Server Applications within APRS™ Internet Server Environment

Thomas M. Schaefer, NY4I
Utah APRS User's Group
11678 Littler Rd
Sandy, UT 84092
nv4imarrlnet

Abstract

This paper discusses programmatic opportunities to access the Internet servers offering APRS data. Particular attention is made to the APRServe system in Miami. Using this system, it is possible for a programmer to create applications that utilize APRS data to create new dynamic and useml applications.

Introduction

With the advent of the APRServe network several years ago, it is now possible to transcend general intemet connectivity between APRS programs and create new server applications to utilize this data. This paper presents the background on the server packet architecture as it applies to server access of the data. Through examination of DXSpot, a program to send DX packet cluste:r spots to the APRS the author illustrates methods to connect to the APRS server network and make use of the data.

Connecting to the APRServe system

The first step to using the APRServe data is getting the APRServe data. Connecting requires some knowledge of socket programming and the port structure of APRServe. APRServe offers data on several different TCP/IP ports. Briefly, TCP/IP ports are different "views" of data on a single system. While a system has a unique TCP/IP address, many different services run on a system. External programs need a way to connect to a distinct service or program. For example, whenever one uses FTP to connect to a server; port 21 is first used to establish the connection. A similar thing happens with a web browser. The web browser generally connects to port 80 to communicate with a specific service (the web server) on the system. In the case of APRServe, the ports that are of general interest are

23	Non-buffered real-time information
10151	Buffered (past 8 hours) of information
14579	Non-buffered Local data (Miami)

For most server applications, buffered data is not useful; therefore, typical server applications connect to port 23. One convenience to port 23 being the non-buffered data is that port 23 is the Telnet port. Therefore, one only need connect to www.aprs.net with telnet to see the real-time feed of data. This single statement is the basis to collect APRServe data in an external application. If the only desire is to read this data, then the programmer simply has the program open a socket to port 23 of the www.aprs.net host, and the data is received as separate messages. The following is a few example packets received from port 23 of the APRServe:

```
KE6QNK-3>APRX46,WIDE3-1 :=3759.28N/12200.6OW#OOO/OOO/
KF4TLJ>APS199/V:>051724z[199YD]*Sheriff1s Tactical Amateur Radio
K5QBM-2~APRS,WIDE3-1:!3254.OONNO9640.88W#PHG7:38O/WIDEn-n
KA7PBI-lO>APZO34,TCPIP* :=4734.07N/12208.06W-PHG563O/XASTIR-Linux
WBOSBH*>APRS,WIDE,WIDE:!4354.88N/O9229.78WoPH(~5230 Rochester
NOCALL>APRS,TCPXX*: !3853.00NN010140.88W#PHG1234
```

As these packets are received, whatever parsing is required should be performed and the required action taken. One possible application is searching for messages to a particular callsign. The processing, of course, is dictated by the application need. While the above messages may appear generic, the number of different APRS messages is large. The APRS protocol reference [l] is best referenced to determine all the possible messages. To keep things manageable, it is only required to search for the messages that are of interest to a program. For example, if a program were written to check for callsign requests, that message would typically be in an APRS message. Since an APRS message has a distinct format, only packets following the message format need to be parsed beyond general identification.

Authenticated packets

APRServe contains a mechanism to verify that a message on the Internet has come from an amateur radio operator. All registered APRS programs and anyone else may contain a registration code that allows generation of a password to send messages to the APRServe and then to a local 2m packet network. The idea being that only “verified” packets should be sent to RF, or “gated”. The distinction in the APRServe between verified and non-verified packets is the TCP field in the path. By observation of the above examples, it can be seen that KA7PBI-10 sent a position via the Internet. This is indicated by the APZ034,TCPIP string in the packet path. By comparison, KEGQNK-3 was received from 2m via an IGate. This is indicated because the path is APRX46,WIDE3-1 (or any other combination typically found in a X.25 path header). The method used by APRServe to indicate unverified packets is to change the TCPIP path to TCPXX. This is illustrated above by NOCALL. If a server application receives a packet with TCPXX, it is important that this packet not be sent to RF since this is not proven to be an amateur originated packet. FCC rules require that only packets originated by hams be sent over RF.

Sending Packets to APRServe

The other side of the APRServe system is to be able to send packets to the server. The topology of the system states that any packet sent to the system will then be sent to every Internet Gateway (or custom server application) connected to the system. It is this system that allows custom server applications to communicate with the APRS stations over RF. The protocol to send to APRServe is more involved than simply receiving data. First in order to send data that is then sent to RF, the server application needs to be authenticated. To authenticate any server program, the following sting must be sent to the connected APRServe socket:

```
user <callsign> PASS <password> vers <Program Name and version>
```

The APRServe system requires that a password derived from the callsign be obtained to send messages to the RF side of the system.

