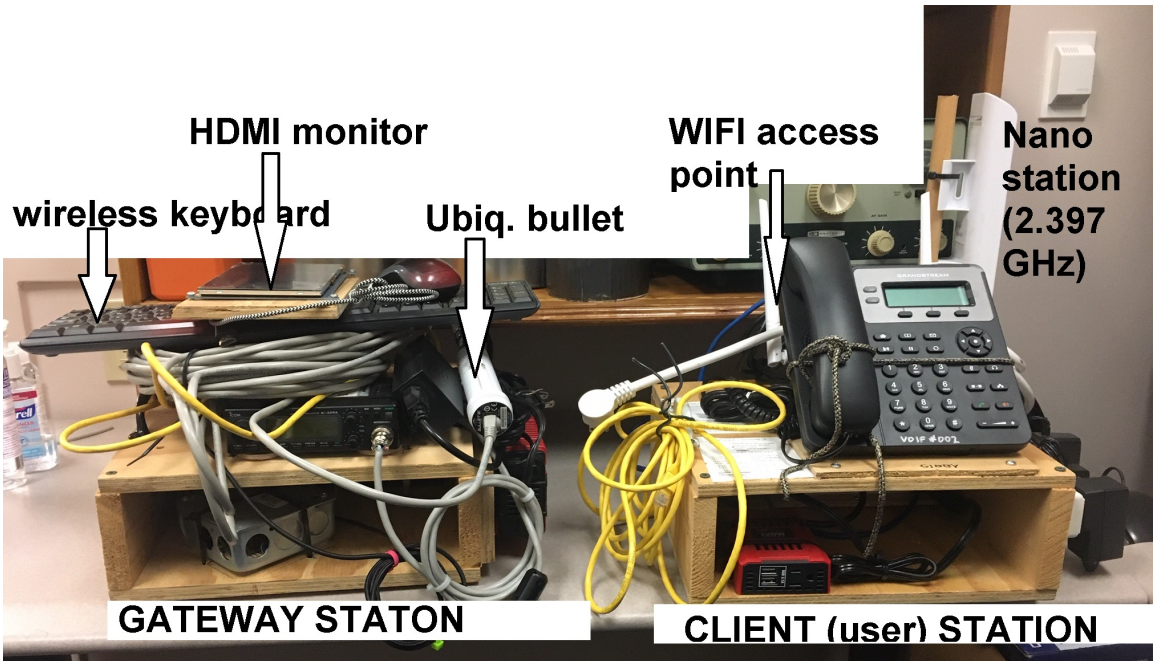
Making KISS Connections from Winlink Express to linbpq nodes
VHF vs MESH: WINLINK either way….

Gordon Gibby KX4Z
Version 1.2 June 24, 2020



Introduction: This is All Very Old-- Here's where the Jargon came from…

This is a story that is 45 years old. The things we need to learn are literally THAT OLD and have been used for DECADES by amateur radio operators…..and now is our chance to take advantage of all the pioneering work done by so many!!

1976: X.25

About 45 years ago, in 1976, the International Telegraph and Telephone Consultative Committee (CCITT, which is now part of the International Telecommunications Union) published what was the X.25 standard for moving organized "packets" of 1's and 0's across a network to accomplish digital communications, particularly for the purpose of banking and automatic teller machines. [1]

1's and 0's, even on a wire connection, in packets, from multiple connected terminals, can bump into each other, so the system dealt with "collisions."

1 https://en.wikipedia.org/wiki/X.25

1978: AX.25 / TNC

Soon, amateur radio operators (and others) replaced 1's and 0's (+12V, - 12V) with high and low pitched audio tones and fed them into their radio microphones and created the amateur radio standard AX.25 ("amateur X.25") shortly after X.25 was born. By 1978, the Montreal Amateur Radio Club (of Canada) and the Vancouver Area Digital Communications group had created an amateur radio "terminal node controller" to pass the AX.25 pitch tones across radios.[2] Only available as a bare circuit board, this didn't become very popular – but the Tucscon Amateur Packet Radio (TAPR) association had KITS for a complete TNC-1 design by 1983. This and its successors became wildly successful.

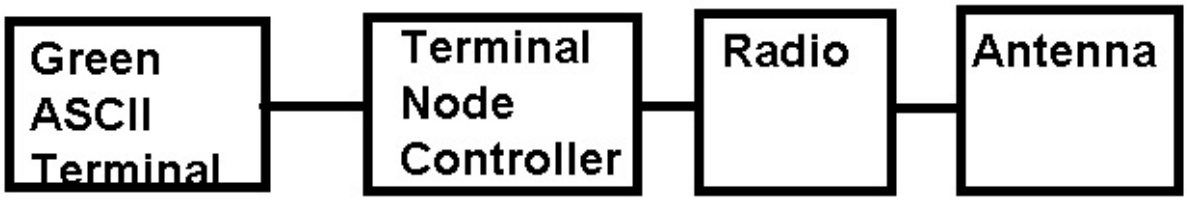


Figure 1: *How amateur radio operators send digital characters back 45 years ago.*

1984: AX.25 2.0

Version TWO of AX.25 was published in 1984 by Terry Fox WB4JFI. [3]

But soon the personal computer was born, and began to double its processing speed every 18 months, making green screen ASCII terminals eventually become obsolete. Why not put more of the processing of the AX.25 protocol (handling the collisions, etc) in the COMPUTER, and less in the external hardware device?

1987: Keep It Simple, Stupid (KISS Protocol)

In 1987, Mike Chepponis, and Phil Karn (KA9Q) presented a new protocol, Keep it Simple, Stupid (KISS) to do just that.[4] The modem would now do only the most essential of radio tasks – create the tones to go to the microphone, know when to hit the push-to-talk line, etc. The COMPUTER on the other end of the KISS connection would be sending out exactly what needed to be transmitted, and receiving and knowing what to do with, what was received. This allowed multiple kinds of communications—including tcp/ip, bulletin boards, whatever --- to be handled by the computer end, and the modem just handled the lowest lowest layers of the OSI model of layered network communications.

2 https://en.wikipedia.org/wiki/Terminal_node_controller
3 See the Foreward: http://www.tapr.org/pdf/AX25.2.2.pdf
4 http://www.ka9q.net/papers/kiss.html

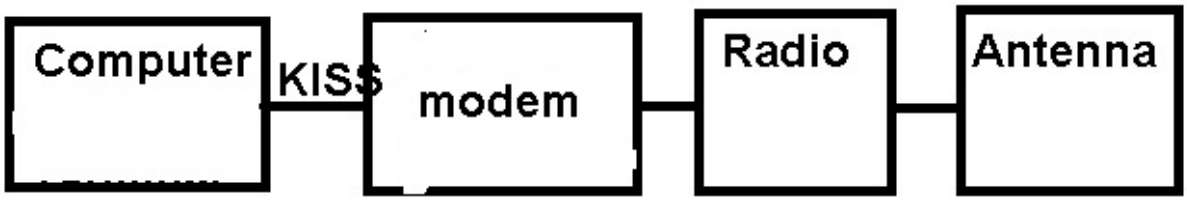


Figure Two: *More of the smarts moves into the computer, communications with the hardware modem are by KISS*

Computers of that era still weren't fast enough to handle the sine waves of TONES and do all of the frequency detection, etc….so the modem was still a very expensive and slow thing. The HAYES company made a lot of money with their "smart modems" that could send such tones over telephones at speeds of up to 1200 bits per second.

