WINMOR Phase 2: Demonstration to Deployment

Rick Muething, KN6KB/AAA9WK; Winlink Development Team
rmuething@cfl.rr.com

Abstract:

In September 2008 I introduced WINMOR (**WIN**link **M**essage **O**ver **R**adio) at the ARRL/TAPR DCC in Chicago [1]. Success with that preliminary test version of WINMOR sparked interest in the protocol and the creation of an active Yahoo WINMOR group. This in turn accelerated the development of client software, RMS Express, needed for on-air beta testing. This paper addresses the challenges of completing the development and deploying viable client and server software supporting WINMOR. The advanced layered Viterbi Trellis Code Modulation and Reed-Solomon FEC used in WINMOR are also summarized. A novel Virtual TNC model that implements the protocol is described along with some of the applications that support WINMOR based on this Virtual TNC.

Key Words:

WINMOR, SCAMP, Virtual TNC, ARQ, TCM. Winlink 2000, RMS Express.

Background and Motivation….What was the need?

First, some brief background and the motivation for the WINMOR project. Pactor has been the benchmark for HF messaging ARQ (Automatic Retry reQuest) protocols since its deployment in the late 1980s. The Pactor 2 and Pactor 3 protocols still yield some of the best throughput available over a wide range of propagation channels. SCAMP [2] had demonstrated sound card ARQ on a PC was possible but identified some critical requirements that were necessary for practical deployment. But proprietary Pactor TNCs can be expensive and too often that restricts the access to these efficient HF modes. Those seeking a lower cost way to access HF messaging kept asking, "Can't you do something that performs like Pactor with a sound card?" Today's higher speed PC's now offer an opportunity to do much of the advanced signal processing required for good HF performance. The goal of the WINMOR project is to provide a sound card ARQ mode (error free delivery of data) that approaches Pactor 2 and 3 in performance on a modern PC using standard PC sound cards. Operating system issues (both Linux and Windows) combined with existing Pactor timing constraints and proprietary issues make emulation of Pactor on a PC impractical. WINMOR was developed from the ground up optimized for data messages and to be compatible with the performance and timing constraints of a modern PC and operating system. Another objective of WINMOR was to work with most existing PC sound cards and sound card interfaces so that WINMOR could be used alongside other popular sound card modes.

At the time of introduction of WINMOR at the 2008 DCC most of the alpha testing had been limited to HF simulators using identical sound cards (SignaLink USB). The simulator allowed testing and optimizing the algorithms over numerous S/N levels and propagation channels. This repeatability of the HF simulator made it possible to go back and make accurate comparisons of different implementations. But like any new system it can't really be tested and wrung out until it is deployed in into the real world….time for over-the-air beta testing!

After the warm glow of initial enthusiasm comes the cold reality of making it work!

It was obvious we needed a vehicle to test the WINMOR protocol in a real message over-the-air environment. Vic Poor, W5SMM my mentor and co-developer came up with a simplified but quite useful Windows radio client program called RMS Express [3]. This allowed creating and managing messages, adding attachments etc. and interfaced to the WINMOR codec software. RMS Express was simple to learn, easy to setup and use and since it didn't (yet!) have all the bells and whistles of more complex radio clients allowed us to concentrate on the primary objective… testing, debugging and optimizing the WINMOR protocol on the PC. Since there were no servers available yet RMS Express would have to operate both as a server and client allowing what we called peer-to-peer testing. (Forwarding messages between two RMS Express clients)

