

NAME

Aprx-2 – An APRS iGate application with integrated Digipeater.

SYNOPSIS

aprx [-d[d[d]]] [-e] [-v] [-I *syslogfacilityname*] [-f */etc/aprx.conf*]

DESCRIPTION

The **aprx** program is a special purpose Ham-radio application supplying infrastructure services for APRS protocol use.

A more detailed manual is available at: <http://ham.zmailer.org/oh2mqk/aprx/aprx-manual.pdf>

FEATURES

The Aprx begun as a receive-only APRS iGate application with minimum system support technology requirements. This version has also multi-port digipeater support, transmit iGate, and experimental D-PRS-to-APRS RF/Rx-iGate.

- The Aprx does not require machine to have any other software in it, than things in UNIX standard libc. In particular no special AX.25 libraries at all, nor widgets or even C++ runtime.
- Important goal has been to keep R/W memory footprint as small as possible, and on general purpose i386 Linux a single radio port iGate+digipeater is now around 250 kB of R/W memory allocations.
- Any UNIX (and UNIX like) platform should work for the Aprx, or be trivially ported.
- The Aprx can listen "TNC2 monitor" and "KISS" speaking TNCs on any serial ports.
- For Aprx the serial port can be ordinary host computer port, a USB serial port, or remote port on a remote server behind the internet, like cisco router AUX ports (port 4001, TCP STREAM without TELNET escapes.)
- The Aprx does not require machine to have AX.25 protocol support internally! (Thus this works also on e.g. Solaris and BSD machines without PF_AX25 sockets.)
- On Linux machine with kernel internal AX.25 protocol support, the Aprx can listen on it with promiscuous mode and in order to use that, the Aprx must be started as *root* user, and be configured to list interface callsigns that APRS packets are coming in. The AX.25 socket listening is not in itself configurable, it is always exists in Linux systems, and related configuration parameters are ignored in other platforms. This socket listening does not need auxiliary "libax25" to function.
- The Aprx program can be run without root privileges at least against remote serial port servers. One must change local serial port ownership or access-groups (if any are used) to userid that runs the program and possibly do several changes of file paths in configuration file beginning with its location (startup parameter). How that is done is up to the user or system integrator of this program.
- The Aprx connects with one callsign-ssid pair to APRS-IS core for all received radio ports.
- The Aprx Rx-iGate knows that messages with following tokens in AX.25 VIA fields are not to be relayed into APRS-IS network:
RFONLY, NOGATE, TCPIP, TCPXX
- The Aprx Rx-iGate knows that following source address prefixes are bogus and thus messages with them are to be junked:
WIDE, RELAY, TRACE, TCPIP, TCPXX, NOCALL, N0CALL
- The Aprx Rx-iGate Drops all *query* messages ("?").
- The Aprx Rx-iGate opens up all 3rd party messages ("}"), and checks the internal data if it is OK to be gated out to APRS-IS.
- The Aprx has built-in "Erlang monitor" mechanism that telemeters each receiving interface to APRS-IS. It can also syslog the interface specific channel occupancy, and optionally can output to STDOUT.
- The Aprx (since version 1.91) can do digipeater functions.

- The Aprx (since version 1.99) does have experimental D-STAR D-PRS to APRS gateway functionality. See the *aprx-manual.pdf* for details.
- The Aprx can be run on systems without writable storage, even with very little memory, like on NSLU2, and OpenWrt platforms. The experiments have shown that a single radio Tx-iGate+digipeater works with less than 300 kB of writable RAM for the Aprx itself. Additional memory is necessary for operating system services of TCP/IP networking, and serial port drivers.

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.
- ddd** the "even more debug" mode shows also detail classification of every kind of frame received in KISS variants.
- e** *Erlang output* prints 10 minute and 60 minute traffic accumulation byte counts, and guesimates on channel occupancy, alias "Erlang". These outputs are sent to STDOUT, which system operator may choose to log elsewhere. This is independent if the "-l" option below.
- f /etc/aprx.conf**
Configuration file, given path is built-in default, and can be overridden by the program runner.
- l syslogfacilityname**
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".