Then the SOUND CARD got developed with digital signal processing special purpose expensive chips, and everyone was doing music on their computer instead of their phonograph.

So now much of the smarts of the modem got moved into the Sound Card. The only problem was that your sound card got used to make all kinds of WARNING SOUNDS also….so sharing your sound card with WINDOWS and with the radio made for embarrassing bells and whistles transmitted on top of your BBS or email…..so people replaced their modem with an EXTERNAL SOUNDCARD, which was then driven by software that did most of the smarts of the modem, still inside the computer.

And the communications program now talked to the remaining modem software INSIDE the computer, by way of KISS protocol….just going from one "port" out of the 64,000 of them on a given computer's tcp/ip implementation, to another. The special Internet Protocol ("ip") number 127.0.0.1 means "myself" – so application program and modem software both running on a multi-threaded operating system like WINDOWS or LINUX might "meet up" at IP# 127.0.0.1 and port 8100.

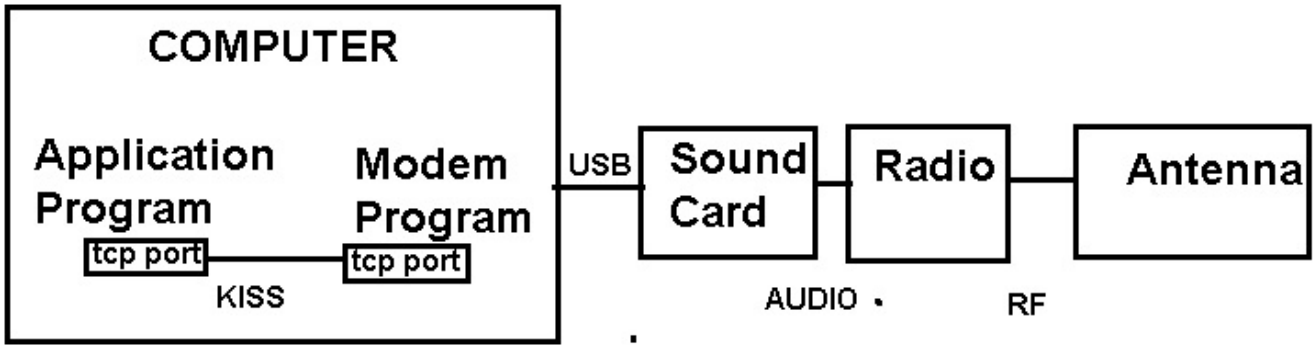


Figure Three: *With the advent of cheap soundcards, even more of the smarts got cheaper and became free modem program software like "soundmodem.exe"*

Soundcard Techniques

Now a *ton* of different ways of making the bleeps and boops came out as innovative people thought of different ways to use AUDIO to transmit 1's and 0's, into microphones of radios. Some of these were designed for 1-to-many "broadcast" type communication (as CW) – such as PSK31, or MFSK16, or RTTY. Others were created specifically for 1-to-1 "connected" communications, such as PACTOR, WINMOR, ARDOP, GTOR, VARA.

Gradually the RADIOs got smarter and smarter and manufacturers eager to distinguish THEIR product from the competition put the SOUND CARD inside the radio itself – like the ICOM 7300.

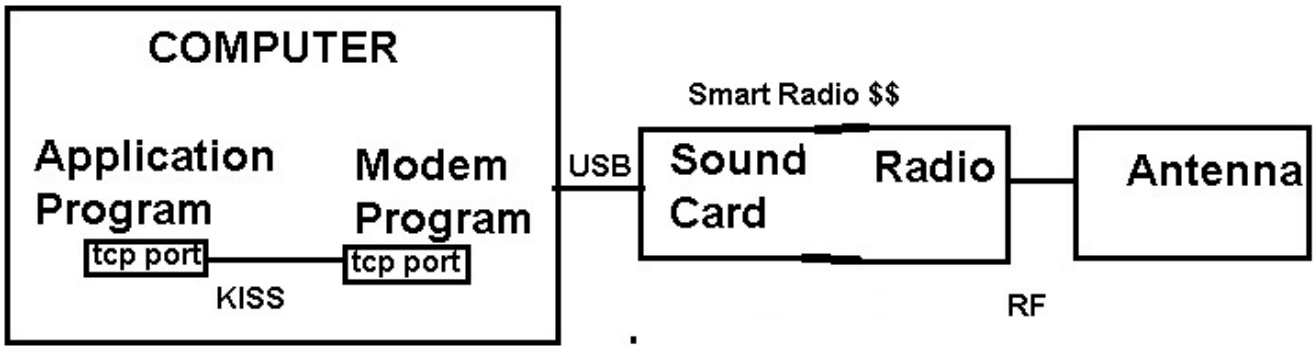


Figure 4: *Moving the sound card inside the radio made a new product, the smart radio.*

Now the funny thing is that *the KISS communication was now going between two tcp/ip sockets --- and those don't HAVE to be on the same computer*! They can indeed be on quite separate computers.

This made it easy for people to operate REMOTE RADIOS where they had one computer at which they sat in California, and another computer connected in Florida to the radio! The connection between was the Internet. Since VOICE can be sent just as easily as "Voice Over IP" this wasn't restricted to digital – it works for voice, or anything!

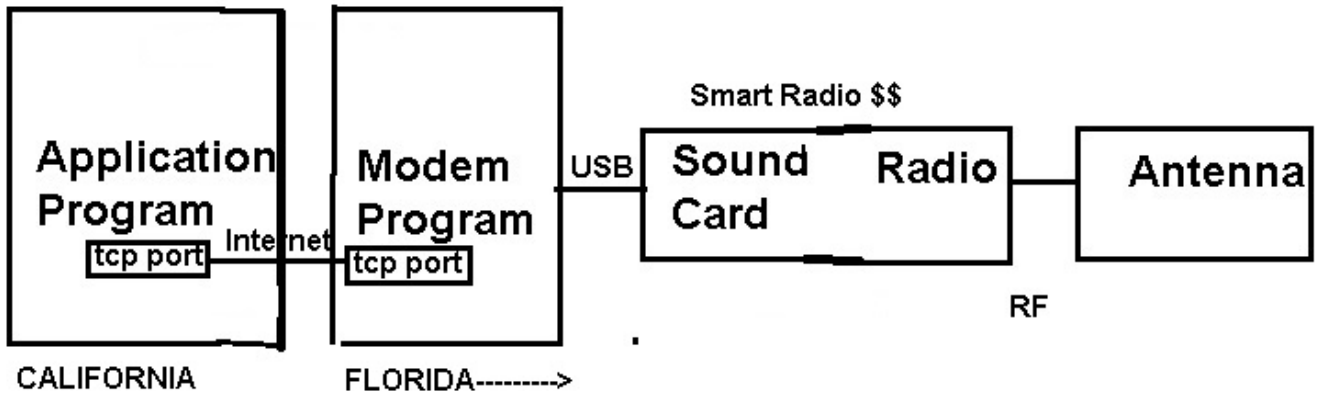


Figure 5. *Now we can split the computer in half and operate from thousands of miles away on a business trip.*

Now so far in this discussion we've left out the OTHER STATION that is being communicated with, but indeed, that station could be physically located in Tennessee..but the operator might be located in Virginia! So we have a ham in California using the internet to control a radio in Florida, then by amateur radio to Tennessee, then over the internet to another ham in Virginia!