One of the first setbacks that came with the initial peer-to-peer radio tests was the realization that all sound cards were far from equal. Specifically the critical sample rate varied on some otherwise quite useable sound cards by over 1%. One percent may not sound like much of an error if you're playing a MP3 file but in a multi-carrier PSK scheme like WINMOR it made it impossible to do reliable decoding Furthermore wide band calibration/correction of the sound card sample rate using software with that large error was simply not practical in WINMOR without a significant impact to throughput. One doesn't have to spend much time looking under the hood of hardware DSP modems to find they universally incorporate some form of precision temperature-stabilized time base (often 10 ppm or better). That's more than 1000 times better than the accuracy observed across typical sound cards! This prompted a relatively broad test of many sound cards across all available sample rates which came up with an unexpected observation. Most of the variations in sample rate were associated with the specific sample rates of 8000, 11025 and 44100 Hz. Sample rates of 96000, 48000 and 12000 almost universally were more accurate with most sound cards being within a few hundred ppm of the target rate…an error small enough to be corrected. The large variations were most likely due to a combination of the chip and driver implementations. The bad news was all the algorithms, carrier spacing, symbol counts, filters etc for WINMOR had all been done for the 8000 Hz sample rate (62.5 baud). It became obvious what was necessary was to select one of the more accurate sample rates and redo the software based on that. I remember Vic and I reaching that same conclusion knowing how much code it would impact. It took over a month of intense effort to rewrite, test and re optimize the software to work with a new 48000 sample rate on capture and 12000 rate on playback….but in the end this solved virtually all the sound card compatibility issues. Sampling at the 48000 rate also made it possible to do direct I Q (quadrature) sampling (keeping half of the samples) yielding I and Q channels sampled at 12000 Hz. That saved processing time and made balanced mixing (tuning) more accurate than with the previous 8000 Hz scheme.

The original demonstration of WINMOR at the 2008 DCC used only Reed-Solomon block type Forward Error Correction (FEC). While this was efficient and helped it did not achieve the degree of robustness desired or comparable to modes like Pactor 2 and 3. The solution for this was to leverage from the same technology used to implement telephone modems. This bandwidth-efficient signaling scheme is named Trellis Code Modulation or TCM [4].

Trellis code modulation adds one additional bit to each PSK symbol doubling the number of constellation sites. While this "tighter" constellation increases the raw bit error rate (assuming constant transmit power) with proper coding of the added bit and optimized bit to phase mapping a significant *net* improvement in bit error rate over the uncoded case can be achieved. To keep the software implementation manageable and to capitalize on the efficient Viterbi decoding algorithm WINMOR actually uses what is called Pragmatic Trellis Code Modulation [5]. PTCM uses a standard convolutional code in place of the slightly more optimum Ungerboeck code. WINMOR uses the standard R=1/2, K=7 (NASA Voyager) convolutional code which is compatible with the Viterbi decoder. This combination yields a coding gain within a couple of tenths of a dB of the optimum Ungerboeck code of the same length. The public Viterbi decoder C code of Phil Karn, KA9Q simplified the software task though it was still necessary to translate the C code to VB.NET for stability in our VB.NET application. To understand how the Trellis code modulation works we can look at the example of WINMOR's 8PSK mode. In WINMOR 8PSK two information (user) bits are transmitted per 8PSK symbol. One of the user bits U_1 is used to generate (using the convolutional encoder) two Code bits, C_0 and C_1. The protocol then sends three bits (the uncoded user bit U_0, plus the two convolutional code bits C_0 and C_1) as a 3 bit 8PSK symbol. The uncoded bit U_0 selects either 0 or 180 degrees to be added to the phase of the coded bits. This mapping of the uncoded bits to phase angle is critical in TCM. Figure 1 shows the complete 8PSK PTCM encoder used by WINMOR.

