J U N E B Ŋ Ş T 2E 0V 1E 6Z

KISS, HDLC, AX.25 and friends

A while ago, I uploaded my gr-kiss out-of-tree GNUradio module to Github. This is a set of blocks to handle KISS, HDLC and AX.25, which are the protocols used in amateur packet radio. There are several other OOT modules that do similar things, but I didn't like the functionality of them very much. While programming this module, I've also noted that the documentation for these protocols is not so good sometimes. Here I'll give a brief description of the protocols and explain how everything works together.

KISS The KISS protocol was originally designed to interface a computer with a TNC (Terminal Network Controller). Think of the TNC as a sort of modem that does some modulation and low level framing functions. The idea of KISS is to move most of the processing to the host computer. Before KISS, a TNC usually performed some AX.25 related tasks. However, a KISS TNC only works with HDLC and passes raw AX.25 frames to the host. The KISS protocol is described here. It provides a way to separate frames and to send commands to the TNC. Since this protocol is so simple, it is also used in many situations where no TNC is involved. In this case, it is just used to divide a stream of data into frames. Thus, a series of packets can be saved to a file "in KISS format". These packets are usually AX.25 frames, but this need not be the case. Also, different pieces of software can exchange packets using the KISS format.

The end of a frame is marked by the byte 0xC0. The start of a frame need not be explicitly marked (but it is usually also marked with 0xC0). Several consecutive 0xC0 bytes may appear. This doesn't mean that there are empty frames between them. Empty frames are not allowed. The first byte of any frame is used to send commands to the TNC. When a TNC is not involved, the first byte is 0x00. Then the real frame bytes come after it. The byte 0xC0 can not appear anywhere inside a frame. If the data contains this byte, it is replaced by 0xDB 0xDC. If 0xDB appears in the data, it is replaced by 0xDB 0xDD.

HDLC The HDLC protocol is a data link layer protocol used in X.25 and other protocols. In AX.25, only a few functions of HDLC are used to do framing and frame integrity checking. The first remark is that usually HDLC is NRZ-I encoded. This means that a logical 0 bit is marked by a change of state (say, by changing from a positive frequency deviation to a negative frequency deviation or vice versa), and a logical 1 bit is marked by no change of state. The end of frame marker is the sequence 01111110 (or 0x7e). Note that other NRZ-I implementations follow the opposite convention. This is also used to mark the start of a frame and between frames as an idle signal. Before a frame is transmitted, several 0x7e bytes are sent to help the receiver synchronize. Also, after the frame is transmitted, it is usual to send the byte 0x7e several times to ensure that one of these flags is received without errors.

HDLC avoids long runs of 1's (which will make the receiver loose synchronization) by forbidding more than 5 consecutive 1's in the data (except for the 0x7e marker, where 6 consecutive 1's appear). Every time that 5 consecutive 1's appear in the data, a 0 is inserted after them. This is known as bit-stuffing. Of course, when the receiver gets 5 consecutive 1's, if the following bit is a 0 it should be ignored, and if it is a 1, the receiver should expect that the following bit is a 0, as this must be the last bit of the 0x7e marker.

Another important aspect of HDLC is that each byte is transmitted starting with the least significant bit. This is the opposite way as in many other protocols. The last function of HDLC that is used in AX.25 is the frame check sequence (or FCS). This is a 16 bit checksum that is appended to the data frame. The checksum is computed using CRC-16-CCITT. The FCS is sent least-significant-byte first (and remember that each byte is sent least-significant-bit first and that bit-stuffing is done).

Keep in mind that since HDLC uses NRZ-I, it isn't sensitive to the polarity of the signal. Thus a signal can be inverted in polarity without causing problems. For this reason, when BPSK is used to transmit HDLC frames, differential encoding is not used.

AX.25 The AX.25 protocol is a data link layer used by amateur radio. The framing and FCS is done using HDLC as described above. The AX.25 frames do not include the FCS or any other checksum. The details of the protocol are here. When reading that document, keep in mind that the 0x7e flags and the FCS are not really part of the AX.25 frame. This protocol has many features and it won't be described here. Since Linux can handle AX.25, it is useful to know how Linux can be used to deal with AX.25. Also, Wireshark can be used to inspect and analyse AX.25 frames.

Modulations It is also important to know how AX.25 HDLC frames are actually transmitted over the air. The first way to do it is using AFSK. This is normally used for a rate of 300 baud on the HF bands. The NRZ-I bits are transmitted as an audio signal which frequency shifts between two tones spaced 200Hz apart. The radio is set to SSB mode, so the actual emission is really FSK. The particular tones that are used are not standard, so this has to be compensated by setting the radio dial frequency correctly. It is not important whether LSB or USB mode is used, because the signal is not sensitive to polarity inversion.