How Winlink Connections Work

And the same kinds of connections are generally used in WINLINK:

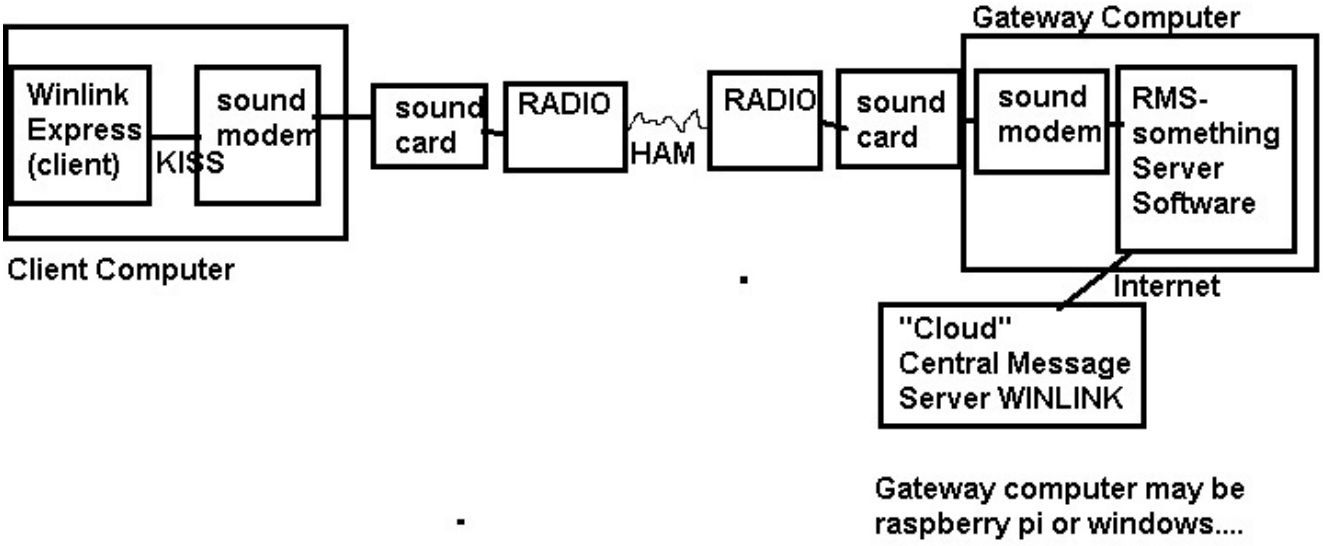


Figure.6 W*inlink client typically sends KISS-framed messages to port 8100 of the same computer where soundmodem picks them up and gets them going out to the radio, over the ether, and back again to the RMS server software which receives them as KISS framed messages from its version of soundmodem, whatever that is for whatever operating system the server is operating.*

THERE IS MORE TO HAM RADIO!

But ham radio is NOT restricted to 80/40/20/15 / 2meters / uhf --- ham radio even includes MICROWAVE…..so there is yet another way you can do this same winlink connection!

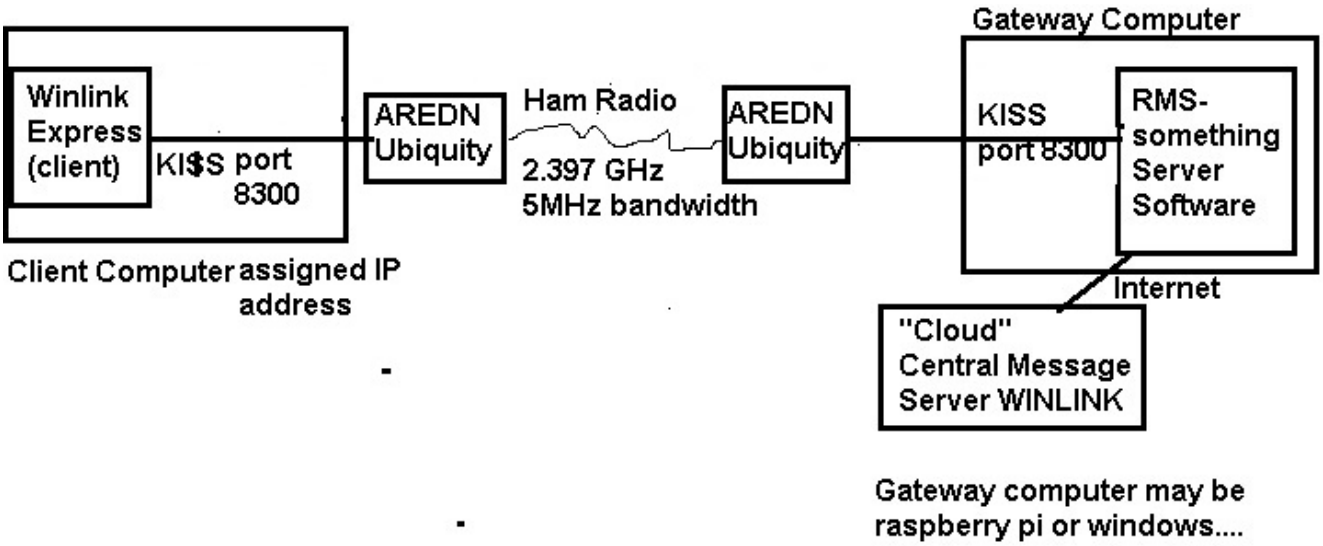


Figure 7. *Replacing the winlink vhf connection with a microwave ham radio connection*

DUAL SYSTEMS

And the really funny thing is….. BOTH computers at BOTH ends of this link….can run BOTH systems! You can have a client station that can just as easily make a winlink connection over 145.070….and both it and the server can just as easily make the KISS connection over 2.397 GHz if you just add the additional radio circuit to the system --- TWO ways of moving traffic…. Both using ham radio signals….

How-To On Making "Packet" Microwave Client Connections

Many of us are already expert at using WINLINK EXPRESS (client software) to make an AX.25 (Packet) connection to a VHF gateway. Some folks do this by having connected a hardware TNC that can do packet to their computer by a RS-232 serial port. In recent years, more of us have moved to using external soundcards (Signalink, or even homebrew devices based on cheap USB sound dongles). In the latter case, we run a modem program "soundmodem.exe" and set our Winlink Express to connect to it (on our same computer), typically on port 8100:

This isn't a talk on VHF Winlink Packet, but typically you will have your Soundmodem Settings | Devices set to communicate, on the same computer, over port 8100 as shown here:

