

# APRX Software Requirement Specification

## Table of Contents

1	APRX Software Requirement Specification.....	2
1.1	Purpose:.....	2
1.2	Usage Environments:.....	3
1.3	AX.25 details for radio channel transmission.....	4
1.4	D-STAR <-> APRS.....	5
2	Treatment rules:.....	6
2.1	Basic IGate rules:.....	6
2.2	Low-Level Transmission Rules:.....	8
2.3	Low-Level Receiving Rules:.....	9
2.4	Additional Tx-IGate rules:.....	10
2.5	D-STAR/DPRS to APRS gating rules.....	11
2.6	Digipeater Rules.....	12
2.6.1	APRS (Control=0x03,PID=0xF0) digipeat.....	12
2.6.2	Other UI (Control=0x03, PID != 0xF0) digipeats.....	13
2.6.3	Other (Control != 0x03) digipeats.....	13
2.6.4	Viscous Digipeating.....	14
2.7	Duplicate Detector.....	15
2.7.1	Control=0x03,PID=0xF0: APRS.....	15
2.7.2	Control=0x03,PID!=0xF0: Others.....	15
2.7.3	Control != 0x03: Others.....	15
2.8	Radio Interface Statistics Telemetry.....	16
2.9	Individual Call-Signs for Each Receiver, or Not?.....	17
2.10	Beaconing.....	18
2.10.1	Radio Beaconing.....	18
2.10.2	Network beaconing.....	18
3	Configuration Language.....	19
3.1	APRSIS Interface Definition.....	20
3.2	Radio Interface Definitions.....	20
3.3	Digipeating Definitions.....	21
3.3.1	<trace>.....	23
3.3.2	<wide>.....	23
3.3.3	<trace>/<wide> Default Rules.....	24
3.4	NetBeacon definitions.....	25
3.5	RfBeacon definitions.....	26

## 5 **1 APRX Software Requirement Specification**

6 This is *Requirement Specification* for a software serving in Amateur Radio APRS service.

7 Reader is assumed to be proficient with used terminology, and they are not usually  
8 explained here.

### 9 **1.1 Purpose:**

10 This describes algorithmic, IO-, and environmental requirements for a software doing any  
11 combination of following four tasks related to APRS service:

- 12 1. Listen on messages with a radio, and pass them to APRSIS network service
- 13 2. Listen on messages with a radio, and selectively re-send them on radio
- 14 3. Listen on messages with a radio, and selectively re-send them on radios on other  
15 frequencies
- 16 4. Receive messages from APRSIS network, and after selective filtering, send some of  
17 them on radio

18

19 Existing *aprx* software implements Receive-Only (Rx) IGate functionality, and the purpose  
20 of this paper is to map new things that it will need for extending functionality further.

21

22

## 23 1.2 Usage Environments:

24 The *aprx* software can be used in several kinds of environments to handle multiple tasks  
25 associated with local APRS network infrastructure tasks.

26 On following one should remember that amateur radio **transmitters** need a specially  
27 licensed owner/operator or a license themselves, but receivers do not need such in usual  
28 case:

- 29 1. License-free Receive-Only (RX) IGate, to add more “ears” to hear packets, and to  
30 pipe them to APRSIS. (Owner/operator has a license, but a receiver does not need  
31 special *transmitter license*.)
- 32 2. Licensed bidirectional IGate, selectively passing messages from radio channels to  
33 APRSIS, and from APRSIS to radio channels, but not repeating packets heard on a  
34 radio channel back to a radio channel.
- 35 3. Licensed bidirectional IGate plus selectively re-sending of packets heard on radio  
36 channels back to radio channels ( = digipeater )
- 37 4. Licensed system for selectively re-sending of packets heard on radio channels back  
38 to other radio channels ( = digipeater ), and this without bidirectional IGate service.
- 39 5. Licensed system for selectively re-sending of packets heard on radio channels back  
40 to radio channels ( = digipeater ), and doing with with “receive only” IGate, so  
41 passing information heard on radio channel to APRSIS, and not the other way at all.

42

43 In more common case, there is single radio and single TNC attached to digipeating (re-  
44 sending), in more challenging cases there are multiple receivers all around, and very few  
45 transmitters. Truly challenging systems operate on multiple radio channels. As single-  
46 TNC and single-radio systems are just simple special cases of these complex systems,  
47 and for the purpose of this software requirements we consider the complex ones:

- 48 1. 3 different frequencies in use, traffic is being relayed in between them, and the  
49 APRSIS network.
- 50 2. On each frequency there are multiple receivers, and one well placed transmitter.
- 51 3. Relaying from one frequency to other frequency may end up having different rules,  
52 than when re-sending on same frequency: Incoming packet retains traced paths,  
53 and gets WIDEn-N/TRACEn-N requests replaced with whatever sysop wants.

