APRServe: An Internet Backbone for APRS

Steve Dimse K4HG
http://www.aprs.net/k4hg.html
k4hg@tapr.org

Last year at the 1996 Digital Communications Conference I predicted that within the next year we would have a working nation-wide APRS backbone running on the internet. This paper details the progress that. has been made towards that goal.

At the time this is written (early August 1997), there is full time live data available from San Francisco, Los Angeles, Nashville, Atlanta, Miami, and New Jersey. Several other sites offer continuously updated files containing almost-live data.

Each of these sites functions independently, using a local WWW server to display the data with javAPRS. The first step has been taken to link these sites into a seamless network with a program I have written called APRServe.

In planning APRServe, I had several goals in mind.

1. The program should relay data from a local TNC onto the internet, just as the other Internet. Gates (IGates) do.

2. The program should be able to connect, to a list of other servers to obtain live data from other full time sites, and to interconnect with other copies of itself running in other locations.

3. There should be an echo function to allow non-permanent. sources to connect to APRServe, and add their data to the stream.

4. The data coming in from the various sources should be filtered to eliminate as much redundant data as possible.

5. There should be a buffer of data to allow the immediate transmission of all known information to a newly connected client.

6. There should to be a way to remotely monitor the status of the server.

7. Local TNC data should be accessible either alone or as part of the merged stream of all data.

Implementation

When I first envisioned this project, I intended to reuse much of t.he Java code I had written for javAPRS. However, as I worked more with Java I came to realize that it is not yet mature enough to handle the demands of a full time server application. I have written the program in C++ under MacOS 7.6, using Apple's Open Transport networking architecture for peak performance. The resulting program is much faster and more stable than a Java program would have been, but is not transportable to other computer systems. However, while writing the program I made every effort to make it as easy to port. as possible. This means no fancy user interface, parameters are read from text files when the program initializes, and status messages are displayed in a simple text window.

Networking

On startup, the program reads a file containing a list of full tim; data servers, and connects to them. It monitors each connection, and if no data is seen in 15 minutes, the connection is assumed to be dead, and a reconnection is attempted every 15 minutes until successful.

Multiple copies of APRServe can interconnect, sharing data among themselves. When other full-time servers are available, a failure in one will not mean that the entire network becomes unavailable. Also, the loads of both clients and IGates can be shared between all operating servers. At present, the protocol between the servers is quite simple. The only difference between a client. port and an interconnect. port at present is that the data sent out to an interconnect port does not contain any data that arrived by that port. (Without this difference, data would echo back and forth between two sites.)

Data Echo

In an ideal world, there would be a full time Internet. gate (IGate) within a single digi hop of everywhere in the country. However, since a full time IGate requires a permanent connection to the internet, this coverage is unlikely t.o ever be attained. To fill in the gaps between full time IGates, individual users of Mac/WinAPRS will be able to send their TNC data to APRServe and have it relayed to other internet users. This feature can also be used for special events by having simple programs that send data from a TNC to APRServe.

Data Filtering

As anyone who has watched the raw data on a busy APRS net can attest, there is a lot of redundant. information in the stream. Others are working on improvements to the TNC digipeater code and protocol to reduce the redundancy. However, given that there may be a large number of redundant. data streams feeding into APRServe, the possibility exists to quickly overwhelm the average dial up internet connection.

To address this I have developed a simple filtering routine that has low processor overhead and good selectivity. As each string comes through APRServe, the program calculates a one byte bitwise checksum and a character count on the data portion of the packet (that which follows the ':', thereby excluding the callsign and digi info). The checksum and count are combined with a character count of the data into a 16 bit integer hash code. For each data stream the program maintains a buffer of a variable number (set at compile time) of packet callsigns and their hash code. The incoming packet’s hash code is compared with those of the packets in the buffer. If a match is found, it is confirmed by checking the callsign at the start of the packet, and if confirmed the packet is discarded. If the packet is not in the buffer, then it. overwrites the oldest data in the buffer and the packet is passed on for processing.

The optimum size of the packet buffer is still to be determined. A larger buffer would cut out some longer period QRM, such as the parked vehicle tracker with the GPS turned off that repeats the same posit every minute. A small buffer, say 16 packets, cuts out all of the pings between digis (a big problem in Miami) with much less processor overhead, and is what I presently use.

In a test on 72 hours of data from the Miami LAN, this method was found to reduce the size of data bv about 60% with no information loss. Using a buffer of 128 packets only improved the reduction to 68%.

Data Buffering

APRServe presently hears somewhere between 450 and 600 stations, depending on propagation. However, many of these are heard infrequently, either because of distance to the IGate, or the low transmission rate used by fixed stations. To let a user see all the heard stations immediately upon connecting to APRServe, the programs keeps an internal buffer of heard stations. This buffer stores a single packet of each of three different types for each station: weather, position, and other. Each incoming packet is parsed to determine it's type, and it overwrites the prior packet. When a new client. connects, the contents of this buffer are sent immediately to the client.

