP framing. If this is unexpectedly not the case, as a courtesy to a peer CoAP client, a slipmux implementation could at least send CoAP Reset messages: a CoAP frame (initial byte 0xA9) with a message that starts with 0x40 to 0x5f could be replied to with a CoAP frame with a CoAP RST message, containing just these four bytes (as always, escaped as needed, and framed with an initial 0xA9 and a CRC):

- 0x70
- 0x00
- The third (unescaped) byte of the message being replied to
- The fourth (unescaped) byte of the message being replied to

(Generating proper CoAP framing in response does, require implementing the PPP CRC.) In conjunction with the CoAP ping response of a normal CoAP implementation, this also can be used for liveness testing.

(The check for the first byte of the CoAP message is needed to avoid endless back and forth of reset messages in certain error situations.)

8. IANA Considerations

The present report does not foresee adding additional frame types, but as a matter of precaution, this section might define a registry for initial bytes in a frame. At this point, this would contain:

- 0x0a: Diagnostics
- 0x45 to 0x4f: IPv4 packet
- 0x60 to 0x6f: IPv6 packet
- 0xA9: CoAP message with 16-bit FCS

If such a registry is desired, the following values for initial bytes should probably be reserved (while all these values could be used if required, implementation is easier if they are not):

- 0x00
- 0xc0: [RFC1055] END
- 0xdb: [RFC1055] ESC

There might also be a need to formally register the URI scheme "coap+uart".

9. Security Considerations

The usual security considerations apply to the IP packets transferred in packet frames.

When displaying information from diagnostic frames, care should be taken that features of a terminal triggered e.g. by escape sequences cannot be used for nefarious purposes.

The CoAP configuration interface does not itself provide any security. This may be appropriate for the local configuration needs of an experimentation platform that is not expected to be physically connected to any system that is not allowed full control over it (e.g., by using the same physical interface for reflashing new firmware). Where the platform might connect to other systems over serial, object security for CoAP [RFC8613] might be employed, or the configuration interface might be restricted to a read-only mode only

providing information that does not need confidentiality protection. (It would be possible to provide a DTLS encapsulation, but this might go beyond the objective of extreme simplicity.)

10. References

10.1. Normative References

[RFC1055] Romkey, J., "Nonstandard for transmission of IP datagrams over serial lines: SLIP", STD 47, RFC 1055, DOI 10.17487/RFC1055, June 1988, <https://www.rfc-editor.org/info/rfc1055>.

[RFC1662] Simpson, W., Ed., "PPP in HDLC-like Framing", STD 51, RFC 1662, DOI 10.17487/RFC1662, July 1994, <https://www.rfc-editor.org/info/rfc1662>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

10.2. Informative References

[RFC1661] Simpson, W., Ed., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, DOI 10.17487/RFC1661, July 1994, <https://www.rfc-editor.org/info/rfc1661>.

[RFC2687] Bormann, C., "PPP in a Real-time Oriented HDLC-like Framing", RFC 2687, DOI 10.17487/RFC2687, September 1999, <https://www.rfc-editor.org/info/rfc2687>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

Appendix A. Implementation

A work in progress implementation of slipmux is available as part of Lobaro's SLIP implementation:

https://github.com/Lobaro/slip
https://github.com/Lobaro/util-slip

Acknowledgements

TBD

Authors' Addresses

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Tobias Kaupat
Lobaro UG
Tempowerkring 21d
Hamburg D-21079
Germany

Phone: +49-40-22816531-0
Email: tobias.kaupat@lobaro.de