

# 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) digipeats.....	13
2.6.3	Other (Control != 0x03) digipeats.....	14
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  $\Rightarrow$  igate-to-aprsis + digi  $\Rightarrow$  tx  $\Rightarrow$  rx  $\Rightarrow$  igate-to-aprsis + 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. If the frame has VIA fields and none are with H-bit clear, feed the packet to duplicate checker, and drop it afterwards. (Necessary for *Viscous Digipeaters*.)
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. 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.
5. 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.<sup>1</sup>
6. FIXME: WIDEn-N/TRACEn-N treatment rules: By default treat both as TRACE, have an option to disable "WIDE-is-TRACE" mode. Configured list of prefix tokens that are treated like TRACE or WIDE.
7. 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.
8. 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.

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:

<sup>1</sup> Possible alternate would be to mark all VIA parts completed, but then the user who uses excessively large requests will observe a digipeat and not have incentive to correct their ways...

259 SRCCALL-n>DEST-2: ...

260

## 261 **2.6.2 Other UI (Control=0x03, PID != 0xF0) digipeats**

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

263 Support for this is optional needing special configuration enable entries.

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

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

279

## 280 **2.6.3 Other (Control != 0x03) digipeats**

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

282 Support for this is optional needing special configuration enable entry.

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

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

289

#### 290 2.6.4 Viscous Digipeating

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

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

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

300 Simplest way to implement this filtering is to count matches on dupe-check database. First  
301 heard packet is number one, second heard *may be* such that it is *fully digipeated* (by  
302 counts or other rules), but it requires that *all* received packets are fed to dupe-check  
303 database.

304 If the dupe-check database has count other than one at the end of the “probation delay”,  
305 then the packet will not to be transmitted by the viscous digipeater.

306

## 307 2.7 Duplicate Detector

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

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

311

### 312 2.7.1 Control=0x03,PID=0xF0: APRS

313 Normal digipeater duplicate packet detection compares message source (with SSID),  
314 destination (without SSID!), and payload data against other packets in self-expiring  
315 storage called “duplicate detector”. Lifetime of this storage is commonly considered to be  
316 30 seconds.

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

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

326

### 327 2.7.2 Control=0x03,PID!=0xF0: Others

328 Other type digipeater duplicate packet detection compares message source, and  
329 destination (both with SSID!), and payload data against other packets in self-expiring  
330 storage called “duplicate detector”. Lifetime of this storage is commonly considered to be  
331 30 seconds.

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

333

### 334 2.7.3 Control != 0x03: Others

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

336

## 337 2.8 Radio Interface Statistics Telemetry

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

- 340 1. Channel occupancy average in Erlangs over 1 minute interval, and presented as  
341 busiest 1 minute within the report interval. Where real measure of carrier presence  
342 on radio channel is not available, the value is derived from number of received  
343 AX.25 frame bytes plus a fixed Stetson-Harrison constant added per each packet  
344 for overheads. That is then divided by presumed channel modulation speed, and  
345 thus derived a figure somewhere in between 0.0 and 1.0.
- 346 2. Channel occupancy average in Erlangs over 10 minute interval. Same data source  
347 as above.
- 348 3. Count of received <packets over 10 minutes.
- 349 4. Count of packets dropped for some reason during that 10 minute period.

350 Additional telemetry data points could be:

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

355



## 356 2.9 Individual Call-Signs for Each Receiver, or Not?

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

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

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

370

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

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

378

## 379 2.10 Beacons

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

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

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

```
393     float dt = (float)cycle_in_seconds;
394     for (int i = 0; i < number_of_beacons;++i) {
395         beacon[i].tx_time = now + (i+1) * dt;
396     }
```

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

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

### 400 2.10.1 Radio Beacons

401 "Tactical situation awareness" beaconsing frequency could be 5-10 minutes, WB4APR does  
402 suggest at most 10 minutes interval. Actively moving systems will send positions more  
403 often. Transmit time spread algorithm must be used.

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

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

### 410 2.10.2 Network beaconsing

411 Network beaconsing cycle time can be up to 30 minutes.

412 Network beaconsing can also transmit positions and objects at much higher rate, than radio  
413 beaconsing. Transmit time spread algorithm must be used.

