Aprx 1.96 Manual

Table of Contents

1 What is APRX?..... 2
2 Configuration Examples:..... 3
2 Minimal configuration of Rx-iGate..... 3
3 Minimal configuration APRS Digipeater..... 4
4 Combined APRS Digipeater and Rx-iGate..... 5
5 Doing Transmit-iGate..... 6
6 Digipeater and Transmit-iGate..... 7
7 A Fill-in Digipeater..... 8
8 Using multiple radios..... 9
9 A Digipeater with Multiple Radios..... 10
10 A Bi-directional Cross-band Digipeater..... 11
11 Limited Service Area Digipeater..... 12
3 Configuration in details..... 13
1 The “mycall” parameter..... 13
2 The “<aprsis>” section..... 13
3 The “<logging>” section..... 14
4 The “<interface>” sections..... 14
5 The “<beacon>” sections..... 15
6 The “<digipeater>” sections..... 15

By Matti Aarnio, OH2MQK, 2009

Version 0.02

1 What is APRX?

The Aprx program is for amateur radio APRS™ networking.

The Aprx program can do job of at least two separate programs:

1. APRS iGate
2. APRS Digipeater

The program has ability to sit on a limited memory system, it is routinely run on OpenWRT machines with 8 MB of RAM and Linux kernel. 128 MB RAM small PC is quite enough for this program with 100 MB ramdisk, web-server, etc.

The program is happy to run on any POSIX compatible platform, a number of UNIXes have been verified to work, Windows needs some support code to work.

On Linux platform the system supports also Linux kernel AX.25 devices.

This program will also report telemetry statistics on every interface it has. This can be used to estimate radio channel loading, and in general to monitor system and network health.

2 Configuration Examples:

2 Minimal configuration of Rx-iGate

To make a receive-only iGate, one needs simply to configure:

1. mycall parameter
2. APRSIS network connection
3. Interface for the radio

```
mycall  N0CALL-1

<aprsis>
     server    rotate.aprs.net    14580
</aprsis>

<interface>
     serial-device   /dev/ttyUSB0  19200 8n1    KISS
</interface>
```

You need to fix the "*N0CALL-1*" callsign with whatever you want it to report receiving packets with (it must be unique in global APRSIS network!)

You will also need to fix the interface device with your serial port, network TCP stream server, or Linux AX.25 device. Details further below.

In usual case of single radio TNC interface, this is all that a receive-only APRS iGate will need.

3 Minimal configuration APRS Digipeater

To make a single interface digipeater, you will need:

1. mycall parameter
2. <interface> definition
3. <digipeater> definition

Additional bits over the Rx-iGate are highlighted below:

```
mycall  N0CALL-1

<interface>
     serial-device /dev/ttyUSB0  19200 8n1    KISS
     tx-ok         true
</interface>

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

The interface must be configured for transmit mode (default mode is receive-only)

Defining a digipeater is fairly simple as shown.

4 Combined APRS Digipeater and Rx-iGate

Constructing a combined APRS Digipeater and Rx-iGate means combining previously shown configurations:

```
mycall  N0CALL-1

<aprsis>
     server    rotate.aprs.net    14580
</aprsis>

<interface>
     serial-device /dev/ttyUSB0  19200 8n1    KISS
     tx-ok         true
</interface>

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

5 Doing Transmit-iGate

At the time of the writing, doing Tx-iGate is still lacking some bits necessary for full function.

```
mycall  N0CALL-1

<aprsis>
     server    rotate.aprs.net    14580
</aprsis>

<interface>
     serial-device /dev/ttyUSB0  19200 8n1    KISS
     tx-ok         true
</interface>

<digipeater>
     transmit $mycall
     <source>
          source         APRSIS
          digi-mode      3rd-party
          filter         t/m
     </source>
</digipeater>
```

This is Rx/Tx-iGate in a form that Aprx version 1.96 can do.

It does omit couple important bits on controlling transmission from APRSIS to radio, and thus a kludge definition of filtering “pass only type M packets” (APRS Messages).