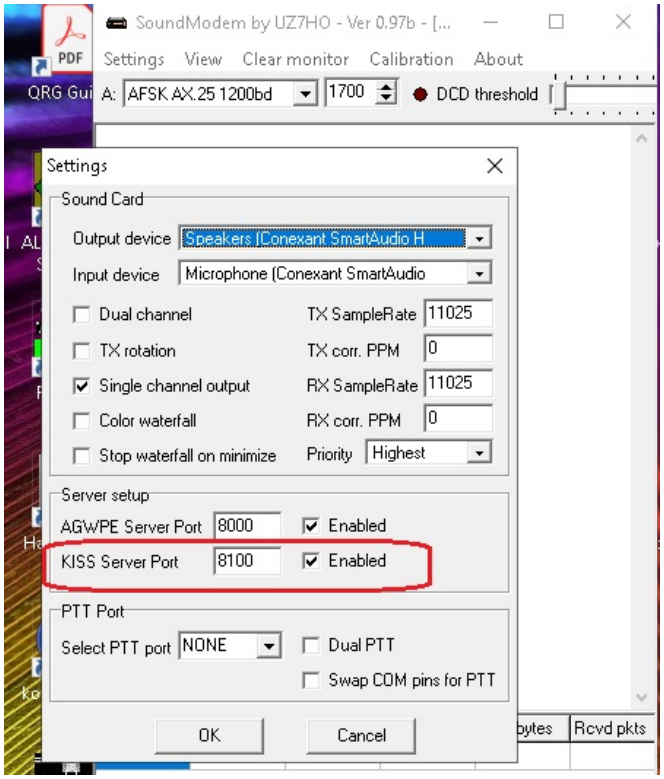


Figure 8: Soundmodem setup.

And the corresponding settings inside WINLINK EXPRESS Packet Setup are:

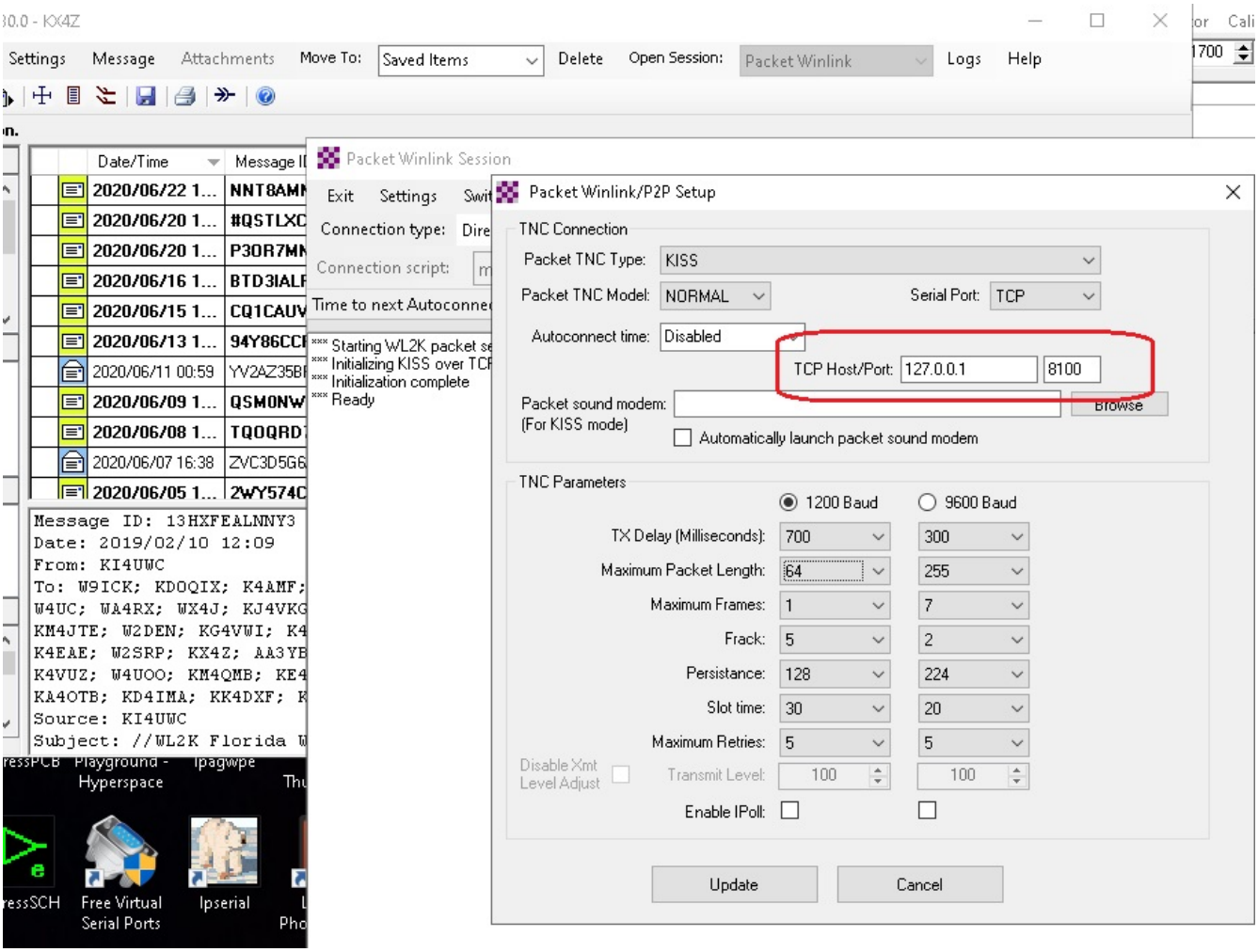


Figure9 . *Note the TCP Host is the LOCAL computer (special ip number 127.0.0.1) and port 8100 so that it links up to soundmodem. You can use any of a number of ports (as long as they aren't already in use!) if both programs are expecting the same number. The other settings for TNC Parameters are for a very modest performance radio….*

You will then initiate a call to the VHF WINLINK gateway node with something that looks like this on your WINLINK Express Screen:

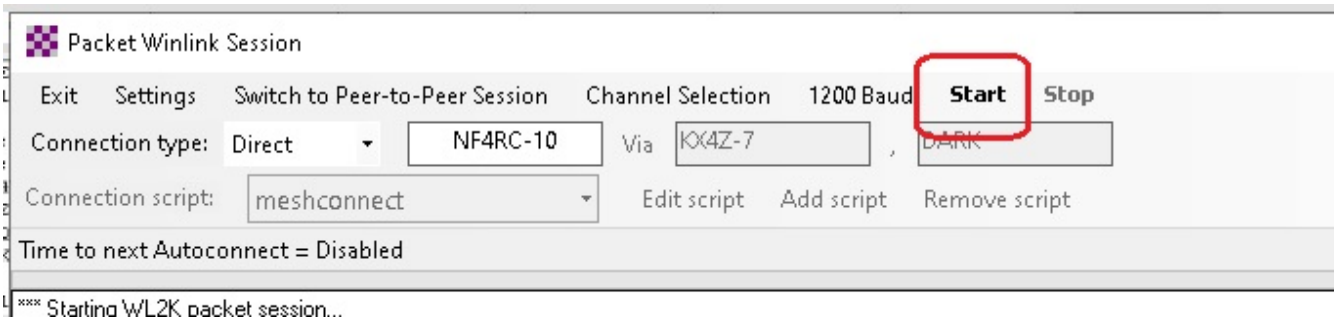


Figure 10. *Making a packet connection….on any frequency*

The START button tells the software to begin the calling sequence; it then generates KISS frames that go to soundmodem, get sent out over the radio, reach the other end and cause a response.

