# Portable Network Time Servers

GPS equipped Raspberry Pi Network Time Servers for on and off-net use

Draft Version of 8/25/2020, Copyright Alan Biocca

Development Begins - writing the uSD card

## Requirements

Digital modes like FT8 require the PC clock to be within a 2 second window of the correct time. The Field Day Logging (FDLog) software we use records logs to the minute, so a few seconds of clock accuracy is good enough for it as well. Our **High Sierra Field Day Group** does Field Day and some VHF contests from field locations that often do not have reliable internet service so we need something along the lines of GPS based time synchronization shown here.

The time system must be able to set the clocks of many different types of computers. The group uses multiple types of computers that run various versions of operating systems from Windows to Linux to the occasional Mac. So the time service client software must be widely available for many common systems. This drives the selection choice of the NTP protocol. Clients in common use on Windows include Meinberg NTP and btkTimeSync, (covered in the Appendices below) and a number of others. The Windows time system is not quite good enough to meet the 2 second requirement, so something else must be chosen and loaded. Mac’s and Linux generally have some type of NTP client already installed, so it is just a configuration task to include our own NTP servers.

In the field we set up a local WiFi network. This is generally not connected to the internet, or if it does have an internet tie it is not considered solid enough to rely on for important services like time. Hence we need a portable time server (or several) to provide our own reliable time service in the field.

After Field Day in June 2020 some of us decided to experiment with Raspberry Pi based NTP servers to see what could be done with simple hardware. We have a long history of trying fairly high tech solutions to problems for Ham Radio and Field Day. Brad N6BDE brought a commercial NTP server to Field Day some years ago, but it would not lock up to the satellites despite being in the clear at high altitude. Eric WD6CMU has made a couple of different NTP servers for home use, we came close to trying one of those on Field Day. Alan W6AKB looked into it when he built the FDLog GPS Clock which synchronized the FDLog applications across the site as part of its own data synchronization protocol (see Appendix), but it fell short of synchronizing the actual computer system clocks. But the arrival of new requirements with popular digital modes has changed the game.

## Basic Architecture of a Time Server

There are six essential parts in these Time Servers - Power Supply, Computer, GPS, Operating System, the GPS service daemon and the Time Server daemon. (Daemons are programs that run in the background on Unix and Linux.)

The Raspberry Pi Computer was selected due to the wide support available as well as the low cost, small size and minimal power consumption. There are a wide range of Raspberry Pi computer models to choose from, and most of them are appropriate for this project, though some are more appropriate than others. The power and size are good criteria to use, as well as the proper I/O mix. The Pi Zero doesn’t have a network interface, but the Pi Zero W has WiFi so that is a good choice. It also has very low power consumption and is low in cost and is small in size. Frank chose the Zero W as the starting point for his server which is detailed in Appendix 1. Alan had a Raspberry Pi 3B+ on hand so he chose that for the initial development testing because it has lots of easy to use interface plugs including ethernet, HDMI, four USB sockets and built in WiFi, and it is a bit faster to develop with. This unit is detailed in Appendix 2. Alan considered the Pi 4 but decided it was far more than this project needed and the power consumption was unnecessarily high. Alan also had a couple of GPS Hats that fit the 3B+ (they also fit the Zero W connector but are twice the size of the Zero W itself). Alan built a Zero W server as well, details in Appendix 3. Eric is planning to build a time server based on the Rock Pi S which is about half the size of the Pi Zero. Combined with one of the smaller GPS modules this will likely be the physically smallest NTP server. It presents some extra challenges to configure due to not having graphics hardware but there is support for USB based host interaction.

Power for the Raspberry Pi is critical. If insufficient power is available the Pi can fail to boot or be intermittent. The amount of power (and the power connector) differs depending on the model of Pi. Make sure you have enough power. Using a larger than needed supply is a good plan for development, who needs power problems then. Later the supply can be tailored and tested on a known working system. Ultimately an efficient 12 volt to 5 volt supply will be the choice for Field Operations, hopefully an RF quiet switching supply. Power consumption at 5 volts ranges from about half an amp peak (Pi Zero W) to about 1.5 amps peak (Pi 3B+). For development a 1.5A supply should be more than sufficient. Average power consumption is expected to be about ⅓ of the peak values. The Zero W in particular is susceptible to power issues if a USB hub is used.

There are many GPS hardware modules available that are suitable for this project. The pulse per second (PPS) signal is desirable but not absolutely required. Using a serial only GPS (with no PPS) will yield a plus or minus 100 millisecond at best time server whereas with PPS a few milliseconds is possible. If used at home the internet will be better than the no PPS GPS but using PPS will be better than the internet. The USB devices don’t generally have PPS. But they do work and are very easy to use. Plus or minus 100 milliseconds is adequate for our needs, but with just one more wire the PPS signal can give so much more. The no PPS time variation makes testing more difficult and time consuming.