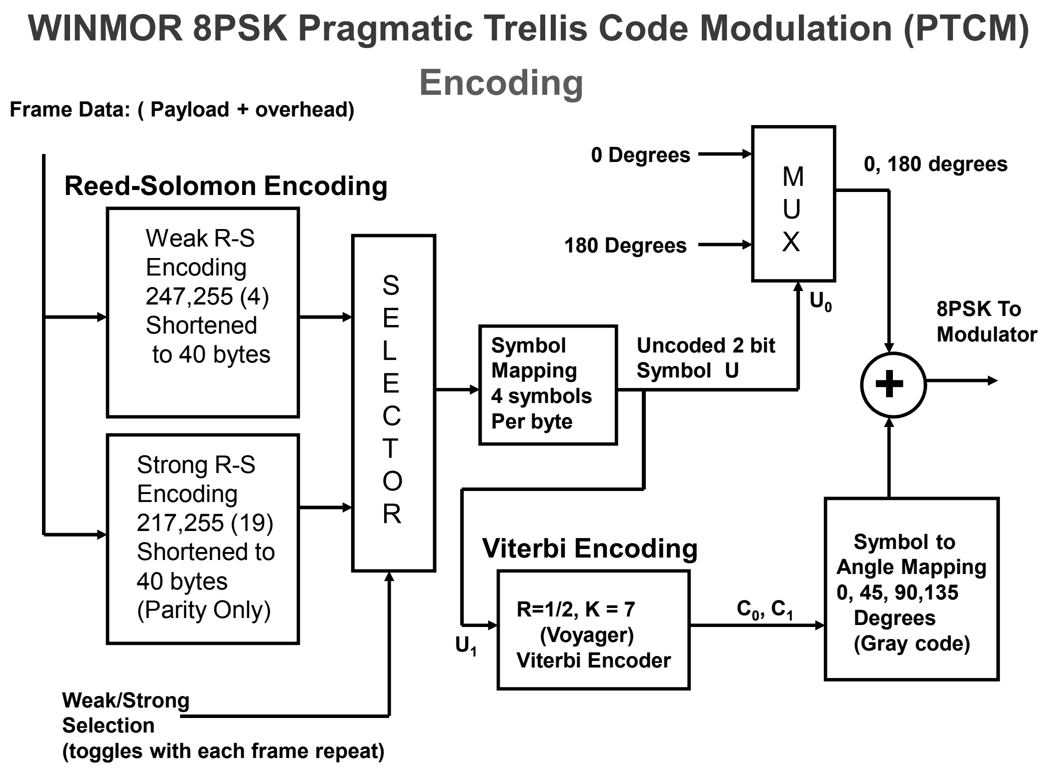
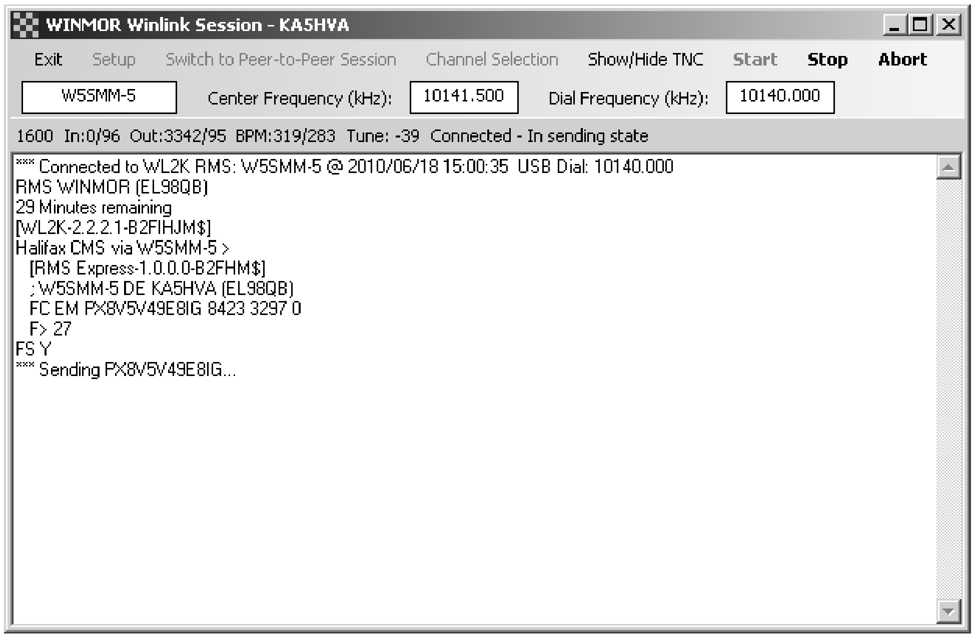

