HamWeb: Rethinking Packet Radio

by John Hansen, WAOPTV
State University of New York at Fredonia

Abstract: This paper describes a general purpose implementation of a simple "broadcast protocol" useful for terrestrial amateur packet links. It allows the transfer of files and entire directory structures from a server to many client stations simultaneously. Consideration is also give to applications of HTML to amateur packet links.

Keywords: Packet, Broadcast Protocol, HTML

Once upon a time I thought there was only one issue with regard to speed and packet radio: how to get more of it. About six years ago I spent a considerable amount of time and effort arguing in favor of abandoning 1200 baud packet in favor of 9600 baud, and ultimately higher speeds. Back then I was running a packet radio BBS and went so far as to open a user port that ran at 9600 baud. It saw extremely little use. I was thinking of offering certificates for the first 5 users of that port, but never got that many takers.

At first, it seemed like the barrier to higher speeds was the lack of commercially-available equipment. While you could buy an add-on modem for your TNC for about $100, installing it was no cakewalk. Furthermore, at that time no commercially-available transceiver would support 9600 baud out of the box. Finally the packet equipment manufacturers started to offer TNC's with 9600 baud built in and some radio manufactures began to offer 9600 baud-ready radios. However, there were still very few takers.

Back when packet radio was getting started, the most commonly used telephone modem ran at 300 baud. TNC's capable of 1200 baud must have seemed fast. I know I was impressed by the speed when I first got on packet. Now the most common telephone modems are 28,800 baud and the most common packet modems are (you guessed it) still 1200 baud. And, according to recent reports, conventional packet radio is withering.

Why did higher speed packet never catch on for end users? The conventional wisdom is that ham radio operators are both too cheap and too lazy to take on the financial and technical challenges of higher speed digital communication. You know the argument... ham radio operators used to be technically oriented, home constructors, but now they are just appliance operators. An ancillary argument is that this is somehow all related to the no code license. This bit of folklore has been repeated so often that almost everyone has begun to believe it. Like most conventional wisdom, it is in large part incorrect.

A couple of years ago at the Rochester, NY Hamfest I had a discussion with Steve Ford, WB8IMY of the ARRL. He raised an interesting point, namely, what would we do with the speed if we got it? Suppose, hypothetically, that overnight every 1200 baud packet TNC/radio

combination in the country was magically converted into a 9600 baud station (or a 19,200 baud station). How would that change anything? You would still have a systern based largely on text BBS's handling roughly 500 messages per day, where most users read and compose messages online. Few people can read faster than a solid 1200 baud packet connection provides data, thus for these applications they see little point in pulling in data any faster. As long as the basic data infrastructure remains the same there is no incentive for speed.

So we have an interesting conundrum: End users don't want to invest in higher speed packet unless there is something interesting and fundamentally different that they can do with it. On the other hand, interesting new applications that might utilize higher speed packet are not being developed because of the lack of potential users. As a result many former users of packet have packed up their TNC's and put them away because they find the Internet more interesting.

Is this really inevitable? For a few months there I thought so. Now I'm not so sure. Any solution to the current packet problem must present a clear migration path that will provide interesting new applications for 1200 baud packet users that will both capture their imagination and provide some real justification for the expense and trouble of moving to higher speeds. That is, this new system must not be predicated on the assumption that we will all operate at faster speeds, rather, it must work reasonably well at 1200 baud. But it is equally important that it offer something genuinely different than what users are seeing on the current crop of packet BBS's

That's a tall order all right, but fortunately many of the building blocks for such a system are already within our grasp. First, we must build on our strengths. From a technological standpoint, what advantages does radio offer over wires (or fiber... the technology that lies behind the Internet)? I think that there are two clear advantages to radio: mobile operation is much easier with radio and radio is inherently a point to multi-point technology. For our purposes here it is the latter item that is significant. Data communication that occurs at slower speed can actually be more efficient than higher speed communication if the slower speed method also transmits data to many users all at once. If you need to communicate, say 100K of information to 100 different people, 1200 baud is actually faster than a 28,800 baud modem (and even faster than 56,000 bps frame relay technology) if the 1200 baud signal can reach everyone at once, but the higher speed signal has to retransmit the information separately to each of the 100 destination stations.