GPS Modules fall into several types depending on the interface, the antenna setup and the physical arrangement. A USB dongle will plug into a standard USB port but generally they lack the precision time pulse. Hats plug onto the GPIO connectors of the Pi. They are usually full size for a regular Pi, but some half size modules exist. These usually have PPS but verify that when selecting. Small PC board modules also exist in many forms. Again some have PPS and some do not. They are generally serial but some also have USB. Antennas vary as well, with many built in and some external with tiny coax and connectors or both. Remember that the Pi inputs and outputs are 3.3V. Insure that anything you connect is compatible. The PPS signal may be brought in on a number of different pins, but GPIO4 seems to be a common choice. The software configuration will need to match the hardware here. Specifics of modules will be covered in the build appendices. The Reyax modules were very common a few years ago and quite inexpensive, but seem to be less common now. Another module, the Goouuu Tech GT-U7 (from China) seems to be very available and inexpensive in 2020. It is also based on the Neo6M chip, possibly using a cloned IC. The made in USA Adafruit Ultimate GPS modules are available in several forms including Hats and Modules. They are based on the MTK3339 chipset which is excellent. They also have various remote antenna options and some perfboard space on the hat boards.

The GPS server software “gpsd” is a service daemon that monitors the GPS hardware and makes the data available to apps on that computer and others that can access it over the network. This allows sharing of the GPS data which is useful for testing and debugging as well as running multiple GPS aware applications. It combines the separate NMEA detailed information with the precision timed PPS pulse so the applications don’t have to figure out how to do that themselves. It makes the setup easier to install and debug.

The NTP Time Server looks at various time sources and deduces the time, sets and disciplines the system time clock, and shares time information across the network with other time service clients and daemons.

There are several Time Server programs available that support the standard NTP protocol. The oldest is the original ntpd which is commonly used on Unix and many other systems. A newer version called secure NTP (NTPsec) has security enhancements primarily designed to guard against distributed denial of service attacks through NTP and is based on a cleaned up version of the original code. A newly rewritten simpler “lightweight” daemon called Chrony implements the most popular features of ‘ntpd’ without being as large or complex. For the Pi it is suggested to use either NTPsec or Chrony. We use both here in different examples.

This document is organized as a general walk through of the build process, followed by several Appendices that contain specifics for different types of installs as well as related useful information. There is more than one way to do this, so the specific installs will address different methods in some detail.

## General Overview of Builds

There are a couple of approaches to Pi builds in general. One is to install a standard Pi OS and then applications are added to it and configured. The other is to install a pre-built combination of OS and application(s). This section will outline the former. The pre-built setups are easy to install - just follow their instructions. I haven’t seen such a package for NTP service, but it may be out there, and if it matches your hardware it could be the easiest install. Performing the individual component process in distinct steps and testing at each stage is less confusing and easier to understand and adjust for differences in hardware and software. Things do tend to change especially in the software area so a fixed or scripted recipe will fall out of date more quickly.

Installing the operating system and standard software is the first order of business. The Raspberry Pi Imager is currently the official Raspberry Pi OS installer. This tool is installed on the host machine and used to prepare an appropriate SD card for the Pi with the selected OS. An 8GB (or 16GB) uSD (or SD depending on what the model of Pi requires) is about the right size. No more than about half the card should be used to allow the space management algorithms to work efficiently. A larger card would be fine too (as long as it is compatible with the Pi hardware). A USB to SD card adapter is used to interface the card on the host machine for the creation process. The install and verify takes awhile. Several operating system choices are available. The Appendices should cover the details of this choice for a particular build.

**Preparing for Wireless Headless Development**
If a keyboard/mouse/monitor will be plugged directly into the Pi this step is not needed. If the Pi will be accessed over WiFi the SD card must be prepared to connect to the network before it is plugged into the Pi for the first boot. WiFi information must be entered. Frank used this technique in his Appendix 1 build.

The next step is to put the (u)SD card in the Pi and power it up for first boot. There are various ways to do this (and lots of recipes on the internet). A local HDMI display, USB keyboard and mouse is one way which Alan used in Appendix 2.

The Pi is then booted and the initial configuration and software updates are done. The password should be set and the hostname may be changed. The local network router can optionally be configured to supply a fixed IP number to the Pi so it will be easier to find, or a search can find it and just use the dynamic IP address (as long as this doesn't keep changing). Since the plan is to use this on various portable networks in the field it is probably best not to program a fixed IP into the Pi itself.

At this point we have a running Pi system with local and/or remote access. Alan used a 10" HDMI monitor which is a real eye test but it fits in a small space. Later a 13" monitor was used which is a little easier on the eyes. Alan did enable SSH logins and used Putty to log in from a larger screen for some of the steps. Frank did everything headless and avoided the need for plugging a monitor and peripherals into the Zero W. A small USB hub can be used with the Zero W to allow direct connections if desired but very little power is available.

