Raspberry Pi 9600 Baud TNC (TNC-Pi9k)

Mark Griffith, KDØQYN
501 S. Oak St., Union MO 63084
mark.griffith@pigate.net

Abstract

A new TNC has been developed by John Wiseman, G8BPQ, for the Raspberry Pi. Using the same form factor as the Pi and the existing TNC-Pi2, also developed by John, a new device was made that gives the Raspberry Pi and any packet applications that run on the Raspberry Pi operating systems, the capability to transfer data at new higher speeds, while still being affordable for most hams. This paper discusses the hardware and software issues of this new device as well as presenting detailed data on high speed (for amateur radio) data transfer tests.

Introduction

Several Terminal Node Controllers (TNCs) have been produced over the years that have given amateur radio operators the ability to communicate via digital packet data modes. Models such as the AEA PK-232 or the Kantronics KPC3+ have been in use for decades with great success. Amateur BBS stations reached via packet radio were popular before the advent of the global Internet. TNCs are still in use today for Automatic Packet Reporting System (APRS) reporting, where they normally link a personal computer with an amateur transceiver to transmit data. The recently developed PiGate and PiGate RMS system use packet radio to transmit email from within a disaster area when all other forms of data communication are down. This new board was of great interest to me for the PiGate project because of the potential to send and receive e-mail with attached files, such as low resolution photographs, via amateur packet radio. While this is possible at 1200 baud speeds, files larger than a few kilobytes take a very long time to transmit. In addition, long file transfers increase the possibility that something will happen to the digital radio link and it will fail, resulting in failure to send the message.

Packet Data Transfer Speeds

There are many digital data methods available, but they all limited by the available radio bandwidth and, to some extent, FCC rules. Amateur slow scan TV (SSTV) requires that the signal be no more than the width of a standard SSB voice signal (~3 kHz), and the data transfer rate is slow. However, many commercial digital data methods rely upon radio bandwidths in the several megahertz frame, such as with Over-the-Air Digital Television (OTA DTv) using the ATSC standard, which requires up to 6 MHz of bandwidth for a single TV channel and has a throughput of about 32Mbits per second. There is a lot of information crammed into a DTv transmission, which also uses special data compression techniques to fit as much data as possible into the signal, just like dial-up modem technology used years ago achieved 56Kbps over a telephone audio link designed for no more than 9600bps. Obviously, these bandwidths are not available for amateur radio packet which has to work normally in the VHF/UHF frequencies, and within the very narrow bandwidth of a Wide Band FM signal.

The capability to transfer data within a specific bandwidth is shown by the Shannon-Hartley theorem.

$$C = B\log_2(1 + S/N)$$

where

C is the channel capacity in bits per second;
B is the bandwidth of the channel in hertz (passband bandwidth in case of a modulated signal);
S is the average received signal power over the bandwidth (in case of a modulated signal, often denoted C, i.e. modulated carrier), measured in watts (or volts squared);
N is the average noise or interference power over the bandwidth, measured in watts.

Amateur radio uses between 10 to 15 KHz of bandwidth for FM modulation in the VHF and UHF bands due to Carson's rule, which states:

FM Bandwidth = $2(\Delta f + f_m)$ where Δf is the peak frequency deviation and f_m is the highest modulating frequency.

From these two formula, we can see that under the **absolute best conditions** with no noise, a 15 KHz wide FM signal can transmit at nearly 100Kbps, so 9600bps is easily within the bandwidth limits.

Packet Radio Modulation Methods

There are two main methods used to transmit a digital packet signal, Audio Frequency Shift Keying (AFSK) and Frequency Shift Keying (FSK). These modes have been in use for many years with RTTY. 1200 and 2400 baud packet use AFSK tones of 1200 Hz and 2200 Hz, which when transmitted on top of the FM carrier give the "ones and zeros" of the digital signal. This happens very quickly so the response of the radio audio circuits must be capable of this, and the transmitted signal must also fit into the FM bandwidth. This is not a problem with 1200 or 2400 baud as nearly all radios have the audio response necessary, although some cheap radios struggle to do this. However, with 9600 baud, using the AFSK method would likely exceed the audio frequency response of most radios and the transmitted signal would be beyond the accepted amateur bandwidth specifications. Because of this, a special radio that has the capability to connect directly to the internal modulator/demodulator is needed for 9600 baud operation. Not only will this configuration provide the frequency response needed at the higher speed by bypassing the radio audio circuits, but the transmitted signal bandwidth will be kept within the prescribed limits. The signal is now simply the shifting of the radio frequency in conjunction with the transmission of the digital ones and zeros (FM). Because there are no tones transmitted, our ears cannot hear the signal. A 9600 baud transmission just sounds like the static you hear when the receiver breaks squelch. However, not all radios are capable of doing this.

