NAME

aprx − A receive-only APRS iGate application,

SYNOPSIS

aprx [−d[−d]] [−e] [−v] [−l syslogfacility] [−f /etc/aprx.conf]

DESCRIPTION

aprx is a *receive-only* APRS iGate application with minimum system support technology requirements.

- Can listen KISS-TNCs on ordinary serial ports (real ones, or USB ones, all work as long as they respond to normal UNIX serial port ioctl()s.) Maximum number of separate serial ports used for radio modem communication is 16, limit is easy to increase in source code.
- On Linux machine with kernel internal AX.25 protocol support, does want to listen on it with promiscuous mode, and in order to use that, must be started as *root* user. The AX.25 socket listening is *not* configurable, it is always on in Linux systems.
- Does not require machine to have AX.25 protocol support internally! (Thus this works also on e.g. BSD machines without PF_AX25 sockets.)
- Does not require machine to have any other software in it, than things in libc. In particular no special AX.25 libraries at all, nor widgets or even C++ runtime.
- Connects with one callsign-ssid pair to APRS-IS core for all received radio ports.
- Knows that messages with following tokens in AX.25 VIA fields are not to be relayed into APRS-IS network:
  RFONLY, NOGATE, TCPIP, TCPXX
- Knows that following source address prefixes are bogus and thus messages with them are to be junked:
  WIDE, RELAY, TRACE, TCPIP, TCPXX, NOCALL, N0CALL
- Drops all *query* messages ("?"), as well as all 3rd party messages ("}").
- Has built-in "Erlang monitor" mechanism, that logs to syslog a interface specific channel occupancy. Optionally can log to STDOUT and/or to syslog.

OPTIONS

The **aprx** has following runtime options:

−d Turn on verbose debugging, outputs data to STDOUT.

−dd the "more debug" mode shows also details of network interaction with the APRS-IS network service.

−e *Erlang output* prints 10 minute and 60 minute traffic accumulation byte counts, and guestimates on channel occupancy, alias "Erlang". These outputs are sent to STDOUT, which system operator may choose to log elsewere. This is independent if the "−l" option below.

−l logfacilityname
Defines syslog**(3)** facility code used by the erlang reporter by defining its name. Default value is: **NONE**, and accepted values are: **LOG_DAEMON**, **LOG_FTP**, **LOG_LPR**, **LOG_MAIL**, **LOG_NEWS**, **LOG_USER**, **LOG_UUCP**, **LOG_LOCAL0**, **LOG_LOCAL1**, **LOG_LOCAL2**, **LOG_LOCAL3**, **LOG_LOCAL4**, **LOG_LOCAL5**, **LOG_LOCAL6**, **LOG_LOCAL7**. That list is subject to actual facility code set in the system, and in any case if you specify a code that is not known, then the program will complain during the startup, and report it. This is independent of the "−e" option above.

−v Verbose logging of received traffic to STDOUT. Lines begin with reception timestamp (UNIX time_t seconds), then TAB, and either data as is, or with prefix byte: "*" for "discarded due to data content", or possibly "#" for "discarded due to APRS-IS being unreachable".

−f /etc/aprx.conf

Configuration file, given path is built-in default, and can be overridden by the program runner.

Use parameter set **−ddv** to test new configuration by running it synchronously to console.

CONFIGURATION FILE

The configuration file is used to setup the program to do its job. Some configurations can be done without the file, namely a channel *Erlang monitor* siphoning APRS packets off the Linux kernel AX.25 network stack.

On the configuration file, following additional notes

- Most parameters are singletons, that is, there can be only one.
- There can be up to (by default) 16 entries of "serialport".
- There can be unlimited number of "netbeacon" lines.
- There can be unlimited number of "ax25-filter" lines.
- There can be unlimited number of "ax25-rxport" lines.