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

415

### 416 3 Configuration Language

417 System configuration language has several semi-conflicting requirements:

- 418 1. Easy to use
- 419 2. Minimal setup necessary for start
- 420 3. Sensible defaults
- 421 4. Self-documenting
- 422 5. Efficient self-diagnostics
- 423 6. Powerful – as ability to define complicated things

424

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

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

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

431

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

434 Some examples:

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

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

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

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

443

### 444 3.1 APRSIS Interface Definition

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

```

447 <aprsis>                                # Alternate A, single server, defaults
448     login  OH2XYZ-R1
449     server finland.aprs2.net:14580
450     filter ....
451     heartbeat-timeout 2 minutes
452 </aprsis>

453 <aprsis>                                # Alternate B, multiple alternate servers
454     login  OH2XYZ-R1
455     <server finland.aprs2.net:14580>
456         heartbeat-timeout 2 minutes
457         filter ....
458     </server>
459     <server rotate.aprs.net:14580>
460         heartbeat-timeout 120 seconds
461         filter ....
462         # Alt Login ?
463     </server>
464 </aprsis>

```

### 465 3.2 Radio Interface Definitions

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

```

468 <interface>
469     serial-device /dev/ttyUSB1 19200 8n1 KISS
470     tx-ok         false           # receive only (default)
471     callsign      OH2XYZ-R2      # KISS subif 0
472 </interface>
473 <interface>
474     serial-device /dev/ttyUSB2 19200 8n1 KISS
475     <kiss-subif 0>
476         callsign OH2XYZ-2
477         tx-ok    true            # This is our transmitter
478     </kiss-subif>
479     <kiss-subif 1>
480         callsign OH2XYZ-R3      # This is receiver
481         tx-ok    false         # receive only (default)
482     </kiss-subif>
483 </interface>
484 <interface>
485     ax25-device OH2XYZ-6        # Works only on Linux systems
486     tx-ok       true           # This is also transmitter
487 </interface>

```

### 488 3.3 Digipeating Definitions

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

- 491 1. Destination device definition (refer to “serial radio” entry, or AX.25 network  
492 interface), must find a “tx-ok” feature flag on the interface definition.
- 493 2. Possible Tx-rate-limit parameters
- 494 3. Groups of:
  - 495 1. Source device references (of “serial radio” or ax25-rxport call-signs, or “APRSIS”  
496 keyword)
  - 497 2. Filter rules, if none are defined, source will not pass anything in. Can have also  
498 subtractive filters – “everything but not that”. Multiple filter entries are processed  
499 in sequence.
  - 500 3. Digipeat limits – max requests, max executed hops.
  - 501 4. Control of treat WIDEn-N as TRACEn-N or not. (Default: treat as TRACE, know  
502 WIDEn-N, TRACEn-N, WIDE, TRACE, RELAY and thread them as aliases.)
  - 503 5. Alternate keywords that are controlled as alias of “WIDEn-N”
  - 504 6. Alternate keywords that are controlled as alias of “TRACEn-N”
  - 505 7. Additional rate-limit parameters

506

507 APRS Messaging transport needs some sensible test systems too:

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

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

511

512

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

```

515 <digipeater>
516     transmit OH2XYZ-2      # to interface with callsign OH2XYZ-2
517     ratelimit 20           # 20 posts per minute
518 #   viscous-delay 5        # 5 seconds delay on viscous digipeater
519     <trace>
520         keys      RELAY,TRACE,WIDE,HEL
521         maxreq    4        # Max of requested, default 4
522         maxdone   4        # Max of executed, default 4
523     </trace>
524 #   <wide>           # Use internal default
525 #   </wide>
526     <source>
527         source OH2XYZ-2      # Repeat what we hear on TX TNC
528         filters             ....
529         relay-format        digipeated # default
530     </source>
531     <source>
532         source OH2XYZ-R2     # include auxiliary RX TNC data
533         filters             ....
534         relay-format        digipeated # default
535     </source>
536     <source>
537         source OH2XYZ-7      # Repeat what we hear on 70cm
538         filters             ....
539         relay-format        digipeated # default
540         relay-addlabel 70CM   # Cross-band digi, mark source
541     </source>
542     <source>
543         source DSTAR         # Cross-mode digipeat..
544         filters             ....
545         relay-format        digipeated # FIXME: or something else?
546         relay-addlabel DSTAR # Cross-band digi, mark source
547         out-path            WIDE2-2
548     </source>
549     <source>
550         source APRSIS        # Tx-IGate some data too!
551         filters             ....
552         ratelimit           10    # only 10 IGated msgs per minute
553         relay-format        third-party # for Tx-IGated
554         out-path            WIDE2-2
555     </source>
556 </digipeater>

```

557

### 558 3.3.1 <trace>

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

560 When system has keys to lookup for digipeat processing, it looks first the trace keys, then  
561 wide keys. First match is done.