6 Digipeater and Transmit-iGate

This is fairly simple extension, but shows important aspect of Aprx's <digipeater> definitions, namely that there can be multiple sources!

```
mycall  N0CALL-1

<aprsis>
     server    rotate.aprs.net    14580
</aprsis>

<interface>
     serial-device /dev/ttyUSB0  19200 8n1    KISS
     tx-ok         true
</interface>

<digipeater>
     transmit $mycall
     <source>
          source  $mycall
     </source>
     <source>
          source         APRSIS
          digi-mode      3rd-party
          filter         t/m
     </source>
</digipeater>
```

Using both the radio port, and APRSIS as sources makes this combined Tx-iGate, and digipeater.

7 A Fill-in Digipeater

Classically a fill-in digipeater means a system that digipeats heard packet only when it hears it as from first transmission. Usually implemented as “consider WIDE1-1 as your alias”, but the Aprx has more profound understanding of digipeating.

```
<digipeater>
     transmit $mycall
     <source>
          source  $mycall
          relay-type directonly
     </source>
</digipeater>
```

With Aprx you can add condition: *and only if nobody else digipeats it within 5 seconds.*

```
<digipeater>
     transmit $mycall
     <source>
          source        $mycall
          relay-type    directonly
          viscous-delay 5
     </source>
</digipeater>
```

8 Using multiple radios

There is no fixed limit on number of radio interfaces that you can use, however of them only one can use the default callsign from “$mycall” macro:

```
mycall  N0CALL-1

<interface>
     # callsign  $mycall
     serial-device /dev/ttyUSB0  19200 8n1    KISS
     tx-ok         true
</interface>

<interface>
     callsign  N0CALL-R2
     serial-device /dev/ttyUSB1  19200 8n1    KISS
</interface>
```

Supported interface devices include:

1. On Linux: Any AX.25 network attached devices
2. On any POSIX system: any serial ports available through “tty” interfaces
3. Remote network terminal server serial ports over TCP/IP networking

On serial ports, following protocols can be used:

1. Plain basic **KISS**: Binary transparent, decently quick.
2. **SMACK**: A CRC16 two-byte CRC checksum on serial port KISS communication. Recommended mode for KISS operation.
3. XOR checksum on KISS: So called “**BPQCRC**”. Not recommended because it is unable to really detect data that has broken during serial port transmission. Slightly better than plain basic KISS.
4. **TNC2** monitoring format, receive only, often transmitted bytes outside printable ASCII range of characters are replaced with space, or with a dot. **Not recommended to be used!**

9 A Digipeater with Multiple Radios

Extending on previous multiple interface example, here those multiple interfaces are used on a digipeater. Transmitter interface is at “$mycall” label, others are receive only:

```
<digipeater>
     transmit $mycall
     <source>
          source  $mycall
     </source>
     <source>
          source  N0CALL-2
     </source>
     <source>
          source  N0CALL-3
     </source>
</digipeater>
```

Adding there a source of APRSIS will merge in Tx-iGate function, as shown before.

It is trivial to make a multiple receiver, single transmitter APRS Digipeater with this.

10 A Bi-directional Cross-band Digipeater

Presuming having transmit capable radio <interface>s on two different bands, one can construct a bi-directional digipeater by defining two <digipeater> sections.

```
<digipeater>
     transmit N0CALL-1
     <source>
          source  N0CALL-1
     </source>
     <source>
          source  N0CALL-2
     </source>
</digipeater>

<digipeater>
     transmit N0CALL-2
     <source>
          source  N0CALL-1
     </source>
     <source>
          source  N0CALL-2
     </source>
</digipeater>
```

Now both transmitters will digipeat messages heard from either radio.

11 Limited Service Area Digipeater

A digipeater that will relay only packets from positions in a limited service area can be done by using filtering rules

```
<digipeater>
     transmit N0CALL-1
     <source>
          source  N0CALL-1
          filter    t/m       # All messages (positionless)
          filter    a/60/23/59/25.20
          filter    a/60.25/25.19/59/27
     </source>
</digipeater>
```

This example is taken from a limited service area digipeater on a very high tower in Helsinki, Finland. The coordinates cover Gulf of Finland, and northern Estonia. Especially it was not wanted to relay traffic from land-areas, but give excellent coverage to sail yacts.