To obtain a password, send an APRS message to ICQServe with PASSWORD as the message.

Once the login is completed, the application is ready to send messages to the APRServe for possible transmission to the 2m network. The format of the packets is very similar to the APRS "on-air" packets. The only exception is that the path used in the packet is TCPIP*. For example, presume a system call DXSpot wants to send a position packet. The format of the packet is as follo.ws:

```
DXSPOT>APRS,TCPIP*:*:!4028.00NN11151.88W#
```

When this packet is received, it is sent to all the other servers on the system. In a similar fashion, if a server called EMAIL wants to send a message to NY41, the packet would appear as follows:

```
EMAIL>APRS,TCPIP*::NY41      :Your email has been sent
```

An interesting feature is that authentication is determined on a port basis. In other words, if the proper login string is sent on the connected socket, any packets sent on that port are considered to be authenticated. It is perfectly reasonable to login with a password of a callsign, and then send the packets from an application name like ICQServe, DXSpot, or EMAIL.

Case Study

To properly illustrate the capabilities of the APRServe system with server applications, an example program will now be examined. DXSpot is a Per1 program that connects to both a DX Cluster and APRServe all via the Internet. DXSpot's purpose is to send DX Cluster information over a local 2m network. Regardless of one's specific feelings over the purpose of send DX information on an APRS frequency, there are some valid reasons. Several amateurs have both an APRS radio like the Kenwood D700 in their vehicle as well as an HF rig like the Yaesu FTl 00 or the Icom 706. The Kenwood radios offer the ability to receive and display DX information sent from a packet cluster. In almost all cases, this information is sent on a frequency separate from 144.39 MHz. If the APRS radio is operating on the APRS frequency, then the packet spots cannot be obtained from the network: at the same time. In order to make DX spots more useable for radios monitoring 144.390, a program called DXSpot was created. This program is written specifically to a certain DX Cluster system (CLX) [2]. The CLX packet cluster is a popular Linux-based system that offers Telnet access to obtain DX spot information. While this program's login may differ from other clusters, it is normally only by slight differences. After login to the cluster, the DXSpot program waits for a DX Spot to be received on the cluster socket. After receiving the packet, the spot is then converted into an APRS DX Spot format [3]. The most important part of this program is the following: the originator of the packet is sent to SLCDX (SLCDX because that is where the program runs). The following is an example of a properly formatted DX spot to be sent on the APRServe socket:

```
SLCDX>DXSPOT,TCPIP*: cDX Spot as received from DX Cluster>
```

This packet is then sent to the APRServe. When the APRServe receives this packet, it then sends it out to all the connected IGates. When the IGate in Salt Lake City receives it, it is sent out to the local 2m RF network. This little bit of magic happens because the IGate in Salt Lake City (running aprsd) will send any TCP/IP packet from SLCDX automatically to the RF network. To duplicate this system for ones own area, first find out the exact method to telnet to the local DX cluster (if possible), then modify DXSpot to connect properly. Ask the IGate sysop to add some code to the gate to send packets from the DXSpot software to RF immediately. Airport codes seem to be the adopted standard in the aprsd case.

Future Issues

With the current availability of the FindU APRS data storage system, a fundamental design architectural question now exists. If the server required by an application is real-time, the connecting to the feed seems to make more sense. If the application only requires query on a specific packet, then a database system like FindU is warranted. The application programmer should carefully consider what type of data is required.