54

### 55 1.3 AX.25 details for radio channel transmission

56 Used frame structure is per AX.25 v2.0 specification, not AX.25 v2.2.

- 57 • Source call-signs are always identifying message sender
- 58 • Destination call-signs indicate target group, most commonly "APRS", but also
- 59 message originator specific software identifiers are used.
- 60 • Digipeater fields use preferably "New-N paradigm" style "WIDEn-N" or "TRACEn-N"
- 61 values on frame origination, and the digipeaters will then place their call-signs on
- 62 the via-field as trace information:
  - 63 • Original: N0CALL-9>APRS,WIDE2-2
  - 64 • After first digipeat either:
    - 65 • N0CALL-9>APRS,WIDE2-1
    - 66 • N0CALL-9>APRS,N1DIGI\*,WIDE2-1
  - 67 • After second digipeat any of:
    - 68 • N0CALL-9>APRS,WIDE2\*
    - 69 • N0CALL-9>APRS,N1DIGI\*,WIDE2\*
    - 70 • N0CALL-9>APRS,N1DIGI\*,N2DIGI\*,WIDE2\*
  - 71 • ('\*' means that H-bit on digipeater field's SSID byte has been set, and that
  - 72 other digipeaters must ignore those fields.)
- 73 • Also several older token schemes in the via-fields are still recognized

74 Important differences on address field bit treatments:

- 75 • Three topmost bits on Source and Destination address fields SSID bytes are never
- 76 validated.
  - 77 • Most common values seen on radio transmissions are based on AX.25 v2.2
  - 78 chapter 6.1.2 "Command" combinations: 011 for source, and 111 for
  - 79 destination.
  - 80 • *In practice all 64 combinations of these 6 bits are apparent in radio networks.*
  - 81 *Receiver really must ignore them.*
- 82 • VIA address fields (digipeater fields) can be up to 8, AX.25 v2.2 changed earlier
- 83 specification from 8 to 2 via fields, and thus AX.25 v2.2 is ignored here.
- 84 • The topmost bit on SSID bytes of VIA address fields is "H" alias "Has been
- 85 digipeated", and the two reserved ones should be "11", but only "H"-bit is used, and
- 86 everybody ignores those two reserved bits!

87 After the AX.25 address fields, used control byte is always 0x03 (UI frame,) and used PID

88 byte is 0xF0 for APRS.

89 This system does process all type of AX.25 frames at least on digipeater, including UI

90 TCP/IP, and AX.25 CONS.

91

92 **1.4 D-STAR <-> APRS**

93 TO BE WRITTEN

- 94 • What is the physical and link-level protocol interface to D-STAR radio?
- 95 • What is the D-STAR's DPRS protocol?
- 96 • Existing D-STAR/DPRS to APRS gateways pass positional packets as 3<sup>rd</sup>-party
- 97 frames, and are one of few 3<sup>rd</sup>-party types that are IGated to APRSIS as is.

98

## 99 2 Treatment rules:

100 Generally: All receivers report what they hear straight to APRSIS, after small amount of  
101 filtering of junk messages, and things which explicitly state that they should not be sent to  
102 APRSIS.

### 103 2.1 Basic IGate rules:

104 General rules for these receiving filters are described here:

105 <http://www.aprs-is.net/IGateDetails.aspx>  
106

107 Gate all packets heard on RF to the Internet (Rx-IGate) EXCEPT

- 108 1. 3<sup>rd</sup> party packets (data type '}' ) should have all before and including the data  
109 type stripped and then the packet should be processed again starting with  
110 step 1 again. There are cases like D-STAR gateway to APRS of D-STAR  
111 associated operator (radio) positions.  
112 2. generic queries (data type '?' ).  
113 3. packets with TCPIP, TCPXX, NOGATE, or RFONLY in the header, especially  
114 in those opened up from a 3<sup>rd</sup> party packets.

115  
116 Gate message packets and associated posits to RF (Tx-IGate) if

- 117 1. the receiving station has been heard within range within a predefined time  
118 period (range defined as digi hops, distance, or both).
- 119 2. the sending station has not been heard via RF within a predefined time  
120 period (packets gated from the Internet by other stations are excluded from  
121 this test).
- 122 3. the sending station does not have TCPXX, NOGATE, or RFONLY in the  
123 header.
- 124 4. the receiving station has not been heard via the Internet within a predefined  
125 time period.

126 A station is said to be heard via the Internet if packets from the station contain  
127 TCPIP\* or TCPXX\* in the header or if gated (3<sup>rd</sup> party) packets are seen on RF  
128 gated by the station and containing TCPIP or TCPXX in the 3<sup>rd</sup> party header (in  
129 other words, the station is seen on RF as being an IGate).

130 Gate all packets to RF based on criteria set by the sysop (such as call-sign, object  
131 name, etc.).  
132

133 Rx-IGate to APRSIS can use duplicate detection, and refuse to repeat same packet over  
134 and over again to APRSIS network.

135 With more advanced looking inside frames to be relayed, both the digipeater and Tx-IGate  
136 can use filtering rules, like “packet reports a position that is within my service area.”  
137

138

139 From multiple receivers + single (or fewer) transmitter(s) follows, than when a more usual  
140 system does not hear what it sent out itself, this one will hear, and its receivers must have  
141 a way to ignore a frame it sent out itself a moment ago.