WINMOR 8PSK Pragmatic Trellis Code Modulation (PTCM)
Encoding
Frame Data: (Payload + overhead)
Reed-Solomon Encoding
Weak R-S Encoding 247,255 (4) Shortened to 40 bytes
Strong R-S Encoding 217,255 (19) Shortened to 40 bytes (Parity Only)
SELECTOR
Weak/Strong Selection (toggles with each frame repeat)
Symbol Mapping 4 symbols Per byte
Uncoded 2 bit Symbol U
0 Degrees
180 Degrees
MUX
U0
0, 180 degrees
8PSK To Modulator
+
Viterbi Encoding
U1
R=1/2, K = 7 (Voyager) Viterbi Encoder
C0, C1
Symbol to Angle Mapping 0, 45, 90,135 Degrees (Gray code)

Figure 1 WINMOR 8PSK PTCM Encoder

The Decoding operation is somewhat more complex. Recall that the received demodulated differential phase information is corrupted by noise, Doppler, phase distortion, and frequency offset induced errors. The solution developed to decode this "noisy" PTCM is to use a mechanism to first null out the uncoded bit (U_0 in the encoding diagram) and then process the coded bits through a conventional Viterbi decoder. The contribution of the uncoded bit is removed by multiplying the demodulated phase by 2 modulo 360 degrees. Once the soft decision Viterbi decoding [6] is complete on the doubled phase angle we have the best estimate of the coded bits C_0 and C_1. Those decoded bits are then "recoded" back to phase angles using the same mapping as the encoder and are then subtracted from the raw phase of the received signal to yield (now) the best estimate of the uncoded bit U_0. Since the uncoded bit phases are widely "spaced" (180 degrees for 8PSK) they have an inherently lower bit error rate and don't require the strong FEC of the Viterbi decoder. A decoder which implements this in WINMOR for 8PSK is shown in Fig 2.

As Figures 1 and 2 also show, WINMOR layers this Viterbi encoded inner layer with an outer layer of weak or strong Reed-Solomon block encoding for additional FEC strength. This layering of block encoding outside convolutional encoding is a well used technique in advanced codecs which capitalizes on the best capabilities of each code type yielding good efficiency and error rates lower than what could be achieved using either code by itself.

A typical communications session looks like this WINMOR session (currently in the process of sending a message)…

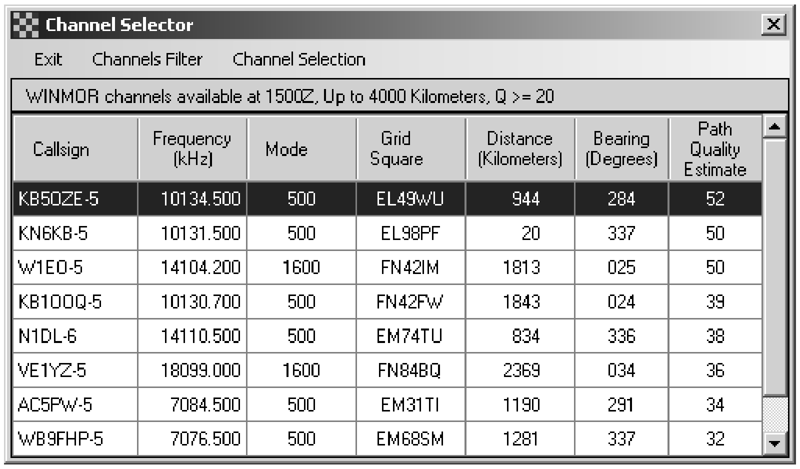

WINMOR Winlink Session - KA5HVA
Exit
Setup
Switch to Peer-to-Peer Session
Channel Selection
Show/Hide TNC
Start
Stop
Abort