DEBUGGING SYSTEM

Use parameter set **-ddv** (or **-dddv**) to test new configuration by running it synchronously to console.

NORMAL OPERATION

Running the **aprx** program without any of option flags: **-d**, **-v**, or **-e** reads possibly given configuration, then automatically backgrounds the process, and writes *pidfile*. When the process whose number written in *pidfile* is then sent a SIGTERM signal, it automatically shuts down itself, and removes the *pidfile*. The *pidfile* can be runtime configured with the **-f /etc/aprx.conf** file, and it has default name of: */var/run/aprx.pid*.

CONFIGURATION FILE

The configuration file is used to setup the program to do its job.

You can construct following configurations:

- A *receive-only* iGate server.
- A digipeater with bi-directional iGate server.
- A *single radio* digipeater. (The most common type of digipeater.)
- A *multi-interfaced* digipeater relaying traffic in between multiple radios. (On same or on separate frequencies.)
- A *viscuous digipeater*, which relays a packet it heard from viscuous source after the viscuous delay, *unless it was heard more times than only once*, or it was heard from non-viscuous source before the

viscuous one was digipeated. This allows of making fill-in digipeaters that will not digipeat the packet, if that same packet was heard twice or more before the viscous delay expired.

In the configuration file a line ending backslash (\) character concatenates next input line into itself. Combined result can be up to 8000 bytes long. This combination can be a bit surprising:

```
#beacon .... long text \
      continuation
```

results in single long input line that begins with '#' (it is comment) and all continuations following it have been folded in. Presented line number of combined continuation is the line number of the *last* line segment in this type of multi-line input.

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 initstrings of the external devices, and in particular for initstrings even a nul byte (\x00) is supported.

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

```
#
# Sample configuration file for the APRX -- an Rx-only APRS iGate with
# Digipeater functionality.
#
#
# Simple sample configuration file for the APRX-2
#
# This configuration is structured with Apache HTTPD style tags
# which then contain subsystem parameters.
#
#
# For simple case, you need to adjust 4 things:
# - Mycall parameter
# - Select correct type of interface (ax25-device or serial-device)
# - Optionally set a beacon telling where this system is
# - Optionally enable digipeater with or without tx-igate
#
#
# Define the parameters in following order:
# 1) <aprsis>      ** zero to many
# 2) <logging>     ** zero or one
# 3) <interface>   ** one to many
# 4) <beacon>      ** zero to many
# 5) <telemetry>  ** zero to many
# 6) <digipeater> ** zero to many (at most one for each Tx)
#
```

```

#
# Global macro for simplified callsign definition:
# Usable for 99+% of cases.
#

mycall NOCALL-1

<aprsis>
# The login parameter:
# Station call-id used for relaying APRS frames into APRS-IS.
# Use this only to define other callsign for APRS-IS login.
#
#login      OTHERCALL-7 # login defaults to $mycall

# APRS-IS server name and portnumber.
# Every reconnect does re-resolve the name to IP address.
# Some alternates are shown below, choose something local to you.
#
server      rotate.aprs.net      14580
#server     finland.aprs2.net    14580
#server     igates.aprs.fi       14580

# Some APRS-IS servers tell every about 20 seconds to all contact
# ports that they are there and alive. Others are just silent.
# Recommended value 3*"heartbeat" + some -> 120 (seconds)
#
#heartbeat-timeout 0 # Disabler of heartbeat timeout

# APRS-IS server may support some filter commands.
# See: http://www.aprs-is.net/javAPRSFilter.aspx
#
# You can define the filter as single long quoted string, or as
# many short segments with explaining comments following them.
#
# Usability of these filters for a Tx-iGate is dubious, but
# they exist in case you for example want to Tx-iGate packets
# from some source callsigns in all cases even when they are
# not in your local area.
#
#filter "possibly multiple filter specs in quotes"
#
#filter "m/100" # My-Range filter
#filter "f/OH2XYZ-3/50" # Friend-Range filter
</aprsis>

<logging>
# pidfile is UNIX way to tell that others that this program is
# running with given process-id number. This has compiled-in
# default value of: pidfile /var/run/aprx.pid
#
#pidfile /var/run/aprx.pid

# rflog defines a rotatable file into which all RF-received packets