142 Without explicit “ignore what I just sent” filtering, an APRS packet will get reported twice to  
143 APRSIS:

144  $rx \Rightarrow \text{igate-to-aprsis} + \text{digi} \Rightarrow tx \Rightarrow rx \Rightarrow \text{igate-to-aprsis} + \text{digi}$  (dupe filter stops)

145 Digipeating will use common packet duplication testing to sent similar frame out only once  
146 per given time interval (normally 30 seconds.)

147

148 An RF/Analog way to handle the “master-TX spoke this one, I will ignore it” could be use of  
149 audio subtones (American Motorola lingo: PL tone, otherwise known as CTCSS.)  
150 Digipeater transmitters have unique CTCSS subtone at each, and all receivers have  
151 subtone decoders. When they detect same subtone as their master has, they mute the  
152 receiver to data demodulator audio signal.

153

154 A third way would be to recognize their master transmitter call-sign in AX.25 VIA path, or at  
155 FROM field, which presumes that the master transmitters will do TRACE mode adding of  
156 themselves on digipeated paths.

157

## 158 2.2 Low-Level Transmission Rules:

159 These rules control repeated transmissions of data that was sent a moment ago, and other  
160 basic transmitter control issues, like persistence. In particular the persistence is fine  
161 example of how to efficiently use radio channel, by sending multiple small frames in quick  
162 succession with same preamble and then be silent for longer time.

163 For each transmitter:

- 164 1. A candidate packet is subjected to a number of filters, and if approved for it, the  
165 packet will be put on duplicate packet detection database (one for each transmitter.)  
166 See Digipeater Rules, below. System counts the number of hits on the packet,  
167 first arrival is count=1.
- 168 2. Because the system will hear the packets it sends out itself, there must be a global  
169 expiring storage for recently sent packets, which the receivers can then compare  
170 against. (Around 100 packets of 80-120 bytes each.) This storage gets a full copy  
171 of packet being sent out – a full AX.25 frame, and it is not same things as duplicate  
172 detector!

173 Also, transmitters should be kept in limited leash: Transmission queue is less than T  
174 seconds ( < 5 ? ), which needs some smart scheduling coding, when link from computer to  
175 TNC is considerably faster.

176 Original KISS interface is defined as “best effort”: if TNC is busy while host sends a frame,  
177 the frame may be discarded, and “upper layers” will resend. In APRS Digipeating, the  
178 upper layer sends the packet once, and then declares circa 30 second moratorium on  
179 packets with same payload.

180



## 181 2.3 Low-Level Receiving Rules:

- 182 1. Received AX.25 packet is compared against “my freshly sent packets” storage, and  
183 matched ones are dropped. (Case of one/few transmitters, and multiple receivers  
184 hearing them.)
  - 185 2. Received packet is validated against AX.25 basic structure, invalid ones are  
186 dropped.
    - 187 1. This means that AX.25 address headers are validated per their rules (including  
188 ignored bit sub-groups in the rules).
  - 189 3. Received APRS packet is parsed for APRS meaning [type, position]/[unknown] for  
190 optional latter area filtering. Received *other* PID packets are not parsed.
  - 191 4. Received APRS packet is validated against Rx-IGate rules, forbidden ones are not  
192 Rx-IGated (like when a VIA-field contains invalid data.) Received *other* PID UI-  
193 packets are not validated.
  - 194 5. Packet may be rejected for Rx-IGate, but it may still be valid for digipeating!  
195 For example an APRS 3<sup>rd</sup> party frame is OK to digipeat, but not to Rx-IGate to  
196 APRSIS! Also some D-STAR to APRS gateways output 3<sup>rd</sup> party frames, while the  
197 original frame is quite close to an APRS frame.
- 198 Divide packet rejection filters to common, and destination specific ones.
- 199

## 200 **2.4 Additional Tx-IGate rules:**

201 The Tx-IGate can have additional rules for control:

202 1. Multiple filters look inside the message, and can enforce a rule of “repeat only  
203 packets within my service area,” or to “limit passing message responses only to  
204 destinations within my service area”. Filter input syntax per javAPRSSrvr's adjunct  
205 filters.

206 2. Basic gate filtering rules:

- 207 1. the receiving station has been heard within range within a predefined time  
208 period (range defined as digi hops, distance, or both).  
209 2. the sending station has not been heard via RF within a predefined time period  
210 (packets gated from the Internet by other stations are excluded from this test).  
211 3. the sending station does not have TCPXX, NOGATE, or RFONLY in the header.  
212 4. the receiving station has not been heard via the Internet within a predefined time  
213 period.

214 A station is said to be heard via the Internet if packets from the station contain  
215 TCPIP\* or TCPXX\* in the header or if gated (3rd-party) packets are seen on RF  
216 gated by the station and containing TCPIP or TCPXX in the 3rd-party header (in  
217 other words, the station is seen on RF as being an IGate).