**Serial Port Configuration**
On many models of the Pi the hardware serial port is used for other things like Bluetooth or a serial login terminal. If we are planning to use that port for a serial GPS then these other features must be disabled. However if a USB based GPS is used it creates a new dynamic serial port separately which avoids this conflict. Even if the hardware serial is used there are ways to access the Bluetooth via a software serial if that is needed. There are bluetooth GPS's but in that case the serial hardware port would not be used so the Bluetooth could use it. We did not experiment with Bluetooth GPS's. Note that Bluetooth uses the same radio gear as WiFi so to minimize latency for a WiFi time server it is best to disable Bluetooth.

**Connect GPS hardware**
There are few connections required - ground, 5 volt power, serial in, serial out and PPS. Care needs to be taken as the Pi inputs and outputs are only 3.3V and easy to damage with 5V. If a Hat is used it should just plug right in. In some cases there are jumpers or other configurations on the hat that must be done to route the serial and PPS signals through to the Pi.

**Download Software**
One or more software packages are downloaded into the Pi - such as gpsd, the ntp daemon, tools, etc. There are a few different NTP server packages. NTP is the oldest one, but is not generally recommended for the Pi. NTPsec or Chrony are newer packages that are more suited to the Pi. Details in the Appendices.

**Configure the Software**
One of the confusing parts of setting up a time server is the many ways it can be configured to work. There are two parts to the interface - a serial connection for the GPS data, and a hardware interrupt for the PPS (pulse per second) pin. Serial connections are fairly straightforward, but the kernel configuration for the PPS is very system dependent and varied. The approach taken here is to use the GPSD software package to interface with the GPS and read the PPS from the kernel driver. This package will supply GPS data to all other software that wants it including the time server, and various test programs. GPSd is configured to read the GPS hardware, and then the NTP Timer Server daemon is configured to read the data from gpsd. Gpsmon can be used to monitor the data from gpsd when debugging. Take care not to incorrectly configure the NTP server to read directly from the hardware as this creates a race condition between it and gpsd that may be confusing to debug. Finally the NTP daemon is configured for various things like backup NTP servers it gets time from, and for the networks it delivers time to.

**Testing and Optimizing**
Getting the Pi Time Server working initially turned out to be less than half the battle. To keep clients using it the delay, jitter and latency must be low. It turned out that many of the default settings for the Pi did not result in great performance for the time service…. The standard configurations for time servers depend on other servers on the internet. When the internet is not available they fail to operate. The time server had to be tuned to operate standalone. Testing and adjustments were required to find a configuration that works in standalone mode and still acquire the time and serve it from the GPS alone.

**Jitter and Delay Reduction**
The stock operating installations often incorporate features that interfere with good time server performance. It is important for the network hardware to be ready to accept time requests and respond to them without delay and within consistent time windows. This requires adjusting some system configuration parameters.

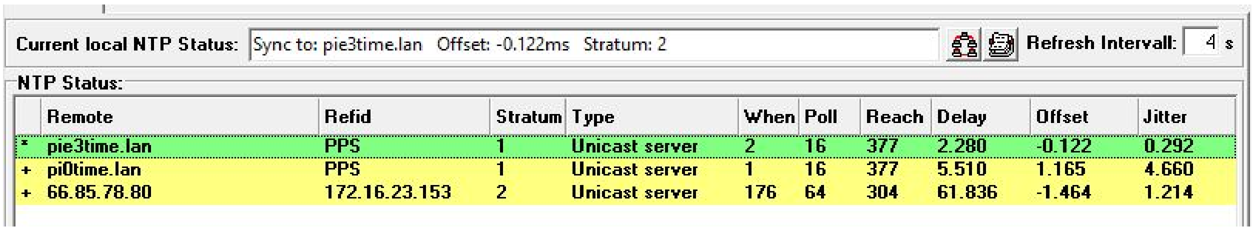
**The Leap Second Bug**
Alan has noticed an occasional 3 second error on cold starts with various GPS modules. It happened on Field Day this year (2020) and it has occurred at other times during testing as well. It goes away after some minutes. This comes from GPS leap second adjustments that are made after the full GPS nav message is decoded which can take over 12 minutes. When it occurs everything appears to be working fine, but the time is 3 seconds off, then suddenly the time is correct. The amount of this error will change as leap seconds are added to the GPS

system. Using a battery backed GPS module should fix this as the leap seconds would likely be stored in the memory. This has been noticed on both Neo6M and MTK3339 chipsets. Start your time servers well before you will need them and let them stabilize. Insure they have good access to the sky at all angles to pick up lots of satellites with the best geometry, at least initially. Once they get locked on they can maintain lock with much poorer satellite signals.