```

```

# are logged.
#
#rflog /var/log/aprx/aprx-rf.log

# aprxlog defines a rotatable file into which most important
# events on APRS-IS connection are logged, namely connects and
# disconnects.
#
#aprxlog /var/log/aprx/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 lose cumulating
# erlang data over the current period, if the restart is quick,
# and does not straddle any exact minute.
# (Do restarts at 15 seconds over an even minute..)
# This file is around 0.7 MB per each interface talking APRS.
# If this file is not defined and can not be created,
# internal non-persistent in-memory storage will be used.
#
# Built-in default value is: /var/run/aprx.state
#
#erlangfile /var/run/aprx.state

# 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

# erlanglog defines a rotatable file into which erlang data
# is written in text form.
#
#erlanglog /var/log/aprx/erlang.log

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

# ***** Multiple <interface> definitions can follow *****

# ax25-device Lists AX.25 ports by their callsigns that in Linux
# systems receive APRS packets. If none are defined,

```

```

#           or the system is not Linux, the AX.25 network receiver
#           is not enabled.  Used technologies need at least
#           Linux kernel 2.4.x
#
# tx-ok      Boolean telling if this device is able to transmit.
#
#<interface>
#   ax25-device $mycall # Either $mycall macro, or actual callsign
#   #tx-ok      false # transmitter enable defaults to false
#</interface>

# The TNC serial options.  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"       - plain basic KISS mode
#   - "XORSUM" alias "BPQCRC" - KISS with BPQ "CRC" byte
#   - "SMACK"  alias "CRC16"  - KISS with real CRC
#   - "FLEXNET" - KISS with real CRC
#   - "TNC2"    - TNC2 monitor format
#   - "DPRS"    - DPRS (rx) Gateway
#
#<interface>
#   serial-device /dev/ttyUSB0 19200 8n1 KISS
#   #callsign     $mycall # Either $mycall macro, or actual callsign
#   #tx-ok       false  # transmitter enable defaults to false
#</interface>
#
#<interface>
#   serial-device /dev/ttyUSB1 19200 8n1 TNC2
#   #callsign     $mycall # Either $mycall macro, or actual callsign
#   #tx-ok       false  # TNC2 monitor can not have transmitter
#</interface>
#
#<interface>
#   serial-device /dev/ttyUSB1 19200 8n1 DPRS
#   callsign      dprsgwcallsign # must define actual callsign
#   #tx-ok       false  # DPRS monitor can not do transmit
#</interface>
#

# ***** Multiple <beacon> definitions can follow *****
<beacon>
#
# Beacons are sent out to radio transmitters AND/OR APRSIS.
# Default if "both", other modes are settable.
#
#beaconmode { aprsis | both | radio }
#
# Beacons are sent from a circular transmission queue, total cycle time
# of that queue is 20 minutes by default, and beacons are "evenly"