218

219 **2.5 D-STAR/DPRS to APRS gating rules**

220 TO BE WRITTEN

221

## 2.6 Digipeater Rules

### 2.6.1 APRS (Control=0x03,PID=0xF0) digipeat

Digipeater will do following for each transmitter for each data source per transmitter:

1. Feed candidate packet to duplicate detector. (Details further below.)
  1. *Viscous Digipeater* delay happens here (see below.)
  2. If the packet (after possible viscousness delay) has hit count over 1, drop it.
2. Check VIA fields for this transmitter's call-sign. If match is found, and its H-bit is not set, mark all VIA field's H-bit set up to and including the call-sign, subject it to duplicate comparisons, and digipeat without further WIDE/TRACE token processing. If the H-bit was set, drop the frame.
3. Optionally multiple source specific filters look inside the packets, and can enforce a rule of "repeat only packets within my service area."
4. Hop-Count filtering:
  1. Count number of hops the message has so far done, and figure out the number of hops the message has been requested to do (e.g. "OH2XYZ-1>APRS,OH2RDU\*,WIDE7-5: ..." will report that there was request of 7 hops, so far 2 have been executed – one is shown on trace path.)
  2. If either request count or executed count are over any of configured limits, the packet is dropped.
5. FIXME: Cross frequency digipeating? Treat much like Tx-IGate?
 

Relaying from one frequency to other frequency may end up having different rules, than when re-sending on same frequency: Incoming packet retains traced paths, and gets WIDEn-N/TRACEn-N requests replaced with whatever sysop wants.
6. Cross band relaying may need to add both an indication of "received on 2m", and transmitter identifier: "sent on 6m":
 

"OH2XYZ-1>APRS,RX2M\*,OH2RDK-6\*,WIDE3-2: ..."

This "source indication token" may not have anything to do with real receiver identifier, which is always shown on packets passed to APRSIS.
7. WIDEn-N/TRACEn-N treatment rules: Have configured sets of keywords for both modes. Test TRACE set first, and by default have there keywords: WIDE,TRACE.
  1. Check if first non-digipeated VIA field has this transmitter call-sign, and digipeat if it is found.
  2. Check if first non-digipeated VIA field has any of this transmitters aliases. If match is found, substitute there transmitter call-sign, and mark H-bit.

The MIC-e has a weird way to define same thing as normal packets do with

SRCCALL-n>DEST,WIDE2-2: ...

The MIC-e way (on specification, practically nobody implements it) is:

SRCCALL-n>DEST-2: ...

## 2.6.2 Other UI (Control=0x03, PID != 0xF0) digipeats

Optionally the Digipeater functionality will handle also types of UI frames, than APRS.

Support for this is optional needing special configuration enable entries.

Digipeater will do following for each transmitter for each data source per transmitter:

1. Optionally check PID from “these I digipeat” -list. Drop on non-match.
2. If the frame has no VIA fields with H-bit clear, feed the packet to duplicate checker, and drop it afterwards.
3. Check VIA fields for this transmitter's call-sign. If match is found, and its H-bit is not set, mark all VIA field's H-bit set up to and including the call-sign, subject it to possible duplicate comparisons, and digipeat without further WIDE/TRACE token processing. If the H-bit was set, drop the frame.
4. Per PID value:
  1. Optional WIDE/TRACE/RELAY processing
  2. Optionally per PID feed candidate packet to duplicate detector. (Similar to APRS case?)
5. Optional Hop-Count Filtering? (Similar to APRS case?)
6. Treat Cross-Frequency Digipeating as anything special? (Compare with APRS case above.)

## 2.6.3 Other (Control != 0x03) digipeats

Optionally the Digipeater functionality will handle also types of frames, than UI frames.

Support for this is optional needing special configuration enable entry.

Digipeater will do following for each transmitter for each data source per transmitter:

1. Explicit transmitter call-sign digipeat handles digipeat of all kinds of AX.25 frames. Comparison is done only on first VIA field without H-bit.
2. There is no duplicate detection.
3. No other type special digipeat is handled. (That is, NET/ROM, ROSE which do hop-by-hop retry and retransmission.)

## 2.6.4 Viscous Digipeating

*Viscous Digipeating* is defined to mean a digipeater that puts heard packets on a “probation delay FIFO” , where they sit for a fixed time delay, and after that delay the system checks to see if same packet (comparison by dupe-check algorithm) has been heard from some other digipeater in the meantime.

The Viscous Digipeaters are fill-in/car/backup type digipeater systems that repeat heard packets **only if somebody else has not done it already**.

The time delay is fixed number of seconds, which is configured on the system, and should be rather small (5-8 seconds), as duplicate detection algorithm uses storage lifetime of about 30 seconds, and digipeaters must **not** cause too long delays.

**With some application space combinatoric analysis, following rules emerged:**

Packets arriving from non-viscous sources trump those waiting in viscous queue. First one arriving will be transmitted, unless the viscous queue has no longer this packet (but it was there.)