**Does Anyone Really Know What Time It Is?**

Time servers are like watches. If you have one you think you know what time it is. If you have two and they disagree you are never sure. For reliable time three or four servers are nice. With one or two clients cannot tell if or which one is broken. With three they can determine if one is inconsistent with the others. With four one can be offline and still there will be three online. If you only have two and they disagree (long term) you have to figure out which one is wrong and turn it off manually. A self setting WWVB watch or a portable GPS or reception of WWV on HF or a phone can be used to decide which servers are wrong. Any malfunctioning server should be turned off to minimize confusion in the clients. Lately I've been running two local GPS based servers and picking up a third from the internet for testing. The clients dance between the two local servers all the time, and the internet one doesn't get used much but makes a good check to visually verify the GPS's are working well. Two servers work pretty well.

Pi0 on the left, Pi3 in the middle time servers running, gpsmon is visible for both of them. The color screen is a weather station, not related to this project. It also gets time from the network.



Current local NTP Status: Sync to: pie3time.lan Offset: -0.122ms Stratum: 2 | Refresh Intervall: 4 s

NTP Status:

| | Remote | Refid | Stratum | Type | When | Poll | Reach | Delay | Offset | Jitter |
|---|---|---|---|---|---|---|---|---|---|---|
| * | pie3time.lan | PPS | 1 | Unicast server | 2 | 16 | 377 | 2.280 | -0.122 | 0.292 |
| + | pi0time.lan | PPS | 1 | Unicast server | 1 | 16 | 377 | 5.510 | 1.165 | 4.660 |
| + | 66.85.78.80 | 172.16.23.153 | 2 | Unicast server | 176 | 64 | 304 | 61.836 | -1.464 | 1.214 |

Above we have a Meinberg NTP view from the desktop computer of two of the time servers. The first is the Pi 3B+, the second is the Pi Zero W, and the third is a random from the pool that is 60 milliseconds away. The Pi3 is faster, the delay is about half the Pi0, and the jitter is lower. These values move around, but either one is more than adequate. These are the two servers detailed in Appendices 2 and 3.