SETTING THIS UP FOR 2.397 GHz

Just as we need to know the FREQUENCY on which to call the gateway station (and we enter that into our RADIO, not into winlink express) for a 2.397GHz connection, the frequency is already known (both Ubiquity transceivers are on 2.397 GHz) but you must know the IP NUMBER of the raspberry pi that is the gateway –

For our field day set up that will be:

`10.247.240.27`

and the PORT on which the KISS frames are going to communicate is port **`8300`**, not 8100. If you wish, you can set things up for longer frames – such as 128. That probably isn't of crucial importance however. We aren't really OPTIMIZING this connection – we're too green for that --- just getting it going is plenty for now.

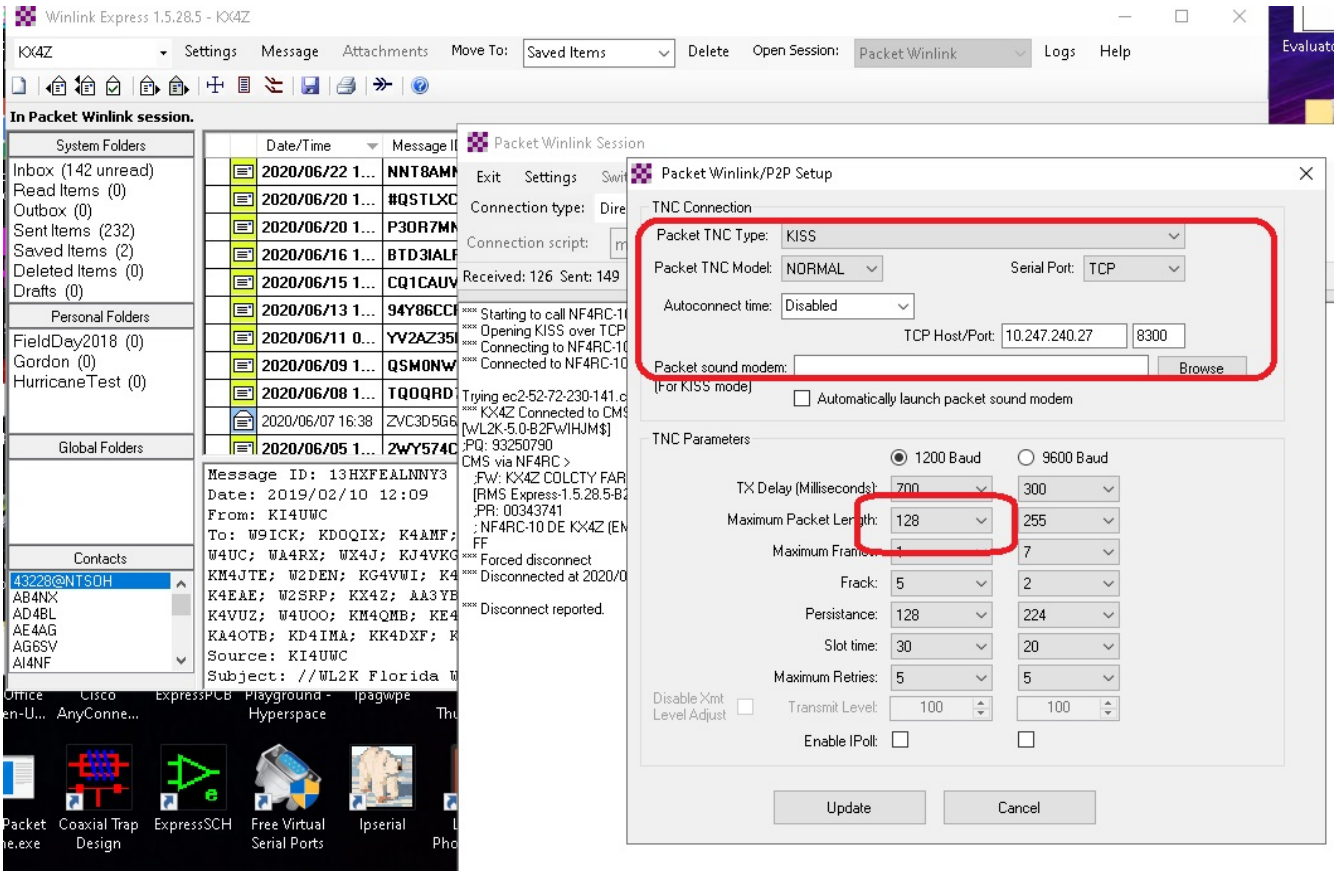


Figure 11: *The WINLINK end of a KISS connection.*

Once that setup is done….you just go back to the Packet Session, and exactly as before, click START and hopefully all should work!

KISS frames come out of your WINLINK EXPRESS, go out the Ethernet connection of your computer, head to the AREDN Ubiquity 2.397 GHz transceiver that is preconfigured with a ham radio callsign and on ham radio frequencies….and over 2.397 GHz speed their way to the far AREDN Ubiquity transceiver where they go to the raspberry pi, on port 8300 and reach the linbpq gateway software and are properly handled and your messages the n moves over normal internet (likely a cell phone hotspot) to the internet-based Central Message Server.