```

```

# distributed along it. Actual intervals are randomized to be anything
# in between 80% and 100% of the cycle-size / number-of-beacons.
# First beacon is sent out 30 seconds after system start.
# Tune the cycle-size to be suitable to your number of defined beacons.
#
#cycle-size 20m
#
#
# Basic beaconed thing is positional message of type "!":
#
#beacon symbol "R&" lat "6016.35N" lon "02506.36E" comment "Rx-only iGate"
#
# Following are basic options:
# 'symbol'      no default, must be defined!
# 'lat'         coordinate latitude:  ddmm.mmN  (no default!)
# 'lon'         coordinate longitude: dddmm.mmE  (no default!)
# 'comment'     optional tail part of the item, default is nothing
#
# Sample symbols:
# R&  is for "Rx-only iGate"
# I&  is for "Tx-iGate"
# /#  is for "Digipeater"
# I#  is for "Tx-iGate + Digipeater"
#
# Additional options are:
# 'srccall'    parameter sets claimed origination address.
# 'dstcall'    sets destination address, default "APRXnn"
# 'interface'  parameter picks an interface (must be "tx-ok true" type)
# 'via'        sets radio distribution pattern, default: none.
# 'timefix'    On APRS messages with HMS timestamp (hour:min:sec), the
#              system fixes appropriate field with transmit time timestamp.
#
# Message type is by default '!', which is positional no timestamp format.
# Other possible formats are definable with options:
# 'type'       Single character setting type:  ! = / @
# 'item'       Defines a name of Item ('') type beacons.
# 'object'     Defines a name of Object (';') type beacons.
#
# 'file' option tells afile at which a _raw_ APRS message content is
#           expected to be found as first line of text. Line ending newline
#           is removed, and no escapes are supported. The timefix is
#           available, though probably should not be used.
#
# The parameter sets can vary:
# a) 'srccall nnn-n dstcall "string" symbol "R&" lat "ddmm.mmN" lon "dddmm.mmE" [co
# b) 'srccall nnn-n dstcall "string" raw "string"'
#
# The a) form flags on some of possible syntax errors in parameters.
# It will also create only "!" type messages. The dest parameter
# defaults to "APRS", but can be used to give other destinations.
# The via parameter can be used to add other keywords, like "NOGATE".
#
# Writing correct RAW format beacon message is very hard,
# which is evidenced by the frequency of bad syntax texts

```

```

# people so often put there...  If you can not be persuaded
# not to do it, then at least VERIFY the beacon result on
# web service like findu.com, or aprs.fi
#
#beacon                file /tmp/wxbeacon.txt
#beacon srccall NOCALL-3 raw "!6016.30NR02506.36E&aprx - an Rx-only iGate"
#beacon srccall NOCALL-3 raw "!6016.30NI02506.36E&aprx - an iGate"
#beacon srccall $mycall symbol "R&" lat "6016.30N" lon "02506.36E" \
                        comment "aprx - an Rx-only iGate"
#beacon srccall $mycall symbol "I&" lat "6016.30N" lon "02506.36E" \
                        comment "aprx iGate"
</beacon>

# ***** <telemetry> definition(s) follow *****
#
# The system will always send telemetry for all of its interfaces
# to APRSIS, but there is an option to define telemetry to be sent
# to radio channel by using following sections for each transmitter
# that is wanted to send out the telemetry.
#
# transmitter - callsign referring to <interface>
# via         - optional via-path, only 1 callsign!
# source      - one or more of <interface> callsigns for which
#               the telemetry transmission is wanted for
#
#<telemetry>
# transmitter $mycall
# via         TRACE1-1
# source      $mycall
#</telemetry>

# ***** <digipeater> definition(s) follow *****
#
# The digipeater definitions tell transmitters that receive
# AX.25 packets from possibly multiple sources, and then what
# to do on the AX.25 headers of those messages.
#
# There is one transmitter per digipeater -- and inversely, there
# can be at most one digipeater for each transmitter.
#
# In each digipeater there is at least one <source>, usually same
# as the transmitter.
#
#<digipeater>
# transmitter      $mycall
# #ratelimit       60 120      # default: average 60 packets/minute,
#                               # burst max 120 packets/minute
#
# <source>
# source           $mycall
# # ratelimit      60 120      # default: average 60 packets/minute,
# #                               # burst max 120 packets/minute
# # viscous-delay  0          # no viscous delay for RF->RF digipeat

```

```

# # ratelimit      120 # default: max 120 packets/minute
# </source>
#
# #<source>      # Adding APRSIS source makes this tx-igate
# # source       APRSIS
# # ratelimit    60 120 # default: average 60 packets/minute,
# #              # burst max 120 packets/minute
# # relay-type   third-party # Must define this for APRSIS source!
# # viscous-delay 5 # Recommendation: 5 seconds delay to give
# #              # RF delivery time make itself known.
# # filter       t/m # Tx-iGate only messages sent to me by APRSIS
# #</source>
#
#</digipeater>

```

GLOBAL MYCALL PARAMETER

In majority of usage models, system needs single configured callsign. This is set by using the **mycall** configuration option, and latter referred to in configurations as **\$mycall** parameter in place of callsigns.