- delayed\_seen > 0, seen == 1, pbuf == NULL -> drop this
- delayed\_seen > 0, seen == 1, pbuf != NULL -> clean pbuf, transmit this
- delayed\_seen == 0, seen == 1 -> transmit this

Subsequent packets arriving from non-viscous sources are dropped as duplicates ( seen > 1 -> drop this )

Packets arriving from any viscous source are dropped, if there already was some direct delivery packet ( seen > 0 -> drop )

First packet arriving from any viscous source is put on viscous queue, unless there was non-viscous packet previously:

- delayed\_seen == 1, seen > 0 -> drop this
- delayed\_seen == 1, seen == 0 -> put this on viscous queue

Then among viscous sources:

- "Transmitter" kind source: an <interface> which is same as that of <digipeater>'s transmitter <interface>.
- "Elsewhere" kind source: an <interface> which is some other than that of transmitter's, but has viscous-delay > 0

Account the number of viscous sourced packets sourced from "transmitter"

if (source\_is\_transmitter)

seen\_on\_transmitter += 1;

For second and subsequent viscous sourced packets, if any of observed packets came from transmitter (seen\_on\_transmitter > 0), then drop current packet, and clear possible viscous queued pbuf.

## 329 2.7 Duplicate Detector

330 Duplicate detector has two modes, depending on PID value of the frame.

331 All packets selected to go to some transmitter are fed on the duplicate detector of that  
332 transmitter, and found matches increase count of seen instances of that packet.

333

### 334 2.7.1 Control=0x03,PID=0xF0: APRS

335 Normal digipeater duplicate packet detection compares message source (with SSID),  
336 destination (without SSID!), and payload data against other packets in self-expiring  
337 storage called “duplicate detector”. Lifetime of this storage is commonly considered to be  
338 30 seconds.

339 APRS packets should not contain CR not LF characters, and they should not have extra  
340 trailing spaces, but software bugs in some systems put those in, The packet being  
341 compared at Duplicate Detector will be terminated at first found CR or LF in the packet,  
342 and if there is a space character(s) preceding the line end, also those are ignored when  
343 calculating duplication match. **However: All received payload data is sent as is without  
344 modifying it in any way!** (Some TNC:s have added one or two extra space characters  
345 on packets they digipeat...)

346 The “destination without SSID” rule comes from MIC-e specification, where a destination  
347 WIDE uses SSID to denote number of distribution hops. Hardly anybody implements it.

348

### 349 2.7.2 Control=0x03,PID!=0xF0: Others

350 Other type digipeater duplicate packet detection compares message source, and  
351 destination (both with SSID!), and payload data against other packets in self-expiring  
352 storage called “duplicate detector”. Lifetime of this storage is commonly considered to be  
353 30 seconds.

354 For PID != 0xF0 the duplicate detection compares whole payload.

355

### 356 2.7.3 Control != 0x03: Others

357 No duplicate detection for other types of AX.25 frames.

358

## 359 2.8 Radio Interface Statistics Telemetry

360 Current *aprx* software offers telemetry data on radio interfaces. It consists of following four  
361 things. Telemetry is reported to APRS-IS every 10 minutes:

- 362 1. Channel occupancy average in Erlangs over 1 minute interval, and presented as  
363 busiest 1 minute within the report interval. Where real measure of carrier presence  
364 on radio channel is not available, the value is derived from number of received  
365 AX.25 frame bytes plus a fixed Stetson-Harrison constant added per each packet  
366 for overheads. That is then divided by presumed channel modulation speed, and  
367 thus derived a figure somewhere in between 0.0 and 1.0.
- 368 2. Channel occupancy average in Erlangs over 10 minute interval. Same data source  
369 as above.
- 370 3. Count of received packets over 10 minutes.
- 371 4. Count of packets dropped for some reason during that 10 minute period.

372 Additional telemetry data points could be:

- 373 1. Number of transmitted packets over 10 minute interval
- 374 2. Number of packets IGated from APRSIS over 10 minute interval
- 375 3. Number of packets digipeated for this radio interface over 10 minute interval
- 376 4. Erlang calculations could include both Rx and Tx, but could also be separate.

377



## 378 2.9 Individual Call-Signs for Each Receiver, or Not?

379 Opinions are mixed on the question of having separate call-signs for each receiver (and  
380 transmitter), or not. Even the idea to use all 16 available SSIDs for a call-sign for  
381 something does get some opposition.

- 382 • There is no license fee in most countries for receivers, and there is no need to limit  
383 used call-signs only on those used for the site transmitters.
- 384 • There is apparently some format rule on APRSIS about what a “call-sign” can be,  
385 but it is rather lax: 6 alphanumerics + optional tail of: “-” (minus sign) and one or two  
386 alphanumerics. For example OH2XYZ-R1 style call-sign can have 36 different  
387 values before running out of variations on last character alone (A to Z, 0 to 9.)
- 388 • Transmitter call-signs are important, and there valid AX.25 format call-signs are  
389 mandatory.

390 On digipeater setup the receiver call-signs are invisible on RF. There only transmitter call-  
391 signs must be valid AX.25 addresses.

392

393 Transmitters should have positional beacons for them sent on correct position, and  
394 auxiliary elements like receivers could have their positions either real (when elsewhere), or  
395 actually placed near the primary Tx location so that they are separate on close enough  
396 zoomed map plot.