Radios and BER

I have researched what radios are available today that advertise as having a 9600 baud packet data port that gives the input directly to the internal mod/demod circuits. However, just having the required port is not enough. These radios must also have the capability to transmit and receive fast data signals without dropping too many bits here and there. Some radios are better at this, as indicated by their Bit Error Rate (BER).

Each time the ARRL Labs tests a radio that has a 9600 baud packet data port, they will (usually) publish the BER, which is simply the number of bits with errors divided by the total number of bits transmitted or received. A BER of 4x10-4 would mean 4 bits would be bad out of 10,000 sent or received. A BER of 3x10-3 means 3 bits out of 1,000 would be bad. This means that if using a 255 byte packet (2,040 bits), it is likely that no packets would be transmitted or received without at least one bit error, causing nearly every packet to be rejected. That would certainly cause data transfer to slow to a crawl. Note that these measurements are under ideal conditions with no radio noise or propagation variables effecting the data signal. Performance in the real world will most likely not be as good. The best BER published so far is less than 1 in 100,000, expressed as <1.0x10-5.

Some radios are better than others, and the old mantra of "you get what you pay for" is certainly true here, although there are some surprises. Below is a list of radios currently available and their BER as published by the ARRL. Not all are currently in production. All are dual band 144 Mhz/440 Mhz radios unless indicated:

Alinco

DR-150T	(2 meters only) Receiver:	<1.0x10–5.
	Transmitter:	1.6x10–3.
DR-605T	146 MHz Receiver:	<1.0x10–5.
	146 MHz Transmitter:	4.4x10–3.
	440 MHz Receiver:	<1.0x10–5.
	440 MHz Transmitter:	3.4x10–4.
DR-610T	146 MHz Receiver:	<1.0x10–5.
	146 MHz Transmitter:	4.2x10–4.
	440 MHz Receiver:	<1.0x10–5.
	440 MHz Transmitter:	5.6x10–4.

ICOM

IC-2500A (440 MHz and 1.2 GHz – No Data available)

IC-2720H	146 MHz Receiver:	<1.0×10–5;
	146 MHz Transmitter:	1.0×10–3.
	440 MHz Receiver:	<1.0×10–5;
	440 MHz Transmitter:	2.0×10–4.
IC-2800H	146 MHz Receiver:	<1.0×10–5;
	146 MHz Transmitter:	3.1×10–4.
	440 MHz Receiver:	<1.0×10–5;
	440 MHz Transmitter:	6.9×10–4.
IC-2820H	146 MHz Receiver:	<1.0×10–5.
	146 MHz Transmitter:	<1.0×10–5.
	440 MHz Receiver:	<1.0×10–5;
	440 MHz Transmitter:	<1.0×10–5.
IC-207H/208H	146 MHz Receiver:	<1.0×10–5;
	146 MHz Transmitter:	4.0×10–4.
	440 MHz Receiver:	<1.0×10–5;
	440 MHz Transmitter:	2.4×10–5.

IC-910H	146 MHz Receiver:	<1.0X10-5
	146 MHz Transmitter:	<1.0x10-5
	440 MHz Receiver:	<1.0x10-5
	440 MHz Transmitter:	<1.0x10-5
	1240 MHz Receiver:	<1.0x10-5
	1240 MHz Transmitter:	<1.0x10-5

Yaesu

FT-2600M (2 meters only)	Receiver:	<1.0×10–5;
	Transmitter:	1.8×10–3
FT-3000M (2 meters only)	Receiver:	2.3x10–5.
	Transmitter:	4.7x10–4.
FT-7100m	146 MHz Receiver:	1.6×10–4.
	146 MHz Transmitter:	1.4×10–2.
	440 MHz Receiver:	8.2×10–5.
	440 MHz Transmitter:	1.7×10–2.
FT-7800R	146 MHz Receiver:	2.1×10–5;
	146 MHz Transmitter:	3.5×10–5.
	440 MHz Receiver:	1.7×10-5;
	440 MHz Transmitter:	<1.0×10–5.
FT-7900R	(BER same as FT-7800R)	
FT-8000R	146 MHz Receiver:	<1.0x10–5.
	146 MHz Transmitter:	7.2x10–5.
	440 MHz Receiver:	<1.0x10–5.
	440 MHz Transmitter:	4.7x10–4.
FT-8100R	(No Data - should be similar to 8000 and 8500)	
FT-8500	146 MHz Receiver:	<1.0x10–5.
	146 MHz Transmitter:	1.2x10–4.
	440 MHz Receiver:	<1.0x10–5.
	440 MHz Transmitter:	6.6x10–4.
FT-8800R	146 MHz Receiver:	<1.0×10–5;
	146 MHz Transmitter:	2.1×10–4.
	440 MHz Receiver:	<1.0×10–5;
	440 MHz Transmitter:	<1.0×10–5.