APRSIS SECTION FOR APRSIS CONNECTIVITY

Settings in the **<aprsis>** section define connectivity with the APRS-IS network service.

Necessary option is *server*, and others are optional.

Available options are:

login *\$mycall*

The APRSIS network login. Defaults to the **mycall** configuration entry.

server *server-name 14850*

Define which APRS-IS is being connected to. Multiple definitions are used in round-robin style, if the connection with the previous one fails for some reason.

filter *'filter specs in quotes' # comments*

Set filter adjunct definitions on APRS-IS server. Multiple entries are catenated together in entry order, when connecting to the server.

LOGGING SECTION

The **<logging>** section defines miscellaneous file names and options for state tracking and logging use.

pidfile */var/run/aprx.pid*

The pidfile is UNIX way to tell that others that this program is running with given process-id number. This has compiled-in default value of: `pidfile /var/run/aprx.pid`

rflog */var/log/aprx/aprx-rf.log*

The *rflog* defines a rotatable file into which all RF-received packets are logged. There is no default.

aprxlog */var/log/aprx/aprx.log*

The *aprxlog* defines a rotatable file into which most important events on APRS-IS connection are logged, namely connects and disconnects. There is no default.

erlangfile */var/run/aprx.state*

The *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 straddle any exact minute. This file is around 0.7 MB per each interface talking APRS. If this file is not defined and can not be created, internal non-persistent in-memory storage will be used. Built-in default value is: `/var/run/aprx.state`

`erlang-loglevel` *NONE*

The `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.

`erlanglog` `/var/log/aprx/erlang.log`

The `erlanglog` defines a rotatable file into which erlang data is written in text form. There is no default.

`erlang-log1min`

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

INTERFACE SECTIONS FOR RADIO PORTS

The `<interface>` sections define connections to radio modems. Several different styles are available:

- Local serial ports in the machine (**device-serial** `/dev/ttyS0` *speed encapsulation*)
- Local USB serial ports in the machine (**device-serial** `/dev/ttyUSB0` *speed encapsulation*)
- Remote served serial ports over a TCP stream. Implemented to talk with Cisco AUX ports on "range 4000" (TCP STREAM, no TELNET escapes) (**tcp-device** `12.34.56.78 4001` *encapsulation*)
- Linux internal AX.25 network attached devices (**ax25-device** `CALLSIGN-1`) are only available when running on a Linux system. On a non-Linux system it connects to a null interface, never getting anything and can always sink everything.

The serial port name tells what kind of port is in question, and while port baud-rate (9600) and character settings (8n1) must always be set, they are ignored for the remote connection.

Following *speed* modes are available:

1200, *1800*, **2400**, **4800**, **9600**, **19200**, *38400*, *57600*,
115200, *230400*, *460800*, *500000*, *576000*

Likely available speeds are in bold, other supported values are listed in italics.

Following *encapsulation* modes are available:

TNC2 is capable only to monitor the packets reported by TNC2 type debug output, and Rx-iGate, but they are not acceptable as source for a `<digipeater>`.

DPRS is special mode for gateway from D-STAR D-PRS to APRS. This must always have a callsign definition for the gateway.

KISS Basic KISS encapsulation. No checksums. Will autodetect (sometimes) packets with SMACK or FLEXNET characteristics.

SMACK *Stuttgart Modified Amateurradio-CRC-KISS*, which runs CRC-16 checksum on KISS datastream much in the same way as HDLC has CCITT-CRC checksum on it.

FLEXNET

FLEXNET which runs a CRC checksum of its own polynomial on KISS datastream much in the same way as HDLC has CCITT-CRC checksum on it.

BPQCRC XOR "checksum" on dataframes. Also known as "XKISS", and "XORSUM". This detects single bit failure, but weakly any multibit failures. Extra 0x00 bytes have no effect on checksum, etc.

On `<kiss-subif` *tncid*`>` sub-options the parameter is *tncid*, which sets up KISS multiplexer parameter so that subsequent options applies only on designated KISS sub-port.