The second way to do it is using FM AFSK. This is normally used for a rate of 1200 baud on the VHF and UHF bands. The NZR-I bits are transmitted as an audio signal which frequency shifts between the tones 1200Hz and 2200Hz. This audio signal is FM modulated before transmission.

The third way to do it is using G3RUH FSK. This is normally used for a rate of 9600 baud on the UHF bands. The HDLC bitstream is scrambled using a multiplicative scrambler with polynomial \(1 + x^{12} + x^{17}\) before NRZ-I encoding is done. The NRZ-I signal is then shaped (low-pass filtered) and used to drive an FM modulator directly, in order to produce an FSK signal.

The fourth way to do it is using BPSK. This is used in a few amateur satellites, using a rate of 1000 or 1200 baud. The NRZ-I bits are transmitted as a BPSK signal (differential encoding is not used). This BPSK signal can be generated as an audio signal on a computer and then used to drive an SSB transmitter.

Finally, fldigi can act as a KISS TNC, allowing to send AX.25 frames in many of the modes supported by this program. However, these digital modes are normally used for text based chat and rarely used for AX.25.

gr-kiss includes example flowgraphs showing how the 1k2 FM AFSK, 9k6 FSK and 1k2 BPSK modulations work.

Interfacing AX.25 with Linux As we have stated above, it is sometimes useful to interface an AX.25 system with Linux. To use the AX.25 functionality of Linux, first one needs to declare a "port" for each AX.25 interface that Linux will handle. The ports are declared in `/etc/ax25/axports`.

```
# /etc/ax25/axports
#
# The format of this file is:
#
# name callsign speed paclen window description
#
test EA4GPZ-10 38400 2000 2 test
test2   EA4GPZ-9 38400 2000 2 test2
```

This is an example. The speed is not very important if these ports won't connect to a hardware TNC through a serial (RS232) port.

Each of the ports needs to be attached to a serial port or a pty device (a sort of virtual serial port). If we want to connect some AX.25 software to a Linux AX.25 interface, we can use two tools: kissnetd and socat. kissnetd is used to create a pair of pty devices which are connected together. Everything written to one of the pty devices can be read on the other one and vice versa. This is fine if our AX.25 software can read and write to a pty device.

As an example, let's use kissnetd to send a KISS file to a Linux AX.25 interface. This can be used to analyse the frames with wireshark, by making wireshark capture traffic on the AX.25 interface.

```
# kissnetd -p 2 &
kissnetd V 1.5 by Frederic RIBLE F1OAT - ATEPRA FPAC/Linux Project

Awaiting client connects on:
/dev/pts/6 /dev/pts/7

# kissattach /dev/pts/6 test
AX.25 port test bound to device ax0
# cat file.kiss > /dev/pts/7
```

socat is a much more flexible tool. It is also used to make two "devices" which are connected together. However, it supports a number of different devices: pty's, TCP and UDP sockets, files, pipes, UNIX sockets… As an example, let's use socat to connect a pty with an UDP socket and then send a KISS file by UDP.

```
# socat PTY,link=/tmp/serial UDP-LISTEN:1234 &
# kissattach /tmp/serial test
AX.25 port test bound to device ax0
$ nc -4u localhost 1234 < sats/firebird-4-2015032418.kiss
nc: using datagram socket
```

As another example, we can use socat to set up a TCP client that connects to one of the examples in gr-kiss.

```
# # run the example in gr-kiss
# socat PTY,link=/tmp/serial TCP:localhost:52001 &
# kissattach /tmp/serial test
AX.25 port test bound to device ax0
```

Basic usage of Linux AX.25 interfaces Linux AX.25 interfaces can be configured for IPv4 in the same way as any other network interface. Then, the usual tools for IP traffic can be used. There are several other tools specific to AX.25. `listen` can be used to monitor AX.25 traffic. Keep in mind that Wireshark can also capture and analyse traffic in AX.25 interfaces. `call` can be used to make AX.25 connections. It is also an easy way to generate test packets.

SOFTWARE

D I G I ,TG AN IU R MA OD DI ID S

11 Replies to “KISS, HDLC, AX.25 and friends”

Bob Mattaliano
JUNE 12, 2016 AT 19:39 UTC

Thanks Dani. Great tools for GNU Radio and very helpful explanations about these protocols.

Reply

Richard Stanley
AUGUST 6, 2016 AT 19:09 UTC

Awesome. I just started reading about ax.25, and this is REALLY helpful. I might be doing some experiments with one of my raspberry pi zeros, ax.25, TCP/ip and file dropping with ftp or tftp (no encryption or security). I would love for my local ham radio club to learn how to file drop with no internet nessisary.