The configuration file is read upon program start, ever latter.

```
#
#  Sample configuration file for the APRX -- an Rx-only APRS iGate
#
#

# The  mycall  parameter:
# Station call-id used for relaying APRS frames into APRS-IS.
#
#mycall           N0CALL-1

# APRS-IS server name and portnumber.
# Every reconnect does re-resolve the name to IP address.
# There can be up to 4 system definitions without code modification,
# they are used in round-robin fashion.  Heartbeat and filter
# definitions must follow each server definition.
#
#aprsis-server    rotate.aprs.net  14580

# Some APRS-IS servers tell every about 20 seconds to all contact
# ports that they are there and alive. Others are just silent.
# Enable only if the server you use does present heartbeat.
# Recommended value 3*"heartbeat" + some  -> 120 (seconds)
#
#aprsis-heartbeat-timeout   120

# APRS-IS server may support some filter commands.  Although this
# program does not transmit out to RF, filter rules can be used to
# ensure that there is sufficient dataflow from APRS-IS server to
# this program that it very likely will not timeout within network
# monitoring timeout..
#
#aprsis-filter "some filter specs in quotes"

# AX.25 filters block selected messages matching on selected regular
```

```
# expressions.  The expressions are case sensitive, and AX.25 address
# elements are in all uppercase text.  There can be unlimited number
# of patterns, type fields are four: "source", "destination", "via",
# and "data".  These patterns can be used in addition to built-in
# hard-coded reject rules listed in documentation.
#
#ax25-filter source      "^NOCALL"
#ax25-filter destination "^NOCALL"
#ax25-filter via         "^NOGATE"
#ax25-filter data        "^\\?"

# ax25-rxport limits reception on listed AX.25 ports, if system
# happens to use AX.25 ports also for other purposes than APRS.
# If this option is not used, all reception ports are accepted.
# Number of port definitions here is unlimited.
#
#ax25-rxport ax0
#ax25-rxport ax1

# rflog defines a rotatable file into which all RF-received packets
# are logged.
#
#rflog /tmp/aprx-rf.log

# aprxlog defines a rotatable file into which most important
# events on APRS-IS connection are logged, namely connects and
# disconnects.
#
#aprxlog /tmp/aprx.log

# erlangfile defines a mmap():able binary file, which stores
# running sums of interfaces upon which the channel erlang
# estimator runs, and collects data.
# Depending on the system, it may be running on a filesystem
# that actually retains data over reboots, or it may not.
# With this backing store, the system does not loose cumulating
# erlang data over the current period, if the restart is quick,
# and does not stradle any exact minute.
# (Do restarts at 15 seconds over an even minute..)
# This file is around 0.5 MB per each interface talking APRS.
# Things go BADLY WRONG if this file can not be created or
# it is corrupted!
#
# Built-in default value is: /tmp/aprs-erlang.dat
#
erlangfile /tmp/aprx-erlang.dat

# erlang-loglevel is config file edition of the "-l" option
# pushing erlang data to syslog(3).
# Valid values are (possibly) following: NONE, LOG_DAEMON,
# LOG_FTP, LOG_LPR, LOG_MAIL, LOG_NEWS, LOG_USER, LOG_UUCP,
# LOG_LOCAL0, LOG_LOCAL1, LOG_LOCAL2, LOG_LOCAL3, LOG_LOCAL4,
# LOG_LOCAL5, LOG_LOCAL6, LOG_LOCAL7.  If the parameter value is
# not acceptable, list of accepted values are printed at startup.
```

```
#
#erlang-loglevel NONE

# erlang-log1min option logs to syslog/file also 1 minute
# interval data from the program. (In addition to 10m and 60m.)
#
#erlang-log1min

# The  serialport  option.  Parameters are:
#   - /dev/ttyUSB1    -- tty device
#   - 19200           -- baud rate, supported ones are:
#                        1200, 2400, 4800, 9600, 19200, 38400
#   - 8n1             -- 8-bits, no parity, one stop-bit,
#                        no other supported modes
#   - KISS/XORSUM/BPQCRC/SMACK/CRC16  -- KISS mode
#
# There can be up to 16 serialport definitions in this file!
#
#serialport   /dev/ttyUSB1  19200 8n1    KISS

# Additional options for the "serialport" line.
#
# "initstring" is of two parts, the keyword, and then a string.
#    initstring "\xC0\xC0\xFF\xC0\r\nMO 0\rKISS $01\r"
#
#  "KISS"                  - plain basic KISS mode
#  "XORSUM" alias "BPQCRC" - KISS with BPQ "CRC" byte
#  "SMACK" alias "CRC16"   - KISS with better CRC


# The  netbeacon  option.
# Parameter string (in quotes) is sent to network (without quotes)
# at varying intervals --  1200-1800 seconds in between restransmits.
# This interval is intentionally randomized.
#
# There can be multiple netbeacon options.
# Symbol  R&   is for "rx-only iGate"
#
#netbeacon  "!6016.35NR02506.36E&aprx Rx-only 'iGate'"
```

In the configuration file there is special treatment for quoted strings. They are stripped of the outer quotes, and "\" character is processed within the source string to produce an output string. The escapes are:

`\n` Produces newline character (Control-J) on the output string.

`\r` Produces carriage return character (Control-M) on the output string.

`\\` Places a back-slash on the output string.

`\"` Places a double-quote on the output string.

`\'` Places a single-quote on the output string.

`\xHH` Lower-case "x" precedes two hex digits which ensemble is then converted to a single byte in the output string.

The complex encodings are for possible init-strings of the external devices, *however: a nul byte is not possible to produce as it terminates a string!* (= "`\x00`")

A configuration token without surrounding quotes does not understand the backslash escapes.

NOTES: ERLANG

The *Erlang* is telecom measurement of channel occupancy, and in this application sense it does tell how much traffic there is on the radio channel.

Most radio transmitters are not aware of all transmitters on channel, and thus there can happen a collision causing loss of both messages. The higher the channel activity, the more likely that collision is. For further details, refer to statistical mathematics books, or perhaps on Wikipedia.

In order to measure channel activity, the **aprx** program suite has these built-in statistics counter and summary estimators.

The *Erlag* value that the estimators present are likely somewhat *underestimating* the true channel occupancy simply because it calculates estimate of channel bit transmit rate, and thus a per-minute character capacity. It does not know true frequency of bit-stuffing events of the HDLC framing, nor each transmitter pre- and port frame PTT times. The transmitters need to stabilize their transmit oscillators in many cases, which may take up to around 500 ms! The counters are not aware of this preamble-, nor postamble-times.

The HDLC bit stuffing ratio is guessed to be 8.2 bits for each 8 bits of payload.

NOTES: PROGRAM NAME

Initially this program had name *aprsg-ng*, which was same as another (less low-tech C++ approach) had.

BUGS

The *Erlang*-monitor mechanisms are of rudimentary quality, and can seriously underestimate the channel occupancy.

SEE ALSO

Couple web sites: *http://www.aprs-is.net/* , *http://www.aprs2.net/*

aprx-stat(8)

AUTHOR

This little piece was written by *Matti Aarnio, OH2MQK* during a dark and rainy fall and winter of 2007-2008 after a number of discussions grumbling about current breed of available software for APRS iGate use in Linux (or of any UNIX) platforms.

Principal contributors and test users include: *Pentti Gronlund, OH3BK*, *Reijo Hakala, OH1GWK*.