The `callsign` option sets port specific callsign when relaying to APRS-IS.

```

<interface>
  serial-device /dev/ttyUSB1 19200 8n1 KISS
  tx-ok         false          # receive only (default)
  callsign      OH2XYZ-R2      # KISS subif 0
  initstring    "...."        # initstring option
  timeout       900            # 900 seconds of no Rx
</interface>

<interface>
  serial-device /dev/ttyUSB1 19200 8n1 SMACK
  tx-ok         false          # receive only (default)
  callsign      OH2XYZ-R2      # KISS subif 0
  initstring    "...."        # initstring option
  timeout       900            # 900 seconds of no Rx
</interface>

<interface>
  serial-device /dev/ttyUSB2 19200 8n1 KISS
  initstring    "...."
  timeout       900            # 900 seconds of no Rx
  <kiss-subif 0>
    callsign    OH2XYZ-2
    tx-ok       true           # This is our transmitter
  </kiss-subif>
  <kiss-subif 1>
    callsign    OH2XYZ-R3      # This is receiver
    tx-ok       false         # receive only (default)
  </kiss-subif>
</interface>

<interface>
  tcp-device    172.168.1.1 4001 KISS
  tx-ok         false          # receive only (default)
  callsign      OH2XYZ-R4      # KISS subif 0
  initstring    "...."        # initstring option
  timeout       900            # 900 seconds of no Rx
</interface>

<interface>
  ax25-device   OH2XYZ-6        # Works only on Linux systems
  tx-ok         true           # This is also transmitter
</interface>

<interface> # RX-IGATE ONLY, NOT USABLE AS DIGIPEATER SOURCE
  serial-device /dev/ttyUSB1 19200 8n1 TNC2
  callsign      OH2XYZ-R6      # TNC2 has no sub-ports
  initstring    "...."        # initstring option
  timeout       900            # 900 seconds of no Rx
</interface>

```

BEACON DEFINITIONS

The beacons are defined using `<beacon>` configuration sections.

Because classical beacon definitions are highly error-prone, this program has a new way to define them:

- The new way to define beacons:

```
beacon symbol "R&" lat "6016.35N" lon "02506.36E" \
      comment "aprx - iGate"
```
- Semi-classical definition of raw APRS packet:

```
beacon raw "!6016.35NR02506.36E&aprx - iGate"
```
- Load beacon text from a file, path data is configurable:

```
beacon file /path/to/file
```

The fields and parameters:

interface	An <i>optional</i> "interface" parameter tells that this beacon shall be sent only to interface whose callsign is named. Default is to send to all interfaces that have "tx-ok true" setting.
type	An <i>optional</i> one character string parameter, with one of following contents: "!", "=", "/", "@", ";" and ")".
srcall	An <i>optional</i> "srcall" parameter tells callsign which is claimed as this particular beacon source. It must be valid AX.25 callsign in text format. When this "srcall" parameter is not given, value of "mycall" configuration entry is used.
dstcall	An <i>optional</i> "dstcall" parameter has built-in software version dependent value, but it can be used to define another value.
via	An <i>optional</i> "via" parameter defaults to nothing, but can be used to define additional "VIA" path tokens, for example: "WIDE1-1".
item	An <i>optional</i> "item" parameter is for defining a name for an item type APRS packet.
object	An <i>optional</i> "object" parameter is for defining a name for an object type APRS packet.
symbol	A <i>mandatory</i> "symbol" parameter is two character code, which for Rx-only iGate is pair: "R&"
lat	This <i>mandatory</i> parameter defines <i>latitude</i> coordinate (that is: north/south.) It is expected to be of format: "ddmm.mmN" where "dd" defines <i>two digits</i> of <i>degrees</i> of latitude, and "mm.mm" defines two digits + decimal dot + two digits of <i>minutes</i> of latitude. Then comes literal "N" or "S" indicating hemisphere.
lon	This <i>mandatory</i> parameter defines <i>longitude</i> coordinate (that is: east/west.) It is expected to be of format: "dddmm.mmE" where "ddd" defines <i>three digits</i> of <i>degrees</i> of longitude, and "mm.mm" defines two digits + decimal dot + two digits of <i>minutes</i> of longitude. Then comes literal "E" or "W" indicating hemisphere.
comment	This <i>optional</i> parameter defines commentary text tail on the beacon packet. If you need characters outside US-ASCII character set, use of UTF-8 encoded UNICODE character set is recommended.
raw	This <i>alternate</i> format defines whole APRS packet content in raw text format. <i>Currently this type of packets are not validated for syntax at all!</i>
file	This <i>alternative</i> way defines path to a file with single text line defining content of <i>raw</i> message data.