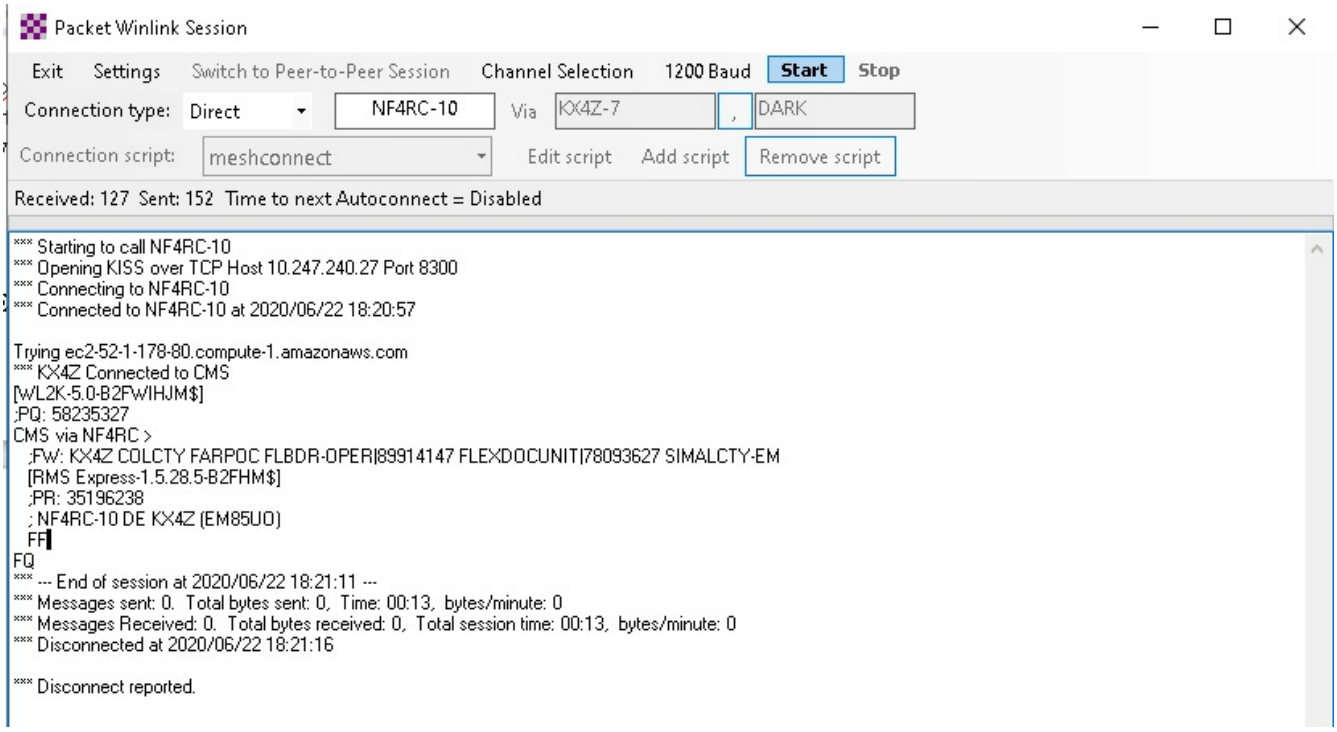


Figure 12 . Actual Winlink Connection over 2.397 GHz connections Look exactly like vhf.

It really is that simple….when it works. That depends on a LOT of settings, just like all amateur radio normally does. Gain settings, voltage settings, tone settings, frequency settings --- we are all used to that whether we work on HF, CW, FM, or whatever. The remainder of this paper is to get you read for this question:

BUT WHAT IF THINGS DON'T WORK?

How A Raspberry Pi LINBPQ Gateway Works

In order to deal with possible errors, we have to understand the entire system – including the far end, which for our not-wealthy group, is a simple raspberry pi running Linux and John Wiseman (G8BPQ)'s wonderful linbpq software.

For years, we've been using his software, which requires carefully chosen configurations in a file named bpq32.cfg, setting up "ports" [not the same as tcp/ip ports, but analogous] for both radio and

telnet. In those days, soundmodem didn't exist on the linux platform, so we used DIREWOLF, which did, and does the same function. The specification for a radio port looks like:

```
; ----------------PORT 4  THE 2 METER RADIO ----------------------
;
PORT
ID=Direwolf Soundcard-based
 TYPE=ASYNC
 PROTOCOL=KISS
IPADDR=127.0.0.1         ; DIREWOLF
TCPPORT=8001             ; DIREWOLF
 SPEED=9600
 INTLEVEL=4
 CHANNEL A ;   worked with TNC-X  & also for Direwolf 1 chan
 QUALITY=0
 MAXFRAME=1            ; training wheels setting
 FULLDUP=0
 FRACK=10000
 RESPTIME=3000
 RETRIES=10
 PACLEN=64             ; also training wheels
 TXDELAY=500           ; intentionally long....training wheels
 TXTAIL=50    ;  used only by KISS devices...long so you can hear packet
 SLOTTIME=100
 PERSIST=64
 DIGIFLAG=1
 DIGIPORT=0
 MHEARD
 BROADCAST NODES     ; so routes will appear in other person's view
ENDPORT
```

And the specification for the TELNET port (that is used to connect via the normal internet to the Winlink CMS system looks like this:

```
; ********************PORT 3  TELNET ************************************
PORT
 ID=Telnet Server
 DRIVER=Telnet
 CONFIG
    LOGGING=1
    DisconnectOnClose=0     ;  1 = closes window when you bye
    TCPPORT=8010
    LOGINPROMPT=user:
    PASSWORDPROMPT=password:
    MAXSESSIONS=10

    CMS=1
    CMSCALL=K4AAA-10       ; CMS Access Callsign (with SSID if used)
    CMSPASS= *******               ; WL2K sysop password
    FALLBACKTORELAY=1    ; will try to get to RMS RELAY if CMS unavailable
    RELAYHOST=192.168.1.21  ; put the name or ip number of your RMS_RELAY here
```

```
    CTEXT=Welcome to K4AAA Telnet Server\n Enter ? for list of commands\n\n
    USER=username1,password1,callsign1,,SYSOP
    USER=username2,password2,callsign2,,
; add as many as you like
ENDPORT
```

And it turns out that we can easily add a “port” that knows how to accept KISS frames over a wired Ethernet direct connection from the Raspberry Pi to a handy Ubiquity transceiver and do the right things with them:

```
PORT
 PORTNUM=10
 ID=KISS over TCP Slave
 TYPE=ASYNC
 IPADDR=0.0.0.0    ;  yes, that is the correct number!
 TCPPORT=8300
    CMS=1
    CMSCALL=NF4RC-10       ;
                   ;CMS Access Callsign (with SSID if used)
    CMSPASS= ********               ; WL2K sysop password
    FALLBACKTORELAY=0    ;
                   ;We aren't going to have RMS_RELAY available
    RELAYHOST=192.168.1.50  ;  futile
ENDPORT
```

STEP BY STEP:

TURNING ON AND DEPLOYING THE RASPBERRY PI WINLINK GATEWAY

PREPARATION – BEST STARTED IN AIR CONDITIONING

1. Plug an HDMI monitor into the raspberry HDMI output. (this may require a USB connection to get power.
2. Plug the USB dongle from a wireless mouse/keyboard into one of the USB ports.
3. Verify that the soundcard USB is plugged into one of the USB ports.
4. Verify that the LAN side Ethernet cable from the POE goes to the Ethernet jack of the Raspberry pi – this is how the Ubiquity communicates to the raspberry pi.
5. Verify that the POE side of the POE injector goes to the Ubiquity – this is how it gets power and data.
6. Plug the positive wire back into the 7Ahr sealed lead acid battery and verify that the raspberry pi (fed from a cigarette lighter USB down-voltage-converter) turns on Lights will flash. The linbpq program is automatically started (or restarted) if not already running by a cron task that operates every 2 minutes. Likewise the direwolf process that handles the soundcard.
7. Plug in the Walmart battery maintainer to keep the battery charged.
8. When the desktop screen comes up, find the up/down arrows or other WIFI indication in the bottom tray to the right, click on them, choose your cell phone hotspot, provide the passphrase in order to get your internet connection working.