## List of Appendices and Etc

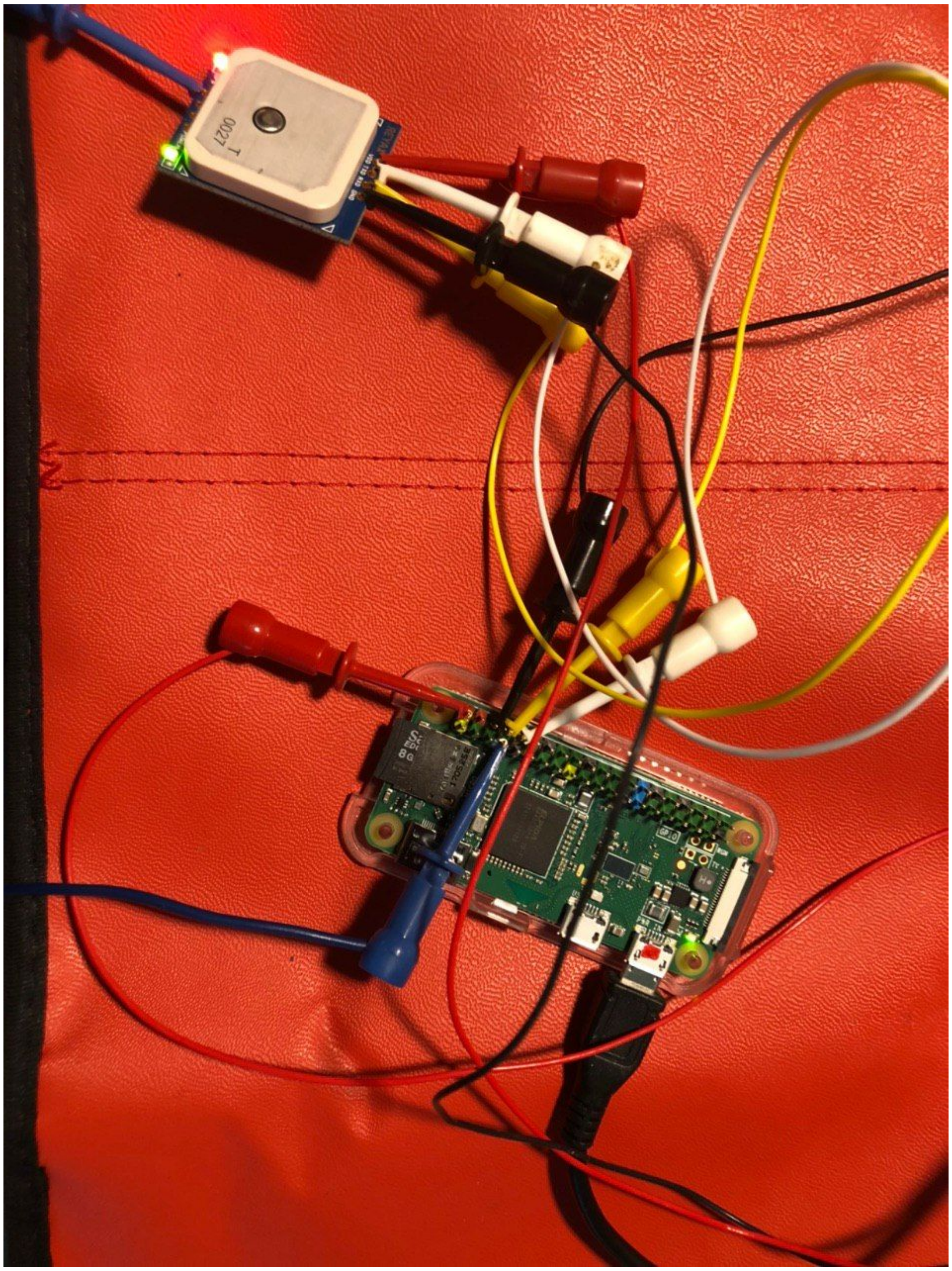
## Appendix 1 - NTPsec on a Pi Zero W

### Small is Beautiful

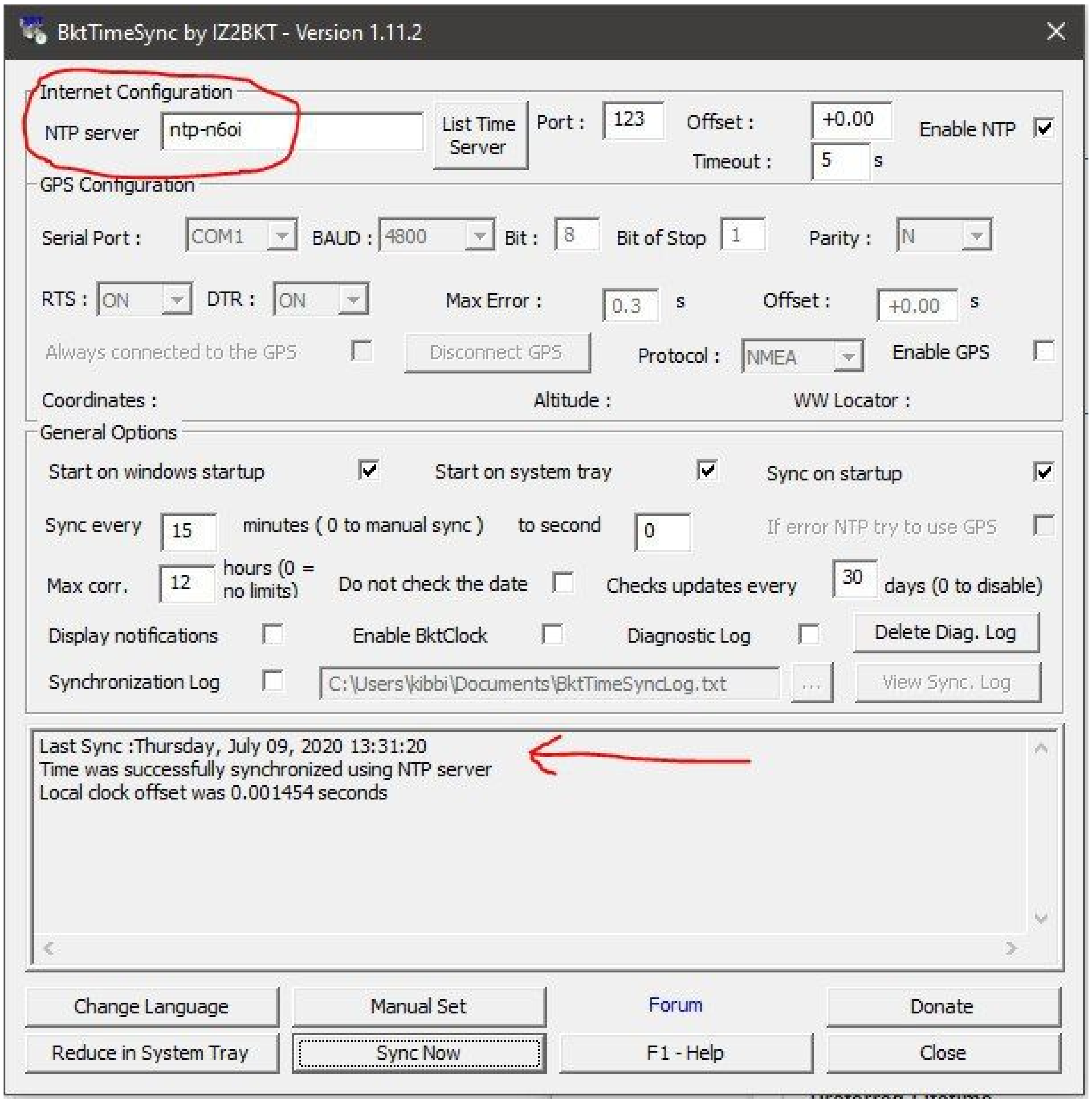
Frank N6OI

This setup utilizes a Pi Zero W and a Reyax RYN25AI (w/PPS) GPS Module.This implementation is based on the information and software documented in The Stratum-1-Microserver HOWTO by Eric S. Raymond. The article describes using an Uputronics GPS Expansion Board, but the author also includes info for the use of the Adafruit GPS HAT.  I substituted the Reyax GPS for either of those units.