In order to assure the timeliness of the data, every half hour the buffer is checked, and if no packet was heard from a station in the prior 12 hours, the station is purged from the buffer.

Remote monitoring

Since the APRServe machine runs in a location I cannot easily access, I have added a status port. When I connect to this port. the program reports the present status of the program, including the current connections. Status messages are reported to the screen display and to the status port. Some very simple commands can be sent to the program from this route as well.

Client connection

To access the data, a client connects to the server using the TCP/IP Telnet protocol. APRServe supports connections on two ports. Port 14579 (chosen for the APRS VHF frequency) send only the local data from the TNC input. This keeps users from "drinking from a firehose" when they only want to see Miami data. Port 1015 1 (the HF APRS frequency) serves the full data stream.

Presently there are two clients for the APRS Internet Network. javAPRS, being "born" to networking, is fully functional now. MacAPRS, thanks to Apple's Communication Toolbox, is able, with a public domain tool, to connect to the network now. It. does not. handle the full protocol at this time. WinAPRS is expected to catch up to MacAPRS once the Sprouls get a chance. I expect that most "hard core" APRS users will prefer the Mac/WinAPRS client, as it. is faster and has more features than javAPRS. javAPRS is more useful for casual users, and most especially for newcomers and non-hams, since it is automatically downloaded and run by the Web browser.

But is it Ham Radio?

I hear this a lot. Of course it is ham radio. Our hobby is about communicating and experimenting; this system does both.

Ham radio needs to embrace new technologies, and concentrate on those aspects that are unique. Let's face it., a 200 mile 1200 baud packet. conversation fades compared to an internet 33.6 kbaud video chat halfway around the world. If ham radio continues to draw in upon itself and celebrate past glory, it. will wither and die. We must show that we can utilize the unique aspects of amateur radio, and merge them with existing technology to produce novel communications systems.

Future plans

1. TIGER map relay. javAPRS has had the capability for the last year of using the TIGER map server from the census bureau. The problem is that the security rest.rict.ions of java prevent. the use of data from servers other than the one from which the program is run. The solution is to relay the map images through the same server that. the applet is loaded from. Steve Boyle (KD6WXD) has this working on his server, and I will be adding the capability to my system.

2. Intelligent routing of messages. I am strongly against. using the internet. for the wholesale routing of traffic between different RF nets. However, the intelligent routing of messages would greatly increase the usefulness of the Internet. backbone while only minimally increasing the traffic. This year's prediction is that next year I will be presenting a DCC paper detailing how this nationwide messaging system was designed and implemented.

3. Improved interconnection. At present, multiple sites running APRServe all need to be connected to the same IGates in order to be assured of access to the data should another APRServe site go offline. The creates unnecessary load on both the IGates and APRServe sites. I am developing a negotiation protocol to allow the servers to split. up the IGate load dynamically. In this way, each server maintains a list of the available IGat.es and APRServe programs. When an APRServe site goes offline, this is sensed by the remaining APRServe site(s), and the missing server's load is split between the remaining site(s).

Appendix A
Annotated Bibliography of APRS Internet Sites

http://www.aprs.net/usa.html The Miami APRServe page, with links to various documents describing the hardware and software in more detail.

ftp://ftp.tapr.org/pub/tapr/SIG/aprssig/files This is the primary FTP site for APRS files. The latest. version of the Mac, Windows, DOS, and Java versions of the program are available here, as well as t.he largest collection of maps for all platforms.

http://web.usna.navy.mil/~bruninga/aprs.html Bob Bruniga's ("The Father of APRS") site contains many pages of almost live data. A must see site to understand APRS capabilities,

http://www.aprs. net/javAPRS.html A series of pages containing demonstrations of javAPRS's capabilities, as well as links to most of the other javAPRS pages I know about.

http://www.aprs. net/javAPRSprog.html The official documentation for javAPRS for those people who are interested in adding javAPRS to their page.

http://www.aprs.net/find/ It. is possible to use the APRServe network to follow individual stations in their travels. A new feature in javAPRS filters out all except specified stations. The above URL lists those pages which I have created for individuals. For example, http://www.aprs.net/find/k4hg.html shows my various stations, and http://www.aprs. net/f ind/w7lus. html follows the exploits of long haul trucker and long time APRSer Peter Gross. If you are on APRS and would like a page here let me know.

http://www.kcaprs.org The Kansas City APRS Group's site is one of the more complete sites for APRS info.

http://aprs.rutgers.edu Run by the Sprouls, containing the most up-to-date documentation on Mac/WinAPRS, and also has an FTP server that contains latest Mac/WinAPRS programs and lots of maps.