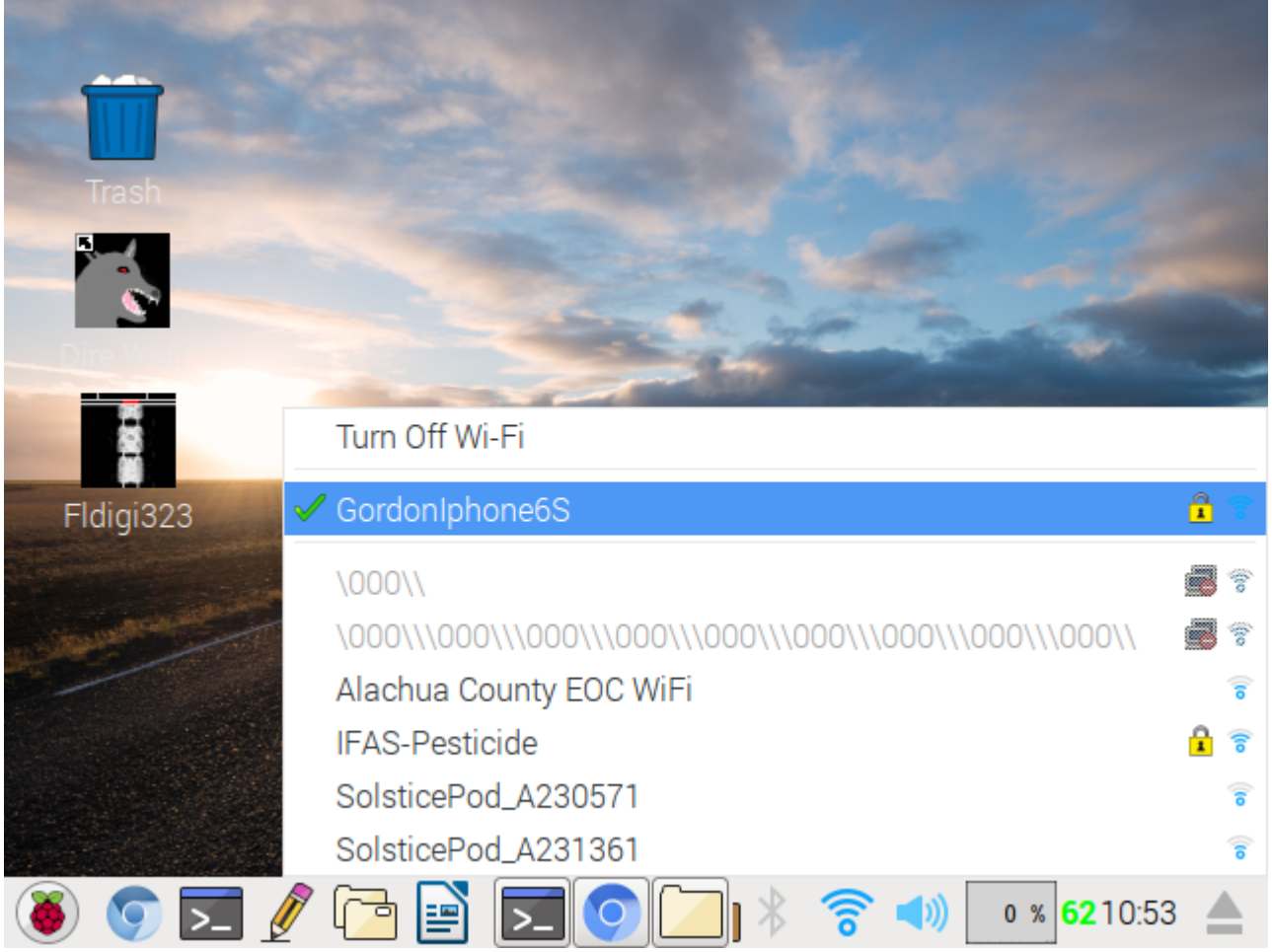


Fig 12: *What the LINUX Raspberry Pi screen looks like and how you select a WIFI source. When you aren't net connected, that WIFI symbol visible in the bottom tray will looks like up/and/down arrows instead. Stock Linux raspberries have their tray at the TOP --- but I move mine down to the bottom to make it more llike Windows. The green number is the temperature in celsius of the processor. The RASPBERRY is like a "start" button. The PENCIL is a text editor; the tri-colored round circle is a CHROME BROWSER.*

9. Do something to verify that your internet connection works – turn on the ancient Chrome browser and go to your favorite news source, for example --- when confirmed, turn OFF the browser so as not to slow down the raspberry pi.

NOW YOU CAN DEPLOY

10. Now deploy (physically move) to the desired location, possibly outside the 1000 foot dia. Circle of your field day effort, keeping the raspberry pi going on the battery and the cell phone hotspot nearby.

11. When deployed, find a source of modest 110V AC, and plug the battery maintainer AND the ubiquity power-over-Ethernet (POE) supply into it – this can even be a small inverter from your vehicle's cigarette lighter, or a small generator – there are only a few watts needed.

12. When the POE is powered, the Ubiquity will begin a 60-second turn-on routine, after which the RED light should illuminate if it sees any other stations on its frequency...and its DHCP server will provide an ip number properly to the hardwired Ethernet port of the raspberry pi.

13. Turn on the VHF radio, provide it with a suitable antenna, set for LOW power and the desired frequency (e.g., 145.070 MHz)

14. Position the 2.397 GHz Ubiquiti so that its antenna has an unobstructed line of sight to your intended client location.

15. Your Server should now be fully operational.

TROUBLESHOOTING

First: Getting the 2.397 Radio link working

1. If the two ubiquity transceivers reach a RED light after about 60 seconds from turn on --- they have seen each other radio-wise and are connected. If this doesn't happen, you don't have a radio connection and should improve the path.

2. You can always use your browser to look at the web-server on the Ubiquity node to see what signal strengths it is seeing, what computers it is seeing on its local area net, and so forth. However, the address of its web server is NOT the standard 80 one, so you will need to add 8080 after its IP address in your browser's URL entry box.