3 Configuration in details

The Aprx configuration file uses sectioning style familiar from Apache HTTPD.

These sections are:

1. mycall
2. <aprsis>
3. <logging>
4. <interface>
5. <digipeater>

1 The “mycall” parameter

The “mycall” is just one global definition to help default configuration to be minimalistic by not needing copying your callsign all over the place in the usual case of single radio interface set-up.

2 The “<aprsis>” section

The <aprsis> section defines communication parameters towards the APRSIS network.

The only required parameter is the server definition:

```
    server    rotate.aprs.net   14580
```

where the port-number defaults to 14580, and can be omitted.

Additional optional parameters are:

- login *callsign*
- heartbeat-timeout *interval-definition*
- filter *adjunct-filter-entry*

The *login* defaults to global *$mycall*, thus it is not necessary to define.

Adding "*heartbeat-timeout 2m*" will detect failure to communicate with APRSIS a bit quicker than without it. The current generation of APRSIS servers writes a heartbeat message every 20 seconds, and a two minute timeout on their waiting is more than enough.

The "*filter ...*" entries are concatenated, and given to APRSIS server as adjunct filter definitions. For more information about their syntax, see:

http://www.aprs-is.net/javAPRSFilter.aspx

3 The "<logging>" section

The Aprx can log every kind of event happening, mainly you will be interested in *rflog*, and *aprxlog*.

There is also a possibility to store statistics gathering memory segment on a filesystem backing store, so that it can persist over restart of the Aprx process. This is possible even on a small embedded machine (like OpenWRT), where statistics "file" resides on a ram-disk. This way you can alter configurations and restart the process, while still continuing with previous statistics dataset. Without the backing store this will cause at most 20 minute drop-off of statistics telemetry data.

Configuration options are:

- aprxlog *filename*
- rflog *filename*
- pidfile *filename*
- erlangfile *filename*
- erlang-loglevel *loglevel*
- erlanglog *filename*

Commonly you want setting erlangfile, aprxlog, and rflog entries.

4 The "<interface>" sections

The <interface> sections define radio interfaces that the Aprx communicates with.

There are three basic interface device types:

1. Linux AX.25 devices (ax25-device)

2. Generic POSIX serial ports (serial-device)
3. Remote network serial ports (tcp-device)

The serial port devices can be reading TNC2 style monitoring messages (and be unable to transmit anything), or communicate with a few variations of KISS protocol (and transmit). On KISS protocols one can use device multiplexing, although cases needing polling for reception do not work.

On Linux systems the AX.25 network devices are also available.

Each interface needs a unique callsign and to help the most common case of single radio interface, it defaults to one defined with *mycall* entry. The interface callsigns need not to be proper AX.25 callsigns on receive-only interfaces, meaning that a *N0CALL-R0 .. R9 .. RA .. RZ* are fine examples of two character suffixes usable on receivers.

MORE TO BE WRITTEN

5 The “<beacon>” sections

You can define multiple <beacon> sections each defining multiple beacon entries.

Beacons can be sent to radio only, to aprsis only, or to both. Default is to both.

You can configure beacons as literals, and also to load beacon content from a file at each time it is to be transmitted. That latter allows external program, like weather probes, to feed in an APRS weather data packet without it needing to communicate with Aprx via any special protocols, nor make AX.25 frames itself.

MORE TO BE WRITTEN

6 The “<digipeater>” sections

With Aprx you can define multiple <digipeater> sections, each to their unique transmitter.

At each <digipeater> section you can define multiple <source> sub-sections so that traffic from multiple sources are sent out with single transmitter.

This allows defining different behaviour rules per each transmission path.

The Aprx implements duplicate checking per each transmitter, and if same message is received via multiple (diversity) receivers, only one copy will be transmitted.

MORE TO BE WRITTEN