562 If a per-source trace/wide data is given, they are looked up at first, and only then the global  
563 one. Thus per source can override as well as add on global sets.

```
564     <trace>
565         keys      RELAY, TRACE, WIDE, HEL2
566         maxreq    4          # Max of requested, default 4
567         maxdone   4          # Max of executed, default 4
568     </trace>
```

569

### 570 3.3.2 <wide>

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

572 When system has keys to lookup for digipeat processing, it looks first the trace keys, then  
573 wide keys. First match is done.

574 If a per-source trace/wide data is given, they are looked up at first, and only then the global  
575 one. Thus per source can override as well as add on global sets.

```
576     <wide>
577         keys      WIDE, HEL
578         maxreq    4          # Max of requested, default 4
579         maxdone   4          # Max of executed, default 4
580     </wide>
```

581

---

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

### 582 3.3.3 <trace>/<wide> Default Rules

583 The <digipeater> level defaults are:

```

584     <trace>
585         keys      RELAY,TRACE,WIDE
586         maxreq    4      # Max of requested, default 4
587         maxdone   4      # Max of executed, default 4
588     </trace>
589     <wide>
590         keys      WIDE    # overridden by <trace>
591         maxreq    4      # Max of requested, default 4
592         maxdone   4      # Max of executed, default 4
593     </wide>

```

594

595 The <source> level defaults are:

```

596     <trace>
597         keys      # Empty set
598         maxreq    0      # Max of requested, undefined
599         maxdone   0      # Max of executed, undefined
600     </trace>
601     <wide>
602         keys      # Empty set
603         maxreq    0      # Max of requested, undefined
604         maxdone   0      # Max of executed, undefined
605     </wide>

```

606



### 607 3.4 NetBeacon definitions

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

```

609 <netbeacon>
610 # to      APRSIS          # default for netbeacons
611   for      N0CALL-13      # must define
612   dest     "APRS"         # must define
613   via      "TCPIP,NOGATE"  # optional
614   type     "!"            # optional, default "!"
615   symbol   "R&"           # must define
616   lat      "6016.30N"     # must define
617   lon      "02506.36E"    # must define
618   comment  "aprx - an Rx-only iGate" # optional
619 </netbeacon>

620 <netbeacon>
621 # to      APRSIS          # default for netbeacons
622   for      N0CALL-13      # must define
623   dest     "APRS"         # must define
624   via      "TCPIP,NOGATE"  # optional
625 # Define any APRS message payload in raw format, multiple OK!
626   raw      "!6016.35NR02506.36E&aprx - an Rx-only iGate"
627   raw      "!6016.35NR02506.36E&aprx - an Rx-only iGate"
628 </netbeacon>
629
```

630 **3.5 RfBeacon definitions**

631 *Netbeacons* are sent only to APRSIS, and *Rfbeacons* to radio  
 632 transmitters.

```

633 <rfbeacon>
634 # to      OH2XYZ-2      # defaults to first transmitter
635   for      N0CALL-13    # must define
636   dest     "APRS"       # must define
637   via      "NOGATE"     # optional
638   type     "!"          # optional, default "!"
639   symbol   "R&"         # must define
640   lat      "6016.30N"   # must define
641   lon      "02506.36E"  # must define
642   comment  "aprx - an Rx-only iGate" # optional
643 </rfbeacon>

```

```

644 <rfbeacon>
645 # to      OH2XYZ-2      # defaults to first transmitter
646   for      OH2XYZ-2     # must define
647   dest     "APRS"       # must define
648   via      "NOGATE"     # optional
649   type     ";"          # ";" = Object
650   name     "OH2XYZ-6"   # object name
651   symbol   "R&"         # must define
652   lat      "6016.30N"   # must define
653   lon      "02506.36E"  # must define
654   comment  "aprx - an Rx-only iGate" # optional
655 </rfbeacon>

```

656

657 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)		

658

659 Optionality notes:

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

674