Imagine how little would be accomplished by a AM broadcast news station if it had to read the news to each listener individually rather than broadcasting it to everyone at once. Imagine how poorly our amateur radio voice nets would run if the net control station had to repeat each transmission over and over again, once to each station that checked in. Well, folks, this is the way we do packet radio. If 100 of us are interested in reading the same message we each connect to the BBS and download the message. None of us hear the other's download, so we must each ask for the transmission separately.

With the exception of a few of the newer technologies, the Internet works exactly the same way. When you connect to a chat server, or a CUSEEME server, the server receives what each connected station sends and then retransmits it individually to each of the stations involved in the conversation. If there are twenty participants in the chat, the server must repeat the information twenty times.

With radio there is no inherent reason that we have to be so inefficient. In fact, not all of packet radio is this inefficient. The digital satellites, for example, have for years been relying on an orbiting server that broadcasts packet frames. They can be received at many, many stations all at the same time. If one or more of the stations misses something, it can make a request for the server to fill the hole, rather than retransmit the whole file. Believe it or not, this is not the way the system was originally envisioned. The first orbiting packet station was aboard the Japanese satellite JAW. It worked pretty much like our terrestrial BBS's work, only with far fewer commands. The original plan for the microsats was similar... it was to orbit a WORLI-compatible BBS (which was then the world standard). I was at the annual AMSAT meeting in Des Moines when Harold Price (NK6K), one of the principle software designers for the spacecraft, announced that the WORLI-compatible system wouldn't work. He did some quick calculations about the number of potential users and the fact the satellites would only be visible for a maximum of about 15 minutes per pass and concluded that if everyone was to try to connect to the satellite individually, very little data would actually get through. This gave rise to the concept of broadcasting files for everyone to receive at the same time. This is essentially the system that is still in use today.

For the 1994 AMSAT meeting in Houston, I presented a paper that argued that this same technology could be put into practice on terrestrial BBS systems. I included some code I had written to construct AX.25 frames and broadcast them through a G8BPQ switch (which was the technique that was commonly used by BBS's at the time). At the end of the article I invited anyone who wanted to, to steal the idea and make it into a workable system.

Over the past year my thoughts have been returning to the proposal I made in 94 and the comments that Steve Ford made last summer. Sometimes it takes a while for all the pieces of an idea to come together, but I think they finally have. What we need is both a means of transmitting far more data to users in a shorter period of time, and a rethinking of the content of those transmissions.

So starting first with the underlying technology, it is necessary to be able to communicate a lot more data. This must be done, however, without requiring the end users to make a substantial investment in new equipment. I think sending frames in an unconnected mode (as occurs with the satellites) is the clear answer here. Not only does the server communicate with more than one station at time, but a lot of time is also saved by getting rid of the acknowledgement packets.

Currently a packet station sends out a transmission of roughly 1000 bytes and then waits for the other station to acknowledge receipt before sending more. Those who have done the calculations say that this causes the 1200 baud data rate to achieve a throughput rate under the most optimal conditions, with no nodes or digipeters, of about 600 bits/sec. In the real world the throughput is most often substantially less. After taking into account overhead, 600 bits/sec will give you about 60 bytes (or characters) per second. That's about 600 words per minute, which is a pretty good reading speed, but not great for file transfers.

The server that I've written (dubbed "HamWeb", for reasons that will become clear shortly) has achieved measured throughput rates of over 115 bytes (or characters) per second. That is, the speed is about twice that of the conventional packet channel when the conventional channel is running at its fastest. And this is using the same basic 1200 baud equipment that most packeteers already own. But this is not where the real throughput boost occurs. It is even more important that the HamWeb Server allows these files to be received by many stations at the same time. The more receivers that are turned on at once, the more efficient the system becomes.