Reply

Pingback: GNU Radio decoders for several Amateur satellites | Iz4fvw.net

Pingback: D-SAT support added to gr-satellites – Daniel Estévez

Jeremy Kitchen
MAY 28, 2019 AT 05:28 UTC

So, I’m confused. The pdf you link to there of the AX.25 spec says there are 2 bytes in the frame just after the control field that are the FCS. However, you also say that they aren’t there. And in my exploration of real world usage so far (sniffing the packets of my winlink client checking in), I concur, that FCS field isn’t being sent with any packet I’ve seen.

Is the spec wrong? Or is that a newer version than most implementations use? Or something else?

I'm working on an AX.25 library for Swift so I can do some packet stuff on my mac / iOS devices and am getting to the FCS portion of the frame 🙂

Thanks!

Reply

destevez

MAY 28, 2019 AT 17:21 UTC

Hi Jeremy,
Sorry about my wording. Perhaps it was confusing. In the terminology I follow in this post (of course there are different ways to explain things), the 0x7e flags and the CRC form part of the HDLC frame. The AX.25 frame (addresses, PID, etc.) is transmitted as the payload of the HDLC frame, and so it doesn't include the 0x7e flags nor CRC. So in this post I don't consider the CRC to form part of the AX.25 frame, but rather of the HDLC frame.

This is consistent with the common use of AX.25 frames through a KISS interface. As a matter of fact, CRC's are not transmitted through the KISS interface.

Reply

Jeremy Kitchen

MAY 28, 2019 AT 21:36 UTC

thanks for the reply!

so an HDLC frame is like a different wrapper around the ax.25 frame but also affects the contents? The spec says there's a protocol field and an FCS field, so I'm confused 🙂 I don't have any experience with HDLC or even know where I would use that, I'm working from a KISS TNC, so maybe things are very different, then 🙂

If there's no CRC with KISS+AX.25, how is packet integrity verified? Or is that left as an exercise for the layer 3 or layer 7 implementation?

Reply

destevez

MAY 28, 2019 AT 21:42 UTC

In AX.25, the physical (OSI L1) and link (OSI L2) layers are somewhat mixed up, so I prefer to distinguish them and say that HDLC is the physical layer, while AX.25 is the link layer. This makes sense, as AX.25 can then be sent on top of other physical layers, such as KISS. (Note that however this correspondence doesn't follow the OSI model 100% accurately).

An HDLC frame doesn't affect the contents of an AX.25 frame. It just encapsulates it.

There is no CRC in AX.25 on top of KISS. Packet integrity is taken for granted, since this is used for a serial link between a host and a TNC, for routing AX.25 between different software, or for storage of AX.25 frames. Of course, when transmitting AX.25 over the air, it is encapsulated on HDLC, so a CRC is added.

Reply

Jeremy Kitchen
MAY 29, 2019 AT 00:12 UTC

OH! So the TNC actually adds its own CRCing before sending it over the air? And that's the HDLC stuff? Interesting! Thanks for the info!

Reply

destevez
MAY 29, 2019 AT 05:53 UTC

Yes, the CRC is usually added/checked by the TNC.

Reply

Pingback: DSLWP-B whole mission telemetry – Daniel Estévez

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

PREVIOUS
← Concurso Sant-Sadurní V-UHF

NEXT
Using the CC1101 and Beaglebone black for IP traffic on 70cm →

Search …

CATEGORIES

- Amateur radio
  - Events
  - Experiments
  - Hardware
  - Opinion
  - Software
- Electronics
  - Projects
  - Test equipment
- Maths
- Space

TAGS

3cat-2 10ghz aausat ads-b arduino astronomy astrophotography contests digital modes doppler dslwp dsp eshail2 fec freedv frequency ft817 funcube gmat gnss gnuradio gomx hermeslite hf jt kits lilacsat limesdr linrad microwaves mods moonbounce networks noise orbital dynamics outernet radiosonde receivers repairs rf amplifiers satellites sdr signal generators vhf & uhf vlbi

ARCHIVES

- January 2020
- December 2019
- November 2019

October 2019
September 2019
August 2019
July 2019
June 2019
May 2019
April 2019
March 2019
February 2019
January 2019
December 2018
November 2018
October 2018
September 2018
August 2018
July 2018
June 2018
May 2018
April 2018
March 2018
February 2018
January 2018
December 2017
November 2017
October 2017
September 2017
August 2017
July 2017
June 2017
May 2017
April 2017
March 2017
February 2017
January 2017
December 2016

November 2016

October 2016

September 2016

August 2016

July 2016

June 2016

May 2016

April 2016

March 2016

February 2016

January 2016

December 2015

November 2015

October 2015