The Pi Zero W is configured with Rasbpian Stretch Lite since the Pi will be operated as a headless server.  The article includes all the steps to manually configure the Pi, and also includes a link to the "Clockmaker" Python script that will automate much of the time server configuration.  For this initial implementation, I went through the manual configuration steps in order to better understand what was being installed and why.



Prototype in Test



Internet Configuration

NTP server: ntp-n6oi  Port: 123  Offset: +0.00  Timeout: 5 s  Enable NTP ☑

GPS Configuration

Serial Port: COM1  BAUD: 4800  Bit: 8  Bit of Stop 1  Parity: N

RTS: ON  DTR: ON  Max Error: 0.3 s  Offset: +0.00 s

Always connected to the GPS ☐  Protocol: NMEA  Enable GPS ☐

Coordinates:  Altitude:  WW Locator:

General Options

Start on windows startup ☑  Start on system tray ☑  Sync on startup ☑

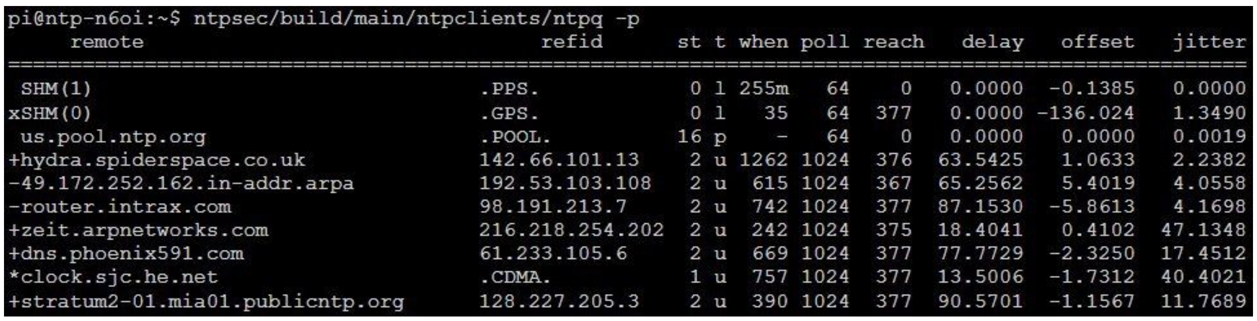
Sync every 15 minutes ( 0 to manual sync )  to second 0  If error NTP try to use GPS ☐

Max corr. 12 hours (0 = no limits)  Do not check the date ☐  Checks updates every 30 days (0 to disable)