Conclusion

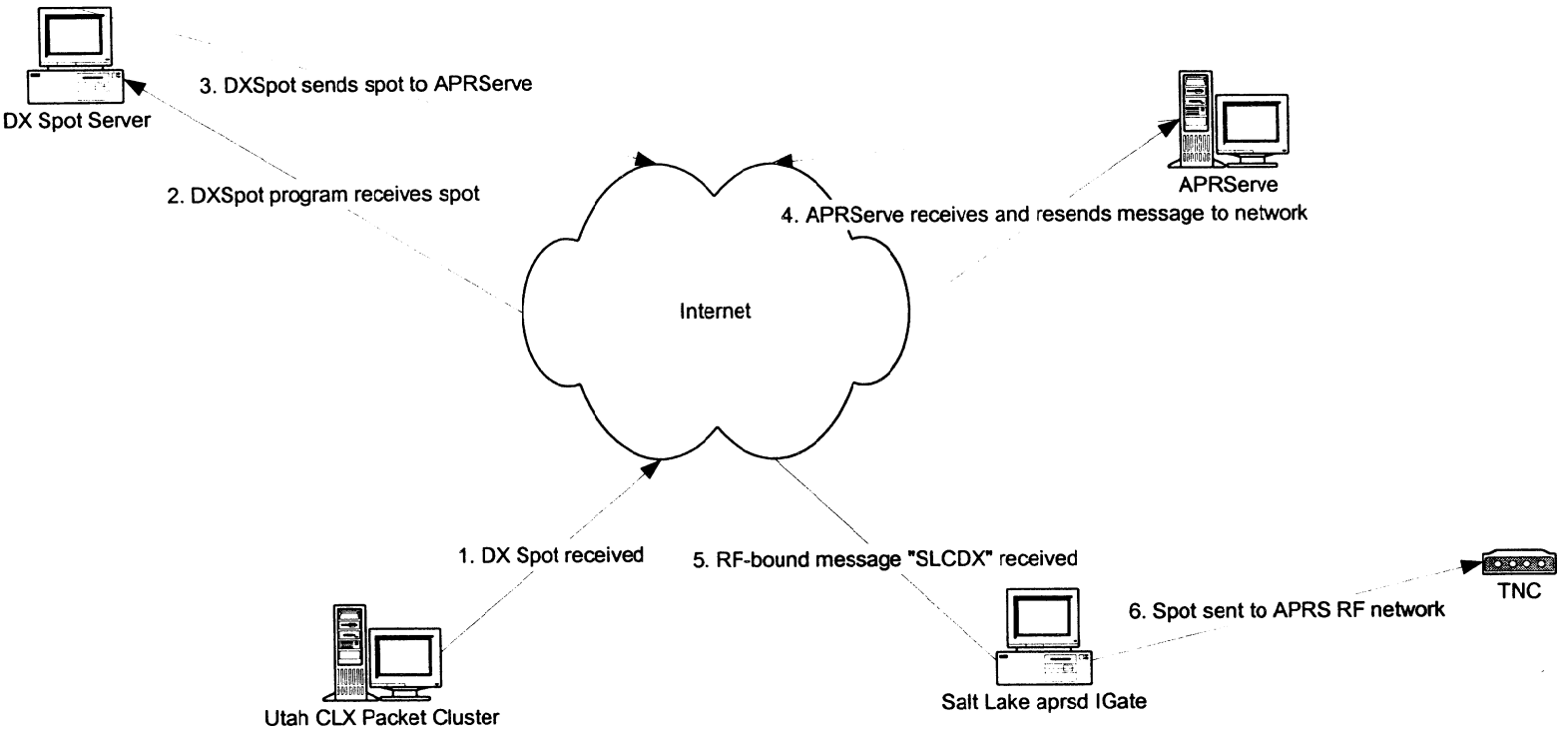
Using existing TCP/IP network programming concepts, it is possible to create useful applications using the APRServe data feed. At the core of the system is a server that allows multiple connections and distribution of data to many different connected programs. The APRServe system is the common link that most local APRS networks share. With this sharing of data, app1ication.s such as DXSpot, Email, and ICQServe can be created to make real-time use of the entire worldwide APRS network. With simply a computer and an Internet connection, packets can be negotiated all over the world for the benefit of all APRS users

References

VI APRS Working Group, “APRS Protocol Reference,” [Online document], 2000 May 1 (Rev 10 1 m), Available FTP: ~://ftp.tapr.org/aprssig/aprsspec/spec/aprs10l~APRSlO1m.pdf

[2] Franta Bendl, DJ0ZY and Bernhard (Ben) Biittner, DL6RAL “CLX Homepage,” [Online Document], 2000 June 18 (Rev 5.03a), Available HTTP: http://www.clx.muc.de

[3] Ian Maude, GOVGS, “CLX User Manual,” [Online Document], 1999 Dee (Rev 4.03), Available HTTP: http:l/~.clx.muc.deluser/english/htmlh~se~an~4.html#ss4.2


DX Spot Server
3. DXSpot sends spot to APRServe
2. DXSpot program receives spot
APRServe
4. APRServe receives and resends message to network
Internet
1. DX Spot received
5. RF-bound message "SLCDX" received
TNC
6. Spot sent to APRS RF network
Utah CLX Packet Cluster
Salt Lake aprsd IGate