397 Using individual receiver identities (and associated net-beaconed positions near the real  
398 location) can give an idea of where the packet was heard, and possibly on which band. At  
399 least the *aprs.fi* is able to show the path along which the position was heard.

400

## 401 2.10 Beacons

402 Smallest time interval available to position viewing at aprs.fi site is 15 minutes. A beacon  
403 interval longer than that will at times disappear from that view. Default view interval is 60  
404 minutes.

405 Beacon transmission time **must not** be manually configured to fixed exact minute. There  
406 are large peaks in APRSIS traffic because of people are beaconing out every 5 minutes,  
407 and every 10 minutes, at exact 5/10 minutes. (Common happening with e.g. *digi\_ned*.)

408 Beaconing system must be able to spread the requests over the entire cycle time (10 to 30  
409 minutes) evenly. Even altering the total cycle time by up to 10% at random at the start of  
410 each cycle should be considered (and associated re-scheduling of all beacon events at  
411 every cycle start). All this to avoid multiple non-coordinated systems running at same  
412 rhythm. System that uses floating point mathematics to determine spherical distance in  
413 between two positions can simplify the distribution process by using float mathematics.  
414 Also all-integer algorithms exist (e.g. Bresenham's line plotting algorithm.)

```
415     float dt = (float)cycle_in_seconds;
416     for (int i = 0; i < number_of_beacons;++i) {
417         beacon[i].tx_time = now + (i+1) * dt;
418     }
```

419 With only one beacon, it will go out at the end of the beacon cycle.

420 Receiver location beacons need only to be on APRSIS with additional TCPXX token,  
421 transmitter locations could be also on radio.

### 422 2.10.1 Radio Beacons

423 “Tactical situation awareness” beaconing frequency could be 5-10 minutes, WB4APR does  
424 suggest at most 10 minutes interval. Actively moving systems will send positions more  
425 often. Transmit time spread algorithm must be used.

426 Minimum interval of beacon transmissions to radio should be 60 seconds. If more  
427 beacons need to be sent in this time period, use of Persistence parameter on TNCs (and  
428 KISS) should help: Send the beacons one after the other (up to 3?) during same  
429 transmitter activation, and without prolonged buffer times in between them. That is  
430 especially suitable for beacons *without* any sort of distribution lists.

431 **Minimize the number of radio beacons!**

### 432 2.10.2 Network beaconing

433 Network beaconing cycle time can be up to 30 minutes.

434 Network beaconing can also transmit positions and objects at much higher rate, than radio  
435 beaconing. Transmit time spread algorithm must be used.

436 Net-beacons could also be bursting similar to radio beacon Persistence – within a reason.

437

### 438 **3 Configuration Language**

439 System configuration language has several semi-conflicting requirements:

- 440 1. Easy to use
- 441 2. Minimal setup necessary for start
- 442 3. Sensible defaults
- 443 4. Self-documenting
- 444 5. Efficient self-diagnostics
- 445 6. Powerful – as ability to define complicated things

446

447 Examples of powerful, yet miserably complicated rule writing can be seen on *digi\_ned*, and  
448 *aprsd*. Both have proven over and over again that a correct configuration is hard to make.

449 On Embedded front, things like UIDIGI have tens of parameters to set, many of which can  
450 be configured so that the network behaviour is degraded, if not downright broken.

451 UIView32 has poor documentation on what to put on destination address, and therefore  
452 many users put there “WIDE” instead of “APRS,WIDE”, and thus create broken beacons.

453

454 Current *aprx* configuration follows “minimal setup” and “easy to use” rules, it is even “self-  
455 documenting” and “self-diagnosing”, but its lack of power becomes apparent.

456 Some examples:

- 457 1. radio serial /dev/ttyUSB0 19200 8n1 KISS callsign N0CALL-14
- 458 2. netbeacon for N0CALL-13 dest "APRS" via "NOGATE" symbol "R&"  
459 lat "6016.30N" lon "02506.36E" comment "aprx - an Rx-only iGate"

460 The “radio serial” definition lacks handling of multiple TNCs using KISS device IDs, and  
461 there is no easy way to define subid/callsign pairs.

462 The “netbeacon” format can do only basic “!”-type location fix packets. Extending it to  
463 objects would probably cover 99% of wanted use cases.

464 Both have extremely long input lines, no input line folding is supported!

465

### 466 3.1 APRSIS Interface Definition

467 There can be multiple APRSIS connections defined, although only one is used at any time.  
 468 Parameter sets controlling this functionality is non-trivial.

```

469 <aprsis>                                # Alternate A, single server, defaults
470     login  OH2XYZ-R1
471     server finland.aprs2.net:14580
472     filter ....
473     heartbeat-timeout 2 minutes
474 </aprsis>

475 <aprsis>                                # Alternate B, multiple alternate servers
476     login  OH2XYZ-R1
477     <server finland.aprs2.net:14580>
478         heartbeat-timeout 2 minutes
479         filter ....
480     </server>
481     <server rotate.aprs.net:14580>
482         heartbeat-timeout 120 seconds
483         filter ....
484         # Alt Login ?
485     </server>
486 </aprsis>