Display notifications ☐  Enable BktClock ☐  Diagnostic Log ☐

Synchronization Log ☐  C:\Users\kibbi\Documents\BktTimeSyncLog.txt

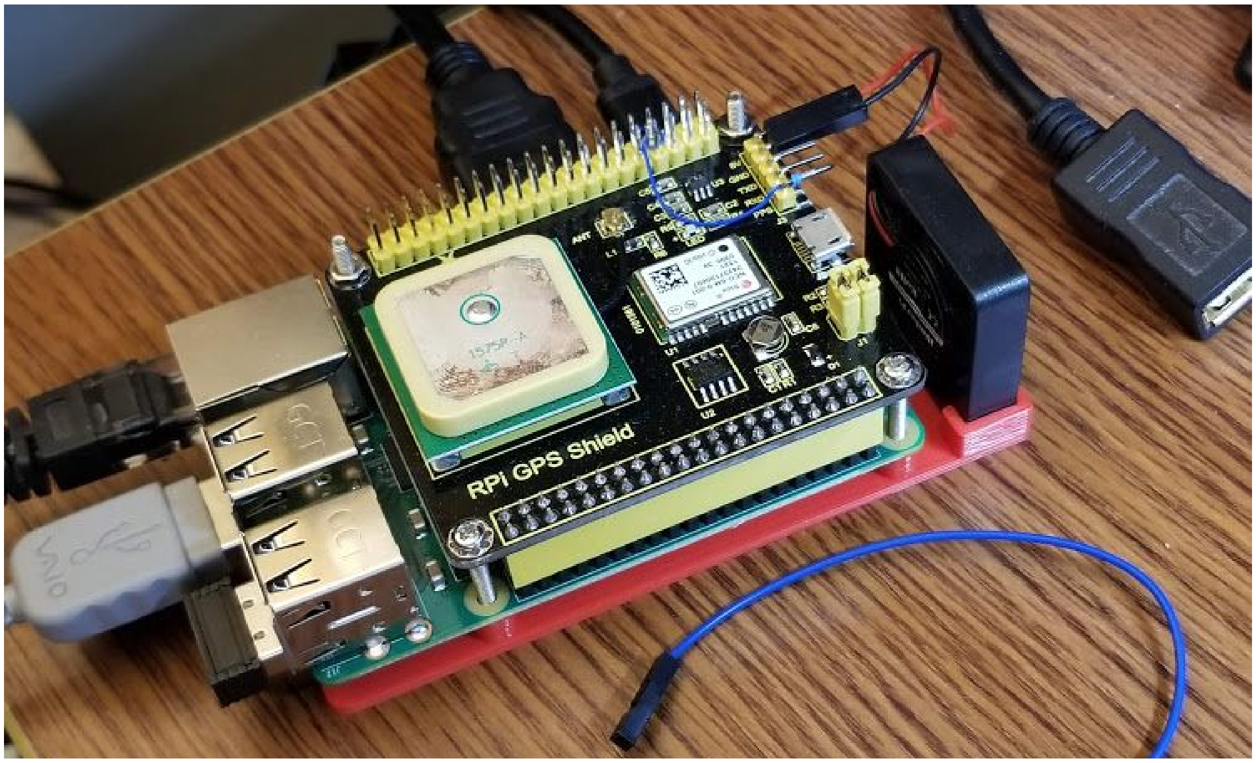
Last Sync :Thursday, July 09, 2020 13:31:20
Time was successfully synchronized using NTP server
Local clock offset was 0.001454 seconds



```
pi@ntp-n6oi:~$ ntpsec/build/main/ntpclients/ntpq -p
     remote                                   refid      st t when poll reach   delay   offset   jitter
=======================================================================================================
 SHM(1)                                  .PPS.            0 l 255m   64    0   0.0000  -0.1385   0.0000
xSHM(0)                                  .GPS.            0 l   35   64  377   0.0000 -136.024   1.3490
 us.pool.ntp.org                         .POOL.          16 p    -   64    0   0.0000   0.0000   0.0019
+hydra.spiderspace.co.uk                 142.66.101.13    2 u 1262 1024  376  63.5425   1.0633   2.2382
-49.172.252.162.in-addr.arpa             192.53.103.108   2 u  615 1024  367  65.2562   5.4019   4.0558
-router.intrax.com                       98.191.213.7     2 u  742 1024  377  87.1530  -5.8613   4.1698
+zeit.arpnetworks.com                    216.218.254.202  2 u  242 1024  375  18.4041   0.4102  47.1348
+dns.phoenix591.com                      61.233.105.6     2 u  669 1024  377  77.7729  -2.3250  17.4512
*clock.sjc.he.net                        .CDMA.           1 u  757 1024  377  13.5006  -1.7312  40.4021
+stratum2-01.mia01.publicntp.org         128.227.205.3    2 u  390 1024  377  90.5701  -1.1567  11.7689
```

## Appendix 2 - “Big Time”: Pi 3B+, GPS Hat, GPSD, Chrony

Alan W6AKB



KS0216 GPS Hat on Pi 3B+

Alan W6AKB - This is my development system for this NTP server project. The Pi 3B+ has plenty of standard connectors and all the requisite things from Wired to Wireless to USB to HDMI. It isn’t the lowest power or the smallest, but it is a very handy launchpad for most projects. Later it can be compressed down to a smaller, slower board with minimum frustration. This server has ethernet which may be desirable for a home network. In the field the wired node would only be useful on the router’s wired ports, the rest of the network is wireless. Zero W based Pi time servers only have wireless network. WiFi is adequate if ethernet is not available or desired.

I looked at building an NTP server last June and noted that the Adafruit Ultimate GPS hat for the Pi is a popular choice for an NTP server. It was a bit pricey and availability was poor  so a lower cost board was ordered, the Keyestudio KS0216. It was about half the cost (later I learned it has a probably cloned Neo6M chip on it which is a 50 channel modern GPS chip). It was available from Amazon at the time. It is still available from other sources like ebay in 2020. The documentation on the KS0216 is poor so they don’t indicate where the PPS signal goes. I asked the vendor a question about it and didn’t even get an answer. I went ahead and ordered one and put it on the shelf. Even though it works in this project I hesitate to recommend it. The onboard rechargeable battery that should speed cold starts does not appear to be working, all restarts are taking 30 seconds rather than the less than the 3 seconds a warm start should need. The Adafruit Ultimate GPS Modules and Hats are excellent choices. They have a 66 GPS channel genuine chipset and a replaceable coin lithium battery slot and they do warm starts in about four seconds. That being said, the KS0216 does a really good job of capturing the satellite signals even when cold starting indoors. The Adafruit seemed to have more trouble on the first cold start, but with the battery onboard the warm starts were amazingly quick after that initial extended cold start.