The basic technology building block, then, is a server that pumps data continuously out a serial port and into a TNC. No receiver needs to be hooked to the TNC, because if the server is going to receive anything, it will be on some other channel (with another radio and TNC) instead of the one where the continuous transmitting occurs. However, the transmitter to be used in this application had better be able to withstand a very high duty cycle, because it will essentially be always on. It is possible that commercial transmit strips my be useful for this purpose. Obviously, since the receiver is being omitted, the T/R switching can be dispensed with as well. The server software takes the files that have been specified (or entire directory structures), breaks them down in to smaller parts, adds some additional information concerning the filename, size, etc. as well as a "checksum" to ensure that the data is received error-free, and sends the data to the TNC.

On the user side, the minimum required station is an FM receiver and a TNC that is capable of running in "KISS" mode (virtually all of them are). What's more, it is possible (though I've not done it) to write an end user program that would use the station computer's sound card instead of a TNC. In this instance, one could simply buy a $50 receiver and run the audio output into the mic input of the sound card. If the information available through this system is sufficiently interesting, I think it is reasonable to think that a significant number of non-ham hobbyists might set up receiving systems to monitor these transmissions in the same way that they currently use scanners to monitor repeater traffic.

The software running on the client computer generally will run in the background and take the incoming packets and reconstruct the files so they appear just as they did on the server. If part of a file is missing, the client keeps track of the pieces that it needs and grabs them the next time they are transmitted, until a complete file is constructed. I assume that the typical user will simply leave his receiver on all the time and collect data around the clock.

So that's the basic file sending/receiving engine. But as Steve Ford said, if all we've done is radically speed up the transmission of data without changing the nature of the service . . . well, what's the point? For example, those who operate the large BBS's in our area tell me that they currently receive about 500 bulletins a day. Assuming an average bulletin is 1000 characters in length (many are longer, but most are shorter), the system described above could transfer every one of those bulletins to every receiving computer in its area in under one hour. I don't mean a listing of the titles of the messages (such as is done by TPK or WinPacket) but rather the entire contents of all of the bulletins that an average BBS receives in 24 hours. Even if we wish to stick with the contents currently available on packet radio, we'll have to think of something for these servers to do the other 23 hours of the day!

Communication on the Internet is based on a language called "Hyper-Text Markup Language" or HTML. Those who have designed their own web pages will be intimately familiar with this language. Once upon a time it was relatively difficult to do this because HTML itself is rather arcane. However, many new software packages (some available for free) have become available that allow folks with absolutely no background in computer programming to design very attractive and functional web pages.

When I first began to explore HTML the thing that most impressed me was its efficiency. Beautiful pages that appeared to have bit-mapped graphics could be put together in only a few kilobytes of data. This is possible because the formatting of the data is actually done on the end user machine, not on the web server. Try the following experiment to see what I mean. Go to one of your favorite web pages with Netscape. Click on "File" and then click on "Save As". Give your file and name and save it. Now switch from Netscape to a listing of the directory where your file is saved and examine the size of that file. I think you will be surprised at how small it is. Graphics, of course, require additional space, but most web page graphics are highly compressed versions of rather small files.

When I first discovered this, what came to mind immediately was that HTML seemed ideally suited to amateur radio applications. It seemed to me to be obvious that the next generation of BBS software would use HTML or something like it to provide information in a much more attractive and useful way. However, I didn't pursue this because I had neither the time nor the expertise to design a new BBS system around HTML. Furthermore, suppose you could design such a BBS. Given the huge installed base of BBS's using the forwarding scheme that traces its routes back to W0RLI's original code, unless this new, improved BBS system was completely compatible with that forwarding system, it was probably doomed to obscurity.

I believe that there is now a possible transition path between older BBS's and new systems that can provide a wide array of services. The key is to use Netscape (or some other browser) as the principal terminal program for utilizing the packet radio store and forward system. The kids down in Mountain View and up in Redmond have spent a lot of time and effort creating these web browsers. We should simply use their technology building blocks for another purpose. This

is possible because "Web Browsers" are not simply programs for browsing the world wide web. They are more general purpose programs which, among other things, can be used for viewing any kind of HTML document.