The type/symbol/lat/lon/comment-format supports only a few types of APRS packets. It splits input into small slices that are possible to validate in detail. (See "DEBUGGING SYSTEM" above.)

RF-TELEMETRY

The *aprx* system will always send telemetry for all of its interfaces to APRSIS, but there is an option to define telemetry to be sent to radio channel by using following sections for each transmitter that is wanted to send out the telemetry.

The parameters of **<telemetry>** configuration section are:

- transmitter** A mandatory callsign referring to an *interface*.
- via** An optional *via-path* parameter. Only 1 callsign!
- source** One or more of *interface* callsigns for which the telemetry transmission is made.

DIGIPEATER

The *aprx* is possible to configure as a AX.25 digipeater with APRS twists. This is done with **<digipeater>** configuration section and its subsections.

There can be at most one <digipeater> definition per each transmit capable interface in the system. On a system with multiple transmitters, this means there can be multiple digipeaters, each with different behaviour rules.

Minimalistic setup for a digipeater will be as follows:

```
<digipeater>
  transmitter      $mycall
  <source>
    source         $mycall
  </source>
</digipeater>
```

In minimalistic approach the system does digipeating of packets heard on the *\$mycall* interface back to same interface. Single requirement is that the **<interface>** block has *tx-ok true* setting on it.

In more complicated approaches it is possible to define multiple sources for packets:

- Multiple device ports.
- APRSIS pseudoport, which creates the Tx-iGate functionality.

<digipeater> options

Main-level **<digipeater>** options are:

- *transmitter* defines which interface the digipeater will output to.
- **<trace>** and **<wide>** sub-options are explained below.
- **<source>** sub-option is explained below.

<trace> and <wide> sub-options

The **<trace>** sub-option has priority over the **<wide>** sub-option, otherwise they are configured the same way.

The **<trace>** sub-option defines which AX.25 address contained keywords are treated with APRS "New-N paradigm" rules in a way where each processing node always marks its transmitter callsign on the transmitted AX.25 packet address header.

The **<wide>** sub-option defines which AX.25 address contained keywords are treated with APRS "New-N paradigm" rules in a way where processing node does not mark its transmitter callsign on the transmitted AX.25 packet address header.

Available parameters are:

keys A string of comma-separated set of string tokens:

```
keys "TRACE,WIDE"
```

Alternative form for this entry is:

```
keys "TRACE"
```

```
keys "WIDE"
```

maxdone Defines maximum number of redistribution hops that these keywords can have completed when reaching here. If accounting finds more done, the system will just drop the packet instead of digipeating it onwards.

maxreq Defines maximum number of redistribution hops that these keywords can define. If accounting finds more requested, the system will just drop the packet instead of digipeating it onwards.

<source> sub-options

Primary definer option is **source** which gives callsign of an <interface> from which the AX.25 packets are received for this <source> block.

Available **relay-type** modes on <source> definitions are:

digipeater Normal AX.25 digipeater behaviour with APRS New-N paradigm support. This is default mode.

directonly Digipeat only directly heard packets. Useful for systems that are designated as "fill-in". See also "viscous-delay".

third-party Special mode for Tx-iGate.

The **ratelimit** defines two parameters: *average* and *limit* number of packets sent in 60 seconds. Its definitions can be both in <digipeater> and in digipeater's <source> sections, and therefore you can limit each individual source to a max accepted rate as well as define separate rate limits for the transmitter.