```

### 487 3.2 Radio Interface Definitions

488 Interfaces are of multitude, some are just plain serial ports, some can be accessed via  
 489 Linux internal AX.25 network, or by some other means, platform depending.

```

490 <interface>
491     serial-device /dev/ttyUSB1 19200 8n1 KISS
492     tx-ok         false           # receive only (default)
493     callsign      OH2XYZ-R2      # KISS subif 0
494 </interface>
495 <interface>
496     serial-device /dev/ttyUSB2 19200 8n1 KISS
497     <kiss-subif 0>
498         callsign OH2XYZ-2
499         tx-ok    true            # This is our transmitter
500     </kiss-subif>
501     <kiss-subif 1>
502         callsign OH2XYZ-R3      # This is receiver
503         tx-ok    false          # receive only (default)
504     </kiss-subif>
505 </interface>
506 <interface>
507     ax25-device OH2XYZ-6         # Works only on Linux systems
508     tx-ok       true            # This is also transmitter
509 </interface>

```

### 510 3.3 Digipeating Definitions

511 The powerfulness is necessary for controlled digipeating, where traffic from multiple  
512 sources gets transmuted to multiple destinations, with different rules for each of them.

- 513 1. Destination device definition (refer to “serial radio” entry, or AX.25 network  
514 interface), must find a “tx-ok” feature flag on the interface definition.
- 515 2. Possible Tx-rate-limit parameters
- 516 3. Groups of:
  - 517 1. Source device references (of “serial radio” or ax25-rxport call-signs, or “APRSIS”  
518 keyword)
  - 519 2. Filter rules, if none are defined, source will not pass anything in. Can have also  
520 subtractive filters – “everything but not that”. Multiple filter entries are processed  
521 in sequence.
  - 522 3. Digipeat limits – max requests, max executed hops.
  - 523 4. Control of treat WIDEn-N as TRACEn-N or not. (Default: treat as TRACE, know  
524 WIDEn-N, TRACEn-N, WIDE, TRACE, RELAY and thread them as aliases.)
  - 525 5. Alternate keywords that are controlled as alias of “WIDEn-N”
  - 526 6. Alternate keywords that are controlled as alias of “TRACEn-N”
  - 527 7. Additional rate-limit parameters

528

529 APRS Messaging transport needs some sensible test systems too:

- 530 • Station has been heard directly on RF without intermediate digipeater
- 531 • Station has been heard via up to X digipeater hops (X <= 2 ?)

532 APRS messaging stations may not be able to send any positional data!

533

534

535 Possible way to construct these groups is to have similar style of tag structure as Apache  
 536 HTTPD does:

```

537 <digipeater>
538     transmit OH2XYZ-2      # to interface with callsign OH2XYZ-2
539     ratelimit 20           # 20 posts per minute
540 #   viscous-delay 5        # 5 seconds delay on viscous digipeater
541     <trace>
542         keys RELAY,TRACE,WIDE,HEL
543         maxreq 4           # Max of requested, default 4
544         maxdone 4          # Max of executed, default 4
545     </trace>
546 #   <wide>                # Use internal default
547 #   </wide>
548     <source>
549         source OH2XYZ-2      # Repeat what we hear on TX TNC
550         filters             ....
551         relay-format digipeated # default
552     </source>
553     <source>
554         source OH2XYZ-R2     # include auxiliary RX TNC data
555         filters             ....
556         relay-format digipeated # default
557     </source>
558     <source>
559         source OH2XYZ-7      # Repeat what we hear on 70cm
560         filters             ....
561         relay-format digipeated # default
562         relay-addlabel 70CM  # Cross-band digi, mark source
563     </source>
564     <source>
565         source DSTAR         # Cross-mode digipeat..
566         filters             ....
567         relay-format digipeated # FIXME: or something else?
568         relay-addlabel DSTAR  # Cross-band digi, mark source
569         out-path WIDE2-2
570     </source>
571     <source>
572         source APRSIS        # Tx-IGate some data too!
573         filters             ....
574         ratelimit 10         # only 10 IGated msgs per minute
575         relay-format third-party # for Tx-IGated
576         out-path WIDE2-2
577     </source>
578 </digipeater>

```

579

580 **3.3.1 <trace>**

581 Defines a list of keyword prefixes known as “TRACE” keys.

582 When system has keys to lookup for digipeat processing, it looks first the trace keys, then  
583 wide keys. First match is done.584 If a per-source trace/wide data is given, they are looked up at first, and only then the global  
585 one. Thus per source can override as well as add on global sets.

```

586     <trace>
587         keys      RELAY, TRACE, WIDE, HEL1
588         maxreq    4          # Max of requested, default 4
589         maxdone   4          # Max of executed, default 4
590     </trace>

```

591

592 **3.3.2 <wide>**

593 Defines a list of keyword prefixes known as “WIDE” keys.

594 When system has keys to lookup for digipeat processing, it looks first the trace keys, then  
595 wide keys. First match is done.596 If a per-source trace/wide data is given, they are looked up at first, and only then the global  
597 one. Thus per source can override as well as add on global sets.

```

598     <wide>
599         keys      WIDE, HEL
600         maxreq    4          # Max of requested, default 4
601         maxdone   4          # Max of executed, default 4
602     </wide>

```

603

---

1 “HEL” is airport code for Helsinki Airport, so it is quite OK for local area distribution code as well.