What I am suggesting is that we use a HamWeb broadcast server to provide local distribution of amateur-radio related information in the form of world wide web pages. End users would run the HamWeb client software in the background to continuously receive these files and dump them into a directory on their hard drives. The HTML documents would then be viewed using any standard WWW browser. Every morning you could wake up and have a completely new electronic amateur radio magazine delivered to your computer. It would have the type of attractive formatting and color that we are used to seeing in web pages. It could have graphics, pictures, even sound. What's more, it could also contain links to world wide web sites, so that if you happened to be simultaneously hooked into the web, you could move seamlessly between information that was provided by radio and information that was available on the web. And in addition, the information that came in by radio would appear to the end user to show up on the screen almost instantaneously, since it would be read directly off his hard drive.

As an isolated system this would provide a very interesting experiment. However, no local community of amateurs is going to develop enough content to provide interesting reading on a regular basis. Instead, we must find a way to layer this system on top of the existing packet forwarding system. If the pool of potential producers of web pages to be distributed in this fashion is extended to the entire globe, then we would begin to see enough innovative content to make such a system worthwhile. Standards would need to be worked out for identifying and forwarding HamWeb pages to insure that file names are not duplicated. I think the way to do this would be to have each set of web pages forwarded as a unit, an encapsulated compressed file. The filename would consist of the home or index page for the set with the originating BBS's callsign tacked on (and perhaps a serial number). Then as the set is received by each server in the network it would be placed into its own subdirectory. Since the HamWeb client/server software causes the entire subdirectory structure to be duplicated on each of the client machines, there should be no problem with individuals writing pages with identical names.

The software that I've written provides a general purpose mechanism for moving data from a central server to a lot of client computers, all at the same time. I have tried to write this program as generally as possible to lend itself to a wide variety of applications beyond serving up web pages. A number of "hooks" have been built in to allow the server to interface with other programs to initiate file broadcasts.

Another interesting application, for example, would be to use the server on a satellite ground station to replicate all of the directory files and message files from the PACSATS on computers in a given region. Thus, anyone who wanted to could run any of the available PACSAT groundstation programs without actually putting up a satellite station themselves.

In emergencies, a HamWeb server would provide a mechanism to distribute large quantities of information (including graphics and other multimedia files) over a wide region. Most importantly, the receiving points would not even have to be equipped with transmitters. Scanners would be adequate to provide the latest updates of information.

The HamWeb Server and Client software are now available in beta version on the TAPR host at:

www.tapr.org/~wa0ptv

or by ftp at:

ftp.tapr.org in the pub/wa0ptv directory

Appendix: Technical Documentation for the HamWeb Server

The HamWeb server is a program designed to broadcast information using amateur packet radio. I have written a client program that collects these broadcast files which runs under Windows 95. However, others may wish to collect the data for other platforms or may wish to construct improved client programs for Windows 95. As a result, this appendix will provide the specifications of the packets that are transmitted by the HamWeb server.

The HamWeb server communicates with the TNC using the KISS protocol. I am assuming that this protocol will also be used on the client side. The KISS protocol starts each packet with a CO and ends each with a CO. To ensure that CO does not occur anywhere else in the packet (thereby causing the client program to think the packet is over) it is necessary to make some substitutions as the data is transmitted. The client program is therefore responsible for undoing these substitutions on the receiving end. For more complete documentation of this process see the Kiss docs on the TAPR website by Mike Chepponis and Phil Karn. In constructing a HamWeb client, your responsibilities with regard to undoing the substitutions will be discharged if you scan through the packet for the byte DB. In any instance where you find the DB byte, if the next byte is a DC, you should substitute the value CO. If the byte that follows the DB is a DD, you should leave the DB and eliminate the DD. Note in both these cases you are substituting a single byte where there were previously two bytes, so the overall length of the packet will decrease. After the substitutions are completed, you will have the raw data from which you may proceed.

THE HEADER

The corrected data from the TNC may usefully be thought of as being in two parts: An AX.25 header and the actual data. The first byte, of course, is a CO. This is followed by a 00 byte. The 00 byte indicates that the packet contains data rather than a command to the TNC. The next 16 bytes of the packet contain information on the source and destination callsigns and ssid's. Generally they can be discarded. If you are interested in the formatting of this information see the AX.25 protocol document on the TAPR web pages or my paper in the 1993 Proceedings of the AMSAT-NA Space Symposium. From the standpoint of receiving broadcast files, unless you are trying to keep track of the callsign of the server, you may throw away this entire 18 byte header.