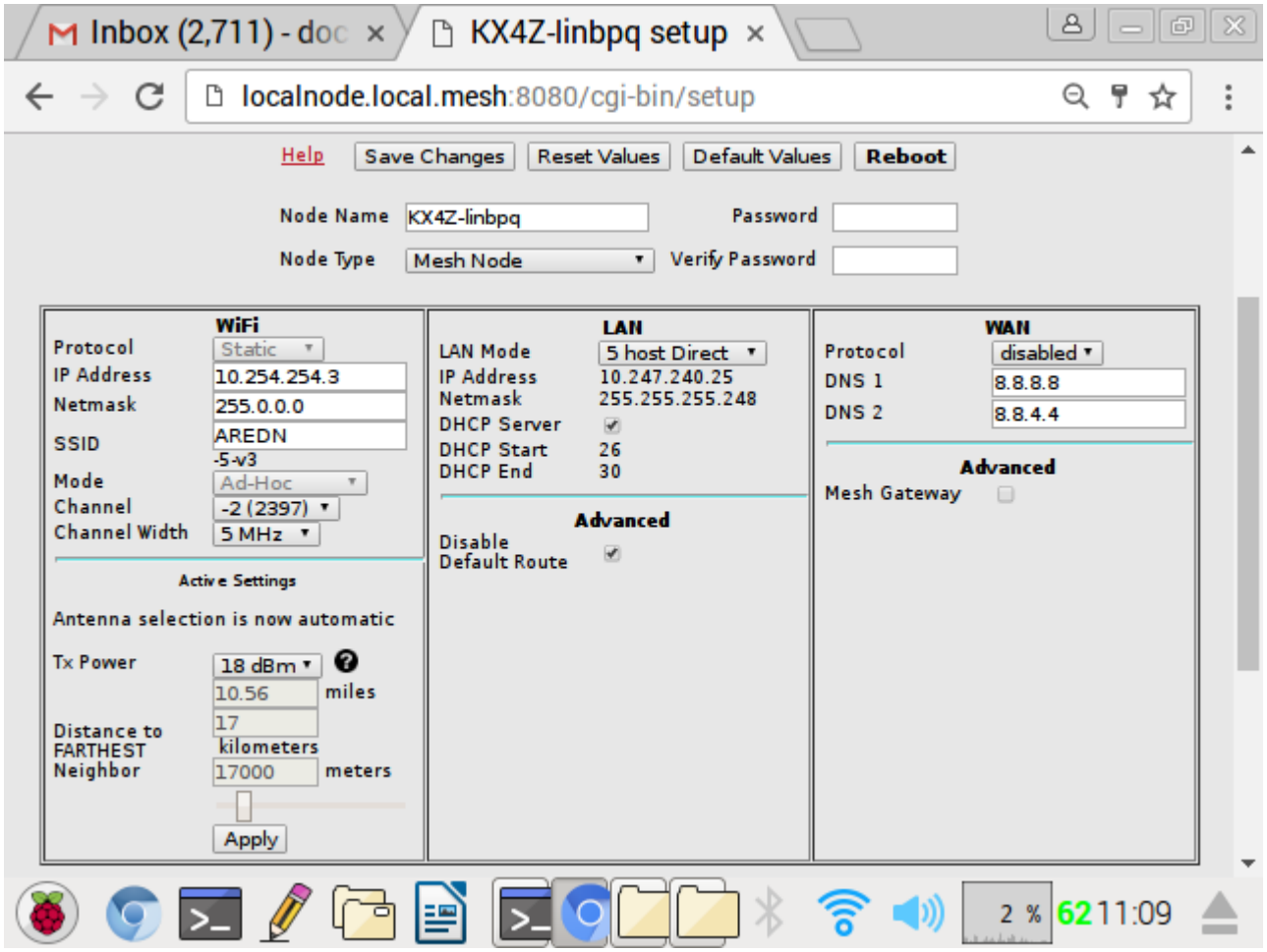


Figure 13. This is ONE of the various setup pages on a AREDN mesh node web-based controller. Note at the lower left you can set the TRANSMITTER power – I have it throttled back to 18dBm here….(63 milliwatts)

A useful URL to try, in order to connect from a computer that is directly connected to your Ubiquity is

`localnode.local.mesh:8080`

as you see in the Figure above….

SECOND: Try Standard TCP/IP Routing Tests

By understanding the "anatomy" of the tcp/ip connections from client computer, to ubiquity device, to far device, to raspberry pi, you can probe the system to detect where a breakage is in two ways:

a) "ping" successive devices

b) use tracert (WINDOWS) or traceroute (linux) to see how far the connection goes before a failure.

The idea behind traceroute is to pick the END OF THE CHAIN ip number, and start from the other end and issue a command to trace the route all the way to the far end – the system shows you each connection as it is made. However, this uses a type of packet that WINDOWS systems are often set to refuse, due to the fact that hackers use these tools nefariously. So they may have variable usefulness unless you know how to enable them deep within windows. From the raspberry pi end, however, you should be able to use them all the way to the device right before a windows device.

[These numbers are for the NFARC / ALACHUA ARES setup – yours will be different.]

REPRESENTATIVE IP NUMBERS

Entity	**IP Number**	**Comment**
Your client computer	Any DHCP-delivered IP address in the 10.239.224.32-based net, subnet mask 25.255.255.240 (16 addresses)	If running ipconfig on your windows based computer shows that you have an ip that begins with 192.something…..your computer hasn't successfully connected to your client-side NanoStation. This MUST be accomplished first.
Client-side NanoStation	10.239.224.33 (local area net side	Note that this is a router, and has ip numbers on two different nets – the local area net, and the radio net.
(2.397 GHz communications)		
Server side Ubiquiti Bullet	10.254.254.3 (radio side IP number)	Note that this is a router, and has ip numbers on two different nets-- the local area net, and the radio net. The ubiquity provides DHCP ip number assignment service to a local area net in the 10.247.240 range
Server Raspberry pi	10.247.240.27 This address is reserved, so the raspberry pi should ALWAYS be at this number.	Linbpq takes in KISS frames over TCP on tcp port 8300

Results of **`tracert 10.247.240.27`**: (tracert done on the WINDOWS computer end – the comparable linux command is **`traceroute`**

```
C:\Users\Gordon>tracert 10.247.240.27

Tracing route to 10.247.240.27 over a maximum of 30 hops

  1     1 ms     *        1 ms  localnode.local.mesh [10.239.224.33]
  2     8 ms    11 ms     9 ms  KX4Z-linbpq.local.mesh [10.254.254.3]
  3     3 ms    10 ms     4 ms  10.247.240.27

Trace complete.
```

If you can't ping from your computer all the way to the raspberry pi (10.247.240.27) something is wrong somewhere. If you don't have ping windows-feature installed...of course you won't be able to do this….

Make Sure the LINBPQ is running properly on the Raspberry

How to tell if the linbpq process is working:
Open a terminal window. (Click the icon that looks like a black terminal screen.)
Type:

`pgrep linbpq`

Figure 14 . *Running the command pgrep linbpq on a raspberry pi to find out if linbpq is currently running*.

If the system responds with a process number, then it is running! (see the 1947 in the figure above/)

If you want to stop it, you can type

`kill <that process number here>`

If you wish to RESTART it, you can type:

/home/pi/linbpq/linbpq

and that will restart the program – and also let you see any error messages. The terminal window will stay dedicated to that program. You can stop the program by typing Control-C to stop it.

If you have a valid account stored within bpq32.cfg (easy to add with any text editor) you can log into the software just as if you had contacted it using packet in this way:

telnet localhost 8010

It will then ask for your username and password, and you'll be granted identical access as if you were connecting by packet – you can even command VHF connections!

If you want to see what network connections exist on your raspberry pi, the route command can help:

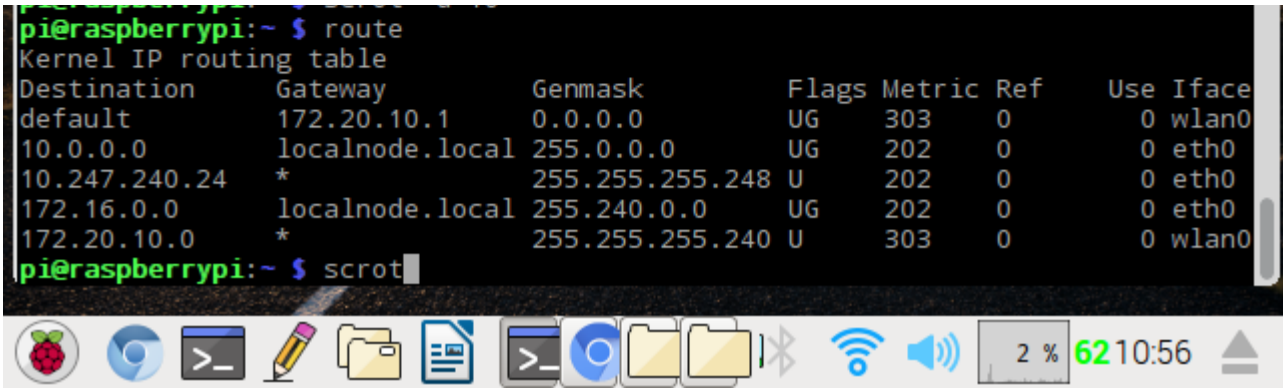


Figure 15. *This* **route** *command shows a wlano0 (wifi) connection to a smartphone at 172.20.10.1 – the default gateway for this raspberry pi, so we can tell that it likely has Internet access through that port…. And and an Ethernet (eth0) connection to a Ubiquity microwave transceiver on 10.247.24.24.*