### 604 3.3.3 <trace>/<wide> Default Rules

605 The <digipeater> level defaults are:

```

606   <trace>
607       keys      RELAY,TRACE,WIDE
608       maxreq    4      # Max of requested, default 4
609       maxdone   4      # Max of executed, default 4
610   </trace>
611   <wide>
612       keys      WIDE   # overridden by <trace>
613       maxreq    4      # Max of requested, default 4
614       maxdone   4      # Max of executed, default 4
615   </wide>

```

616

617 The <source> level defaults are:

```

618   <trace>
619       keys      # Empty set
620       maxreq    0      # Max of requested, undefined
621       maxdone   0      # Max of executed, undefined
622   </trace>
623   <wide>
624       keys      # Empty set
625       maxreq    0      # Max of requested, undefined
626       maxdone   0      # Max of executed, undefined
627   </wide>

```

628



629 **3.4 NetBeacon definitions**630 *Netbeacons* are sent only to APRSIS, and *Rfbeacons* to radio transmitters.

631 &lt;netbeacon&gt;

632 # to APRSIS # default for netbeacons

633 for N0CALL-13 # must define

634 dest "APRS" # must define

635 via "TCPIP,NOGATE" # optional

636 type "!" # optional, default "!"

637 symbol "R#" # must define

638 lat "6016.30N" # must define

639 lon "02506.36E" # must define

640 comment "aprx - an Rx-only iGate" # optional

641 &lt;/netbeacon&gt;

642 &lt;netbeacon&gt;

643 # to APRSIS # default for netbeacons

644 for N0CALL-13 # must define

645 dest "APRS" # must define

646 via "TCPIP,NOGATE" # optional

647 # Define any APRS message payload in raw format, multiple OK!

648 raw "!6016.35NR02506.36E&amp;aprx - an Rx-only iGate"

649 raw "!6016.35NR02506.36E&amp;aprx - an Rx-only iGate"

650 &lt;/netbeacon&gt;

651

652 **3.5 RfBeacon definitions**

653 *Netbeacons* are sent only to APRSIS, and *Rfbeacons* to radio  
 654 transmitters.

```

655 <rfbeacon>
656 # to      OH2XYZ-2      # defaults to first transmitter
657   for      N0CALL-13    # must define
658   dest     "APRS"       # must define
659   via      "NOGATE"     # optional
660   type     "!"          # optional, default "!"
661   symbol   "R&"         # must define
662   lat      "6016.30N"   # must define
663   lon      "02506.36E"  # must define
664   comment  "aprx - an Rx-only iGate" # optional
665 </rfbeacon>

```

```

666 <rfbeacon>
667 # to      OH2XYZ-2      # defaults to first transmitter
668   for      OH2XYZ-2     # must define
669   dest     "APRS"       # must define
670   via      "NOGATE"     # optional
671   type     ";"          # ";" = Object
672   name     "OH2XYZ-6"   # object name
673   symbol   "R&"         # must define
674   lat      "6016.30N"   # must define
675   lon      "02506.36E"  # must define
676   comment  "aprx - an Rx-only iGate" # optional
677 </rfbeacon>

```

678

679 Configuration entry keys are:

name	Optionality by type				
	! /	;	)		
to	x(1)	x(1)	x(1)		
for	--	--	--		
dest	--	--	--		
via	x	x	x		
raw	X(2,5)	X(2,5)	X(2,5)		
type	x(2)	x(2)	x(2)		
name	invalid	x(4)	x(4)		
symbol	X(3,4)	X(3,4)	X(3,4)		
lat	X(3,4)	X(3,4)	X(3,4)		
lon	X(3,4)	X(3,4)	X(3,4)		
comment	X(3,4)	X(3,4)	X(3,4)		

680

681 Optionality notes:

- 682 1. Netbeacons default is APRSIS system, and no transmitter is definable. Rfbeacons  
683 default to first transmitter call-sign defined in <interface> sections, any valid  
684 transmitter call-sign is OK for “to” keyword.
- 685 2. When a “raw” is defined, no “type” must be defined, nor any other piecewise parts  
686 of symbol/item/object definitions.
- 687 3. Piecewise definitions of basic positional packets must define at least *type* + *symbol*  
688 + *lat* + *lon*. The *comment* is optional, and *name* is rejected if defined.
- 689 4. Piecewise definitions of item and object packets must define at least *type* + *name*  
690 + *symbol* + *lat* + *lon*. The *comment* is optional.
- 691 5. Multiple “raw” entries are permitted, they share *to* + *for* + *dest* + *via* -field data, and  
692 each generates a beacon entry of its own.
- 693 6. Defining timestamped position/object/item packet will get a time-stamp of “h” format  
694 (hours, minutes, seconds) generated when beacon is sent. This applies also to *raw*  
695 packets! Computer must then have some reliable time source, NTP or GPS.

696