THE DATA

That leaves the data. For reference purposes I will call the first byte of the data portion byte 0 even though it is not the first byte received from the TNC. I'm assuming here that you have either discarded the header information or extracted what you needed from it and then discarded it. The first four bytes of the data portion (bytes O-3) are a checksum. The checksum is constructed by adding the actual values of the rest of the bytes in the data portion of the packet starting with byte 4. This will allow you to test to see if the data that you are receiving has become corrupted The

checksum (as well as all other numerical values) are expressed in the standard format of least significant byte first. Thus, the four byte series:

12 AB 28 00

would translate to an decimal value of:

2,665,234

Bytes 4 and 5 are a serial number. Each file is transmitted 1024 characters at a time. The server breaks the file up into 1024 character sections and gives each one a serial number. The first section is serial number 00, the second 01, etc. I have used serial number rather than the actual file offset to reduce the overhead of the system. A two byte serial number will allow single files as large as 64 megabytes to be transmitted. To accomplish this using offsets instead, I would have to use a 4 byte value. The next four bytes (bytes 6-9) are the file size. The following four bytes (bytes 10-13) are the time the file was last modified. This is recorded in the standard format of the number of seconds since Jan 1, 1970. The time is included in the broadcast so that the client software can check whether an existing copy of this file is older than the one currently being received. Thus files can be updated only when they are newer than the existing version. One use for this would be if the server maintained an "index" page of the files available. Thus when the sysop modifies the index page to add or delete files, this page would automatically be modified on all the client computers as well.

Byte 14 starts the path/filename. When the server is replicating directories rather than just transmitting files, the entire path is transmitted in this field rather than just the filename. In writing client software, you are responsible for parsing this path/filename and creating the directory for the file if it does not already exist. I arbitrarily limited the path/filename to 128 characters. You will know that the server is transmitting files to be placed in directories because the path/filename will start with a "\".

Even when the server is simply transmitting files, you should not assume that filenames will be in an 8.3 format. Web page filenames are commonly much longer than this. Longer filenames are supported by Win95, WinNT, Mac system 7.5, and virtually all flavors of UNIX. Thus it seems silly not to use this capability. As a result however, you do not know how long the filename will be before you receive it. I have added an FF character to mark the end of the filename. All characters starting with byte 14 and ending with the character before the FF constitute the filename.

All characters after the FF are the actual data contained in the file. For the most part, this will be 1024 characters, since the file is broken into those chunks. However, you can not assume that the data will be exactly 1024 characters long, since the last chunk of any given file is likely to be less than 1024 characters. The packet ends with a character CO, which also may be discarded. Note

that you must identify where the packet ends before you perform the DB/DC to CO conversion or you may be fooled into thinking the packet ends before it really does.

OBSERVATIONS AND HINTS

Some may wonder why I didn't simply use the PACSAT protocol that is currently used on amateur satellites. There are a number of reasons. First it contains a whole lot of information that is simply never used for anything. This is not a criticism of the folks who wrote this protocol. At the time it was written we had absolutely no operational experience with it and it probably seemed like many items (such as the number of times a file had been downloaded) were useful. The PACSAT protocol creators were also writing a system that could be implemented on DOS computers (the most popular operating system for PC's of that day) and so shorter filenames (which in the case of the PACSATS were numbers) were essential. Because I expect that most people will view these files with a web browser rather than a terminal program, my only goal in this project is to transfer files from the server to the clients and provide enough information to manage the files on both systems. Thus, I've opted for a substantially simpler protocol.

In writing your client software you should assume that any character could be lost in transmission. Thus, any time you are filling a array by waiting for an ending character (such as waiting for a CO to know the packet has ended, or waiting for an FF to know that the filename is ended) make sure that if that character is lost something will stop the array from exceeding the limit you set for it. Otherwise the results could be very nasty.

Finally, make certain that you keep careful track of the length of the packet. Every time you replace a DB/DC with a CO or a DB/DD with a DB you must make sure you reduce the length of the packet by 1.