W5SMM-5 Center Frequency (kHz): 10141.500 Dial Frequency (kHz): 10140.000

1600 In:0/96 Out:3342/95 BPM:319/283 Tune: -39 Connected - In sending state

```
*** Connected to WL2K RMS: W5SMM-5 @ 2010/06/18 15:00:35  USB Dial: 10140.000
RMS WINMOR (EL98QB)
29 Minutes remaining
[WL2K-2.2.2.1-B2FIHJM$]
Halifax CMS via W5SMM-5 >
  [RMS Express-1.0.0.0-B2FHM$]
  ; W5SMM-5 DE KA5HVA (EL98QB)
  FC EM PX8V5V49E8IG 8423 3297 0
  F> 27
FS Y
*** Sending PX8V5V49E8IG...
```

From this window you can select a channel to an RMS WINMOR server with a double click…



WINMOR channels available at 1500Z, Up to 4000 Kilometers, Q >= 20

Callsign	Frequency (kHz)	Mode	Grid Square	Distance (Kilometers)	Bearing (Degrees)	Path Quality Estimate
KB5OZE-5	10134.500	500	EL49WU	944	284	52
KN6KB-5	10131.500	500	EL98PF	20	337	50
W1EO-5	14104.200	1600	FN42IM	1813	025	50
KB1OOQ-5	10130.700	500	FN42FW	1843	024	39
N1DL-6	14110.500	500	EM74TU	834	336	38
VE1YZ-5	18099.000	1600	FN84BQ	2369	034	36
AC5PW-5	7084.500	500	EM31TI	1190	291	34
WB9FHP-5	7076.500	500	EM68SM	1281	337	32

…and start a link with the selected station. Channels are ranked by relative forecast path conditions using the built in propagation forecaster.

Once connected, all pending outbound and inbound messages are automatically exchanged with a WL2K CMS site and the link is closed. A minimum amount of air time is used.

Problems Users Have

The problems users have installing and running RMS Express can, for the most part, be summarized as this:

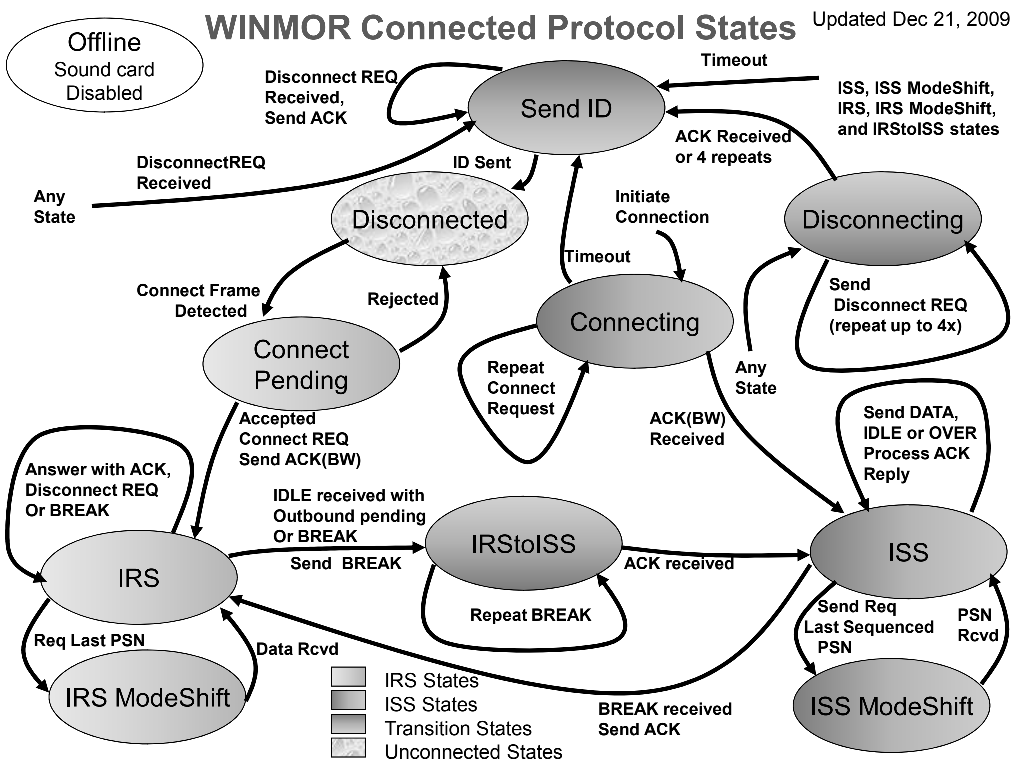


Figure 3. The WINMOR Protocol state diagram

One of the final challenges in the path of beta testing WINMOR was interfacing WINMOR and RMS Express to the many different sound cards and radios. While radio control isn't an absolute necessity it was very helpful during beta testing to quickly and accurately set the radios to specific beta test frequencies (what we often called "watering holes"). We became convinced that no two radio models are equivalent when it comes to computer control! And while all sound cards and interfaces perform the same basic functions there are unique ways each might key the transmitter and adjust drive and receive levels that have to be accommodated and documented.

Deploying and Providing Access to the WINMOR Protocol

As the beta testing progressed and developers and beta testers gained confidence in the new protocol one issue repeatedly came up. To be a truly viable protocol we needed a way to distribute the WINMOR protocol and make it easy for other application developers to use. While the public specification for WINMOR provides all the information necessary to write the WINMOR codec the complexity of the real-time code and the challenge of verifying each implementation of the protocol made it unlikely many programmers could easily duplicate or integrate the WINMOR TNC code. The solution to this was what we called a virtual TNC. This virtual TNC is a stand-along program which implements the WINMOR HF modem making it appear similar to a conventional hardware TNC. The Virtual TNC has ports through which commands and data are passed like hardware TNCs. The WINMOR TNC even has a “front panel” which emulates the “flashing LEDs” of a hardware TNC as shown in Figure 4.

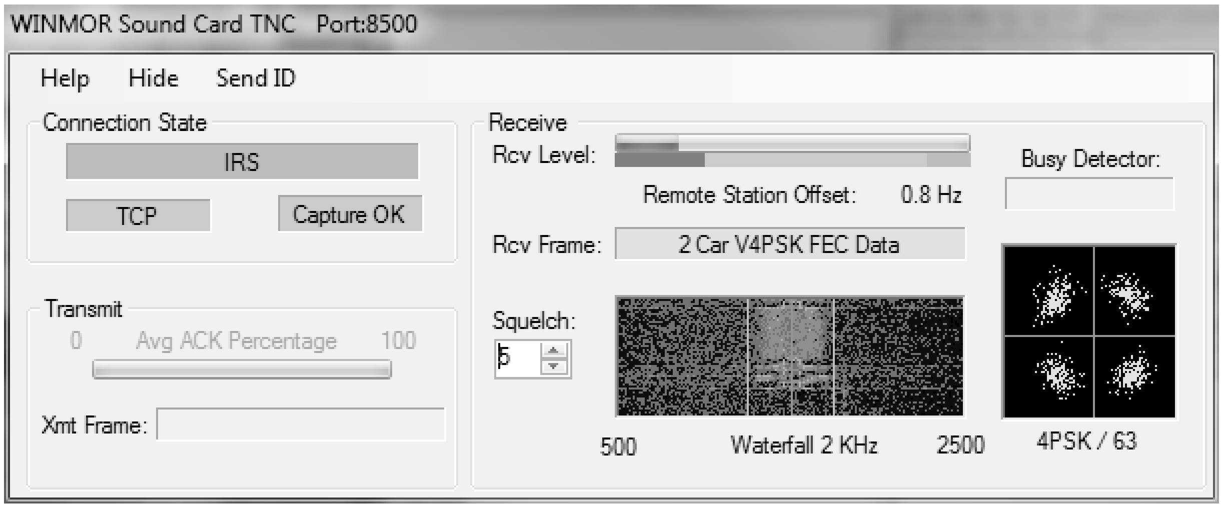


Figure 4. The virtual WINMOR TNC “front panel”