FT-8900 (quad band 10m 6m 2m 70cm - no BER data published in QST review)

FTM-350R (TNC is internal APRS only)

Kenwood

TM-V7A	(no data published in QST review)	
TM-V71A	146 MHz Receiver	9.1×10–5;
	146 MHz Transmitter:	9.3×10–4.
	440 MHz Receiver:	1.6×10–4;
	440 MHz Transmitter:	9.6×10–4
TM-733A	146 MHz Receiver:	<1.0×10–5;
	146 MHz Transmitter:	8.0×10–5.
	440 MHz Receiver:	<1.0×10–5;
	440 MHz Transmitter:	3.3×10–4
TM-D700A	146 MHz Receiver:	<1.0×10–5;
	146 MHz Transmitter:	<1.0×10–5.
	440 MHz Receiver:	<1.0×10–5;
	440 MHz Transmitter:	<1.0×10–5.
TM-D710A	146 MHz Receiver:	<1.0×10–5.
	146 MHz Transmitter:	<1.0×10–5.
	440 MHz Receiver:	<1.0×10–5.
	440 MHz Transmitter:	<1.0×10–5.
TM-D710GA (No data)		
TS-2000	146 MHz Receiver:	<1.0x10-5
	146 MHz Transmitter:	<1.0×10–5.
	440 MHz Receiver:	<1.0×10–5.
	440 MHz Transmitter:	<1.0×10–5.
TM-G707A (No data published in QST review)		

The TNC-Pi9k

This new board is developed by John Wiseman, G8BPQ, who also a major developer of the currently used TNC-Pi2 for the Raspberry Pi and what is used in all the versions of the PiGate and PiGate RMS devices to date. John is also known for his excellent BPQ32 software suite that has been in use by hams for decades and has years of experience in digital modes.

Because of the computing speed required to effectively encode and decode 9600 baud signals, and some other technical hurdles including some enhancements John had in mind, he decided to use some new technology. The TNC-Pi9k is based upon a Teensy microcomputer board that is the size of a stick of gum (and just a little thicker). The overall TNC board size is the same as the TNC-Pi2 and fits perfectly on top of a Raspberry Pi3. Here is a picture of one of the two boards I used for testing (larger than real size):

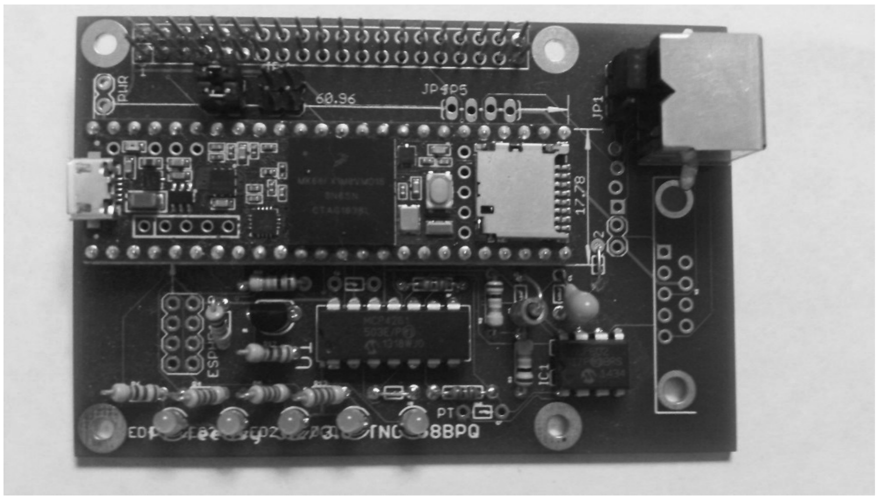

Figure 1

You’ll notice the Teensy 3.6 board with the mini-USB plug on the left end (used for programming) and the socket for a mini-SD card on the other. This super neat device boasts a 32 bit 180 MHz ARM processor with a built-in floating point unit, and 512Mb of RAM memory. The other board components are just a dual op amp, a digital potentiometer, and the circuits to integrate to the Teensy which does all the work.