The **viscous-delay** defines a number of seconds from 0 (default) maximum of 9 that the source will put the message on duplicate detector delay processing. All occurrences of same packet per duplicate detector during that time will be accounted on duplicate detection, and if at the end of the delay period there are more than one hit, the packet is discarded. Use delay of 0 seconds for normal digipeater, 5 seconds for a fill-in, or a Tx-iGate.

A javAPRSSrvr filter-adjunct style rules are possible with the **filter** options. When you want multiple filters, use multiple options with associated parameters:

```
filter t/m # APRS messaging type packets
filter a/la/lo/la/lo # APRS positional packets within this area
```

Also negative filters are possible (prefixed with minus character), which upon match cause rejection of the packet. Filters are evaluated in definition order, and first matching one will terminate the evaluation. When no filters are defined, everything is passed thru. When any filter is defined, only those matching non-negative filters are passed thru, and no default "pass everything else" behaviour exists.

Supported "adjunct filters" are following:

A/latN/lonW/latS/lonE

Area filter, defined as area enclosing within latS/latN and lonW/lonE. Latitude and longitude are entered as degrees and decimals.

B/call1/call2...

Budlist filter. Supports *-wildcards.

F/call/dist_km

Great-circle distance in kilometers from friend's coordinates. No wildcarding.
(*TODO: check that it really works!*)

O/object1/obj2...

Object name filter. Supports *-wildcards.

P/aa/bb/cc...

Prefix filter.

R/lat/lon/dist

Range filter. Latitude and longitude are in degrees and decimals. Distance is in kilometers. No wildcards.

S/pri/alt/over

Symbol filter

T/.../call/km

Type filter. Couple possible usages:

- t / c Everything except CWOP
- t / * /OH2RDY /50 Everything within 50 km of OH2RDY's last known position

Type code characters are:

- * An "all" wild-card.
- C** A CWOP.
- I** An ITEM.
- M** A MESSAGE.
- N** A NWS message.
- O** An OBJECT.
- Q** A QUERY.
- S** A STATUS response.
- T** A TELEMETRY packet or parameter message.
- U** A USERDEF message.
- W** A WX data packet

U/unproto1/unproto2...

Filters by value in destination address field, supports wildcard.

The **<trace>** and **<wide>** sub-options exist also within each **<source>**. Where such occur, the **<source>** specific **<trace>** sub-option trumps the definition on **<digipeater>** level, and same with **<wide>** sub-options. This allows things like overriding flooding control keywords on source basis, should such be necessary.

A set of **regex-filter** rules can be used to reject packets that are not of approved kind. Available syntax is:

- regex-filter source RE
 source address field
- regex-filter destination RE
 destination address field
- regex-filter via RE
 any via path field
- regex-filter data RE
 payload content

The regex-filter exists as ad-hoc method when all else fails.

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 1:1.025 (1 extra bit every 5 bytes)

NOTES: PROGRAM NAME

Initially this program had name *aprs-g*, which was too close to another (a less low-tech C++ approach) program had.

BUGS/WARTS

The *Erlang*-monitor mechanisms are of rudimentary quality, and can seriously underestimate the channel occupancy by ignoring pre- and postamble transmissions, which can be as high as 50 centiseconds for preamble, and 20 centiseconds for postamble! When entire packet takes 50 centiseconds, such preamble alone doubles channel occupancy. A 6pack protocol on serial link (instead of KISS) could inform receiver better on carrier presense times, however even that underestimates RF power presense (RSSI) signal. (6pack is not supported.)

On serial lines supports really only 8n1 mode, not at all like: 7e1. On the other hand, there really is no sensible usage for anything but 8n1...

SEE ALSO

Couple web sites: <http://www.aprs-is.net/>, <http://www.aprs2.net/>, <http://wiki.ham.fi/Aprx.en>, <http://ham.zmailer.org/oh2mqk/aprx/aprx-manual.pdf>

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. Fall and winter 2009-2010 saw appearance of digipeater functionality.

Principal contributors and test users include: *Pentti Gronlund*, *OH3BK*, *Reijo Hakala*, *OH1GWK*. Debian packaging by *Kimmo Jukarinen*, *OH3GNU*. Testing of SMACK variant of KISS by *Patrick Hertenstein*, *DL1GHN*.