Implementing the Virtual TNC required adapting all the RMS Express WINMOR code into a stand-alone package that interfaced via simple TCP commands and data. . In the virtual TNC software it was easy to implement features too costly on hardware TNCs like a waterfall display and a constellation diagram. These allow the operator to quickly identify signals and see the quality of the received nPSK or 4FSK symbols. Once this effort was completed and the virtual TNC documented it became straight forward for other developers to integrate the WINMOR protocol into other applications. To date there have been several successful integration efforts based on this virtual WINMOR TNC. At the time of this writing (June 2010) this includes:

RMS Express, Paclink, and RMS WINMOR by Vic Poor and Rick Muething

BPQ32 BBS and Switch by John Wiseman, G8BPQ

SNOS (Protocol linking) by Hank Oredson, W0RLI

AT-AUTO Keyboard client by Howard Zuckerman, N3ZH.

Spinoffs…Adapting and optimizing WINMOR for other applications

Two things became evident during the beta testing of WINMOR:

1) To get good solid performance on HF you need to optimize virtually everything.
2) There are always some innovative hams thinking up new applications and uses for something that exists.

This was the case with the WINMOR virtual TNC. For example some wanted to use it on VHF and through voice repeaters. Some wanted a keyboard chat mode for fun QSO's when they weren't forwarding messages. While both of these examples *could* be accommodated with the existing WINMOR TNC the results may not be comparable to a protocol optimized specifically for a specific frequency, bandwidth, mode, or operating activity. The flexibility of software and the way the Virtual WINMOR TNC is partitioned and implemented make it relatively easy to change parameters like bandwidths, modulation modes, symbol rates, carrier placement, error coding etc. This means that it is no longer necessary to live with sub optimal solutions that were necessary in the days when we had to adapt those surplus teletype machines or Bell 300/1200 baud modems. One recent example of this "application targeted protocol" is a new keyboard oriented protocol called V4 which uses the WINMOR TNC concept to implement a simple yet very robust keyboard compatible non ARQ modem that is optimized for narrow bandwidth (200 Hz) applications at normal keyboard typing speeds (55 WPM). V4 uses a novel application of Viterbi encoded 4FSK for excellent robustness even with poor multipath propagation. As we have the opportunity to explore and adapt WINMOR to other applications it is likely we'll see more of these "targeted protocols" spring up and be adopted.

Summary

Driving a new sound card protocol from demonstration through deployment turned out to be a bumpier road than initially anticipated. But like many difficult projects it is the entire process, the roadblocks, the setbacks and the solutions that become the real catalysts for progress. I am reminded of the words the great inventor Thomas Edison spoke when asked by a reporter, "Weren't you frustrated at the number of failures you experienced during your attempts at inventing the electric light?" "No, not at all", Edison quipped sharply, "I now *know* 150 different filaments that *don't* work!"

References:

[1] WINMOR … A Sound Card ARQ Mode for Winlink HF Digital Messaging; Rick Muething, KN6KB/AAA9WK; 27th ARRL and TAPR DCC 2008

[2] SCAMP (Sound Card Amateur Message Protocol); Rick Muething, KN6KB; 23rd ARRL and TAPR DCC 2004

[3] RMS Express = A Multimode Winlink 2000 User Client Program; Vic Poor, W5SMM/AAA9WL; 29th ARRL and TAPR DCC 2010

[4] Trellis Coded Modulation Tutorial; Charan Langton, 2004. http://www.complextoreal.com/chapters/tcm.pdf

[5] A Pragmatic Approach to Trellis-Coded Modulation; A. Viterbi, J. Wolf, E. Zejavo, R.Padovani; IEEE Communications Magazine July 1989.

[6] Data Recovery in Differentially Encoded Quadrature Phase Shift Keying; J. Bard, M. Nezami, M. Diaz; Mnemonics Inc. Melbourne, FL.