Because the Teensy is a minicomputer, the actual TNC is the software that is loaded on the Teensy, which appears to the Raspberry Pi as a hardware KISS TNC. This board can replace the currently used TNC-Pi2 without any changes to software on the Raspberry Pi or other hardware. Just plug it into the Raspberry Pi GPIO pins and you’ll have 9600 baud packet (if your radio is capable).

How it works

The Teensy microcomputer software is developed using the Arduino software development tool. The software is coded in the “C” programming language and is based upon the Thomas Sailer (HB9JNX/AE4WA) Soundmodem software. The development tool includes a “C” compiler and once successfully compiled, it will upload the software to the Teensy device using the mini-USB programming port on the card. During the testing cycle, John would send me the source code and I would compile it and upload to the test Teensy boards. It was an easy process. John also fixed several issues with this rather old software and added some enhancements, such as support for the new but not yet officially released Amateur Radio Digital Open Protocol (ARDOP). More information on this new and exciting capability will be forthcoming.

The TNC connects to the radio through a single 6-pin mini-DIN connector that is standard on all radios supporting 9600 baud packet. There can be two connectors on the TNC-Pi9k, with the 6-pin mini-DIN (see Figure 1) or a DB-9 serial connector. Both will work, but the 6-pin mini-DIN is so much easier to use, you just need a straight through cable, available just about everywhere (I got mine from eBay). The DB-9 will require a special cable to be made or purchased.

Programming the Teensy on-the-fly is possible by setting what looks like programmable registers on the older TNC PIC microcontroller. The settings that are most useful are:

Register	**Use**	**Normal Setting**
1	TX Delay	100
8	Baud rate	12 or 96
9	Receive Audio	0 (sometimes not)
10	Transmit Audio	4 or 200
15	TNC reset	2

These registers can be changed using a new version of the **pitnc_setparams** utility that will be available online for download (site not yet determined). To switch from 1200 baud to 9600 baud mode would normally require changing the baud rate register, the transmit audio register, and then setting the reset register to reset the TNC. Of course, a manual change would probably be necessary on the radio to switch it to 9600 baud mode through the radio's menu system.

A change to the receive and transmit audio setting would most likely be required since most of the 9600 baud capable radios I have reviewed have different line voltage levels needed for 1200 and 9600 baud. This should be noted in the manufacturer's manuals for that radio. For example, my Yaesu FT-7900 radio shows a maximum input level of 40 millivolts (mV) peak-to-peak for 1200 baud, and 2,000mV (2 volts) peak-to-peak for 9600 baud. To be successful, the transmit audio would need to be changed. The TNC-Pi9k supports a range of audio levels using a scale from 0 to 255, with 0 being zero volts and 255 being 3 volts (or 3,000mV). Here is a quick reference table:

Setting	Line Voltage
255	3,000mV
192	2,250mV
160	2,000mV
128	1,500mV
64	750mV
32	375mV
16	188mV
8	94mV
4	47mV
2	24mV
0	0mv

Most radios use a level of about 300-500mV for receive audio. Setting the receive audio register to match would be necessary, except John has programmed into the software the ability for the receive audio to be automatically adjusted depending upon the level seen on the RX input. This level is constantly being adjusted. However, in my initial testing, I found that at 1200 baud, a receive audio level of 50 (~500mV) decoded more packets than with the level set to zero. This may not be necessary as the production units arrive and John has the chance to tweak the software. A level of zero was necessary for 9600 baud to decode correctly.

I also found that a transmit level of 200 for 9600 baud operation yielded the best results. Setting the level to less than that started to cause packets to be rejected on the receiving end, while a setting of 4 for 1200 baud was the best.

Radio Transfer Tests

Once the ideal, or at least usable, signal levels were determined, and I was consistently able to send and receive data, it was time to do some data transfer speed tests. In my tests of transfer rates at 1200 and 9600 baud, I used a simple setup to try and eliminate all variables except the data speed. I used two identical Yaesu FT-7900 radios, selected because they have some decent BER values as shown in the table above (and they were not too expensive). Both were set to 5 watts output power, with the antennas only about 25 feet apart. I attenuated the signals somewhat by one antenna being on the roof and the other being in the basement, with a roof and two floors between them. I found that in my testing environment, 145 MHz just did not work very well, even at 1200 baud. This was most likely due to interference from the many electronic devices in my home on the antenna and radio in the basement. Ar antenna clear of this electronic noise would work better. Switching to 433 MHz for all tests fixed that issue. I was not able to do 9600 baud testing between two different locations. Those tests will come later.

