ACS - Autopilot Telemetry Radios

This document provides an overview of how serial telemetry radios are used to communicate directly with aircraft autopilots. These radios are utilized to test and debug autopilot behavior, as well as to provide a secondary connection to the aircraft in the event of a payload computer or wireless link failure.

Radio Description

Autopilots running the ardupilot firmware can speak the MAVLink protocol across serial links. 3D Robotics supplies a "telemetry radio," which provides a bidirectional wireless serial link between two endpoints (nominally one aircraft and one ground station). Their radio boards use the Silicon Labs Si1000 wireless MCU (thus the firmare is named 'SiK' for Si1000).

These radios allow a MAVLink-speaking ground station (e.g., Mission Planner or MAVProxy) to communicate directly with an autopilot while the aircraft is in flight. The radio passes whatever serial traffic the aircraft emits to the ground station, and vice versa. Aircraft by default emit some gratuitous MAVLink traffic, and upon request from the ground station, emit their "full rate" of traffic.

Network Identifiers

To distinguish between multiple pairs of aircraft and ground station, radios can be configured with a "netid" parameter. Radios will only accept incoming packets with a matching netid in the packet header. In the current firmware, only two radios may participate in a netid at once; there is no multiple access protocol in place.

Frequency Hopping and Bandwidth

In the USA, 3DR radios operate in the 902-928 MHz ISM band, though by default the radios are configured to operate between 915-928 MHz (which is the equivalent band in Australia). The SiK firmware implements a frequency hopping mechanism within this band, subdividing the bandwidth into 50 (by default) "channels" that the radio hops between using a predefined pattern based on netid.

Intuitively, with at least 51 radios in use (50 aircraft plus at least one ground station) and only 50 frequency hopping channels, the band could be saturated even by the gratuitous traffic load. Empirical tests confirmed that the noise level (which includes any traffic for other netids) is significant with greater than 30 aircraft powered at once.

To mitigate this, a different scheme is currently used that subdivides the full 902-928 MHz band into three 10 MHz sub-bands: 902-910 MHz, 911-919 MHz, and 920-928 MHz. The default of 50 channels is still used,
thus the width of each channel is reduced. Radios alter their modulation accordingly to maintain the same bitrate, with the effect of reducing receiver range. Alternatively, the requested bitrate could also be reduced.

Radios are assigned to sub-bands based on netid: a radio with netid 1 is assigned to sub-band 911-919 MHz, and in general, the min and max frequencies are determined by:

\[
\begin{align*}
\text{min\_freq} &= 902 \text{ MHz} + (\text{netid mod 3}) \times 9 \text{ MHz} \\
\text{max\_freq} &= 910 \text{ MHz} + (\text{netid mod 3}) \times 9 \text{ MHz}
\end{align*}
\]

For 50 aircraft, each sub-band will have to support at most 17 aircraft across its unique 50 frequency hopping channels. Note that this approach will work for a single fleet of aircraft; if needing to operate multiple fleets in parallel, a different frequency allocation scheme may be necessary.

**Configuration Tools**

Some ground station software, including Mission Planner, includes telemetry radio configuration utilities. A command-line utility, `atcommander.py` in the SiK repository, was developed in-house for radio configuration. It is available in the standard ACS installation, and can be found in the SiK repository on GitHub or via the links below. Another utility, `telem_config.sh` in the `acs-env` repository, wraps `atcommander.py` and a MAVLink traffic-checking script to automate radio configuration for the above frequency scheme.

Note: On Linux systems, radios are visible as `/dev/ttyUSB*` and as `/dev/serial/by-id/usb-FTDI_FT231X_USB_UART_*`; the latter is a softlink that is unique to the radio, regardless of which TTY the kernel chooses for the radio. This turns out to be the safer filename to use with the radio; an open softlink will be redirected to the correct TTY even if the radio is unplugged and replugged with the filehandle open. However, for the usage examples below, the shorter file paths will be used.

**atcommander.py**

To view the configuration of a locally-attached radio:

```
atcommander.py -l /dev/ttyUSB0
```

To set the netid, minfreq, and maxfreq of a locally-attached radio (note that frequencies are in kHz):

```
atcommander.py -L netid 5 -L minfreq 920000 -L maxfreq 928000 /dev/ttyUSB0
```

The `-r` and `-R` flags may be used to get and set attributes on a remote radio; commands are forwarded across the locally-connected radio, assuming that the two radios are initially configured with compatible parameters. If changing both a remote and a local radio, be sure to change the remote radio first since the local radio will no longer be able to communicate with the remote radio once reconfigured. The `-B` option will set an attribute on the remote and then the local radio.

When in doubt, try the `-f` option to force the radio out of bootloader mode (it sometimes gets stuck in this mode), or try unplugging and replugging (or otherwise power cycling) the radio.
telem_config.sh

**telem_config.sh** can be used to configure a new aircraft radio (which comes with defaults of netid = 25, minfreq = 915000, and maxfreq = 928000), or to reconfigure an aircraft radio once it has been set to a sub-band configuration. It can also configure a locally attached radio to talk to a plane.

To configure a locally attached radio to talk to a configured aircraft radio with netid = 37:

```
telem_config.sh /dev/ttyUSB0 37
```
To configure a new aircraft radio to netid = 42 (power up aircraft, have locally attached radio plugged in; script will handle local radio reconfiguration and confirmation of settings):

```
telem_config.sh -c /dev/ttyUSB 42
```
To reconfigure an aircraft radio from netid = 42 to netid = 26 (power up aircraft, have locally attached radio plugged in; script will handle local radio reconfiguration and confirmation of settings):

```
telem_config.sh -r -n 42 /dev/ttyUSB0 26
```
Note that **telem_config.sh** uses another program, **mavdump.py**, to confirm that a connection exists to the aircraft before it proceeds with reconfiguration. It will stop and report an error if the connection cannot be verified.

**References**

Information about configuring the radios:


Even more information:


Main firmware repo on GitHub (with atcommander.py):

https://github.com/Dronencode/SiK

Fork of firmware repo with improvements to atcommander.py:

https://github.com/mikeclement/SiK/tree/mc_ate_unstick