I used several different file sizes to make the tests, but all were text files, no binary data.

Tests results used this formula: file size (bits) / time (seconds) = transfer speed (bps)

The results of the tests were about what I expected. Even though mathematically, 9600 baud is eight times faster than 1200 baud, the actual data transfer rate will not be eight times faster. This is due to the ARQ method of packet radio, where each packet transmitted needs to be acknowledged as received correctly. The details of this protocol are beyond the scope of this paper, so if you are interested, please research this on the Internet.

The back-and-forth nature of the ARQ packet protocol has a greater effect on 9600 baud transmissions than it does at 1200 baud because the wait time between sending or receiving a set of packets and getting the acknowledgment from the other end is a higher percentage of the overall data transfer time. For example, if there is a pause of 1 second each time the transmitter has to wait to send the next packets, and there are 20 such pauses to transfer a 25kb file, with a total transmit time of 10 minutes at 1200 baud, that wait time is 3.3 percent of the overall data transfer time. For a total transfer time of one minute (60 seconds) at 9600 baud that same wait time is 33 percent of the total transmission time! These figures are a little exaggerated, but you see the point.

Add to this the time it takes to resend a packet that was received in error, and the same notion applies….any pauses in the data transfer can severely slow down a 9600 baud transmission, much more so than at 1200 baud.

So to achieve high transfer rates at 9600 baud, you must have two things:

1. Quick turnaround time for both the transmitter and receiver
2. Very good signal strength to offset bad data and avoid resending rejected packets

Even with these two parameters at optimal, you still won't get 9600 bps transfer rates because of the nature of the packet protocol.

For my initial tests, I eliminated as much radio interference and noise as I could by using the setup I described. Initially, I set the transmit delay time to 1,000 milliseconds (1 second). In addition, to prevent any delays caused by the radio needing to un-squelch first when it receives a transmission, both radios had their squelch turned off.

These are the test results for 9600 baud. These are the most interesting, considering we already know what the data transfer rate would be for a 1200 baud connection. I made the tests several times with different file sizes, and the actual transfer rates varied slightly. These are the typical results seen.

Baud Rate	File Size	Transmit Time	BPS
9600	25kb	40 sec	5,000bps
9600	17kb	32 sec	4,250bps
9600	6.6kb	10 sec	5,280bps

You would expect the transfer rate to be higher (or faster) with the larger files since the ARQ exchange might be compensated by the increased number of large data packets being sent. This was not the case as transfer rates seemed to jump all over the place and no consistency could be measured.

Changing some settings on the Teensy TNC had an effect on the data transfer rate. The TX Delay register indicates the amount of time the TNC will wait from the moment the last transmission was received until it will start to transmit. For example, the setting for the TX Delay is defined as the **TX Delay = (value * 10ms)**, so a value of 100 is equal to a 1 second TX delay time. Changing that to ½ second delay time (value of 50) increased the data rate about 13% as measured by several transfer tests of the same file within minutes of each test, the only variable was the TX Delay setting. Keep in mind that the radios themselves must be able to handle quick turnaround times, and turning off the radio squelch is necessary so incoming packets are not cut off by slow un-squelching.

Baud Rate	File Size	Transmit Time	TX Delay	BPS
9600	20kb	47 sec	1sec	3,404bps
9600	20kb	41 sec	½ sec	3,909bps

This is not too much of a change, just 6 seconds faster. Changing the TX Delay time has more of an effect on the ARQ exchange packets than the transfer of the actual data packets. This is most likely due to software induced delays in building the packets to transmit, or decoding those just received and calculating a CRC to determine if all the packets were received successfully. It was easy to hear during a file transfer the very quick exchanges of control packets and the much longer delays between sending and receiving data packets. Tweaking of the software will likely result in improvements.

Note also that the time to transmit the 20kb file above is slightly longer than what would be expected and the data rate slower than the previously listed tests. This was confusing to me but several tests using the same file confirmed this. Keep in mind that a decrease of only 10 seconds in the time taken to send this 20kb file would have resulted in a transfer rate of near the maximum for 9600 baud (~5,400bps). At these speeds, it is harder to consistently achieve a steady transfer rate given all the variables in play. More testing will likely determine the reason.

DEBUG Port

John added a handy debug port as part of the Teensy software. On the Teensy board, there is a mini-USB connector that is normally used to upload programs to the board, but also acts as a debug port. To be able to read this port, you use a standard USB cable with a mini-USB connector at one end, plug that into the Teensy and the other end into one of the Raspberry Pi's USB ports. When you do this, a new tty device will show up and this can be accessed to read the debug data. On my testing devices, this port showed up as /dev/ttyACM0. By using the “cat” utility to read the port after the device is powered on, you can see the initial output, including all the registers and their values:

```
pi@pigate9600:~ $ cat /dev/ttyACM0
Monitor Buffer Space 64
Host Buffer Space 511
Adjusting RX Level 1500 mV Pot 128
Adjusting TX Level 4 mV Pot 0
Hardware Serial No 00:04:11:66
Teensy Packet TNC by G8BPQ Version 0.2 May 2017
based on Soundmodem by Thomas Sailer
CPU 180000000 Bus 60000000 FreeRAM 225223
0    2
1    100
2    64
3    10
4    0
5    0
6    255
7    255
8    12
9    0
10    4
11    255
12    255
13    255
14    255
15    255
AFSK Mode 1200 Baud
Baud 1200 AFSK 1 FSK 0 PSK 0 Samplerate 12000
```

To continue monitoring the debug port while the TNC is in operation will require a serial program to be attached to that device. **Minicom** is a terminal program freely available for the Raspbian operating system and is installed using the **apt-get** utility.

Most of the data output on this debug port is of little use except to the developer, but it’s interesting to watch as a file transfer is going on.

Conclusion

The new 9600 baud capabilities of the TNC-Pi9k board along with John Wiseman's software gives the amateur radio community the ability to achieve relatively high speed data transfer at a fraction of the cost of existing 9600 baud capable TNC hardware and with an easy to use form factor that connects directly to the popular Raspberry Pi minicomputer. However, data transfer rates are not going to be as high as what might be expected, but still a higher rate than many other transfer protocols on the market, including PACTOR or the Windows WINMOR protocols. 9600 baud packet transfer rates normally hover around 3,400 to 5,400 bits per second. Not the 8 times increase in speed one would hope for, but still, up to 5 times faster, fast enough to be useful in transferring fairly large files. More tweaking of the software will likely improve these figures.

However, the need for low noise with 9600 baud FSK communications means only the VHF and UHF frequencies can be used. This makes the new board effective for only relatively short communications links. Previous testing of 1200 baud links at 145 MHz indicates that 12 to 15 miles in a hilly area with simple antennas is a realistic range expectation. While testing long distance links at 9600 baud was not conducted due to time constraints, we can infer that the useful distances will likely be less. We should also remember that QRM can be a major factor in establishing a good data link.

As with all amateur radio stations, antennas are the key to better performance. Using high gain Yagi style or even parabolic dishes at each end of the transfer link would greatly enhance the signal strength and therefore the distances that can be realized. My previous tests with 1200 baud packet on 145 MHz showed that a base station with a good omni-directional antenna and a remote station on a hilltop with a good 13db gain Yagi antenna running 20 watts output power, data links were able to be established over 25 miles. UHF links will likely be just a little less. This is very acceptable for the PiGate project, to be able to transmit e-mail from within a disaster area. A useful factor of the PiGate and PiGate RMS devices is the radio link is under the complete control of the operator. By choosing the frequency and both antennas, clear data links can be made that would otherwise not be possible.

In the future, it is possible that data compression techniques would be one method of increasing the data transfer rate, as has been done in the past with dial-up modem technology. Also, since this TNC is software based, it might be easy to develop new methods, perhaps PSK, QPSK or MT63, or other modes to further increase the data rates. ARDOP is a sound card mode that will soon be available with this same device, so who knows what the future may hold for amateur radio digital communications.

References

1. Shannon-Hartley theorem. https://en.wikipedia.org/wiki/Shannon%E2%80%93Hartley_theorem
2. Teensy minicomputer. https://www.pjrc.com/store/teensy36.html
3. G3RUH 9600 baud modem design. http://www.amsat.org/amsat/articles/g3ruh/109.html
4. KD2BD 9600 baud modem. http://www.amsat.org/amsat/articles/kd2bd/9k6modem/
5. ARDOP. https://www.tapr.org/pdf/DCC2015-ARDOP-KN6KB-N8OHU-GM8BPQ.pdf
6. Learn more about the PiGate. http://pigate.net