The KS0216 GPS hat was plugged in and bolted to a Raspberry Pi 3B+ CPU. This connected the power, ground and serial but as I found it did not connect the PPS signal through to the Pi. A scope was used to verify this. A jumper wire was added to connect the PPS pin on the edge of the KS0216 board with GPIO4 on Pin 7. It is the blue jumper in the photo above. It is on the outside row of pins with three empty pins to the right in this view. The PPS pin on the right edge is labelled, it is also the closest pin to the USB connector on the KS0216 board. The following recipe should work for other hats or hard wired GPS modules that use the hardware serial port pins and GPIO4 for the PPS signal.

Since the hat board blocks the natural airflow from the CPU and chipset things got a bit warm, so a Pi4 fan (5V 100mA) was connected and used to provide some cooling. Using a GPS module instead of a hat would allow more flexibility in mounting and cooling. Connecting this fan to the 3.3V supply made it quieter.

In the photo above we have 3D printed a base for the Pi stack that has a slot to hold the fan, and we have changed the original push-on test jumper to a wrapped wire connection.

Several days of intense debugging ensued. It was easy to find recipes on the internet that partly worked, but a number of issues had to be resolved to get true standalone operation. Out of many pages of notes the following recipe was teased out, and many blind alleys were discarded. Let me know if you have suggestions to improve it.

In the following I'm using <editor> where you can choose your favorite linux editor. Nano and vi are popular choices.

**Hardware List**
Raspberry Pi 3B+
8, 16 or 32GB Micro SD Card and USB Adapter for initially programming it
Keyestudio KS0216 GPS Board (or Adafruit Ultimate GPS Pi Hat)
Raspberry Pi Power Supply
USB Keyboard and Mouse
Small HDMI Monitor and cable
Pi4 Cooling Fan

3D Printed Base

**Condensed Recipe for Pi 3B+ with PPS on GPIO4 (pin 7) and Hardware Serial to GPS**

Download Raspberry Pi imager.exe from [https://www.raspberrypi.org/downloads](https://www.raspberrypi.org/downloads)
Install and run imager (I used V1.3)
Choose Raspberry Pi OS 32 bit (Raspbian Debian 10 (buster))(Release 3.3 May 2020)
Install to uSD card and verify (takes awhile)

Put uSD card into Pi, plug in keyboard and monitor and power it up
It resizes the SD card and reboots
Raspberry Pi Desktop comes up, click Next
Set country, language, timezone and keyboard, click Next
Set password for pi user account, click Next
Set up screen, click box if there's a black border around screen, click Next
Select wireless network, click Next,
Enter wireless network password, click Next
Update software by clicking Next (this will take awhile)
In local network router set Mac address of the Pi to map to desired IP address and name for the Pi (optional, this will take effect at next reboot)
When Pi software updates are complete, Click Restart

Configure the system
sudo raspi-config
Network options / set hostname
Boot options / desktop / console autologin (Disable X, autologin on boot, optional)
Interfacing / ssh / yes (allow network login)
Enable vnc????
Interfacing / serial / login no / enable serial port yes (disable serial port login shell)
Finish
Reboot yes

Install software
sudo apt -y install gpsd gpsd-clients python-gps chrony python-gi-cairo

Add two lines at the end of config.txt to disable bluetooth and setup kernel pps on GPIO4
sudo <editor> /boot/config.txt
dtoverlay=disable-bt
dtoverlay=pps-gpio,gpiopin=4
Save File and Exit

Make four lines in the gpsd config look like these to read serial gps and pps
sudo <editor> /etc/default/gpsd

```
START_DAEMON=”true”
USBAUTO=”false”
DEVICES=””
GPSD_OPTIONS=”/dev/ttyAMA0 /dev/pps0 -n”
```
Save File and Exit

Make gpsd run automagically at every reboot
```
sudo systemctl enable gpsd
```

Comment out net pool line to force chrony to operate in standalone GPS mode
```
sudo <editor> /etc/chrony/chrony.conf
# pool 2.debian.pool.ntp.org iburst
```
Add 3 lines to end of the chrony.conf file to connect chrony to gpsd and allow client access
```
refclock SOCK /run/chrony.ttyAMA0.sock refid GPS precision 1e-1 offset 0.9999
refclock SOCK /run/chrony.pps0.sock refid PPS precision 1e-7 trust
allow
```
Save File and Exit

Add iwconfig line at end of rc.local right above the exit 0 line to reduce wireless latency
```
sudo <editor> /etc/rc.local
iwconfig wlan0 power off
```
Save File, Exit

Shutdown and Install GPS hardware (if not already installed)

Power Up or Reboot

Review System Messages to verify that gpsd and chrony came up and didn’t fail, etc…..

more

**Testing**
The only way to really test a time server is to compare it with others. For this I installed Meinberg NTP and Meinberg NTP Monitor for Windows on a couple of test clients. Bring up the Meinberg Monitor by right clicking on it and selecting “run as administrator” then select toe NTP Configuration File tab. (Note that if this file is empty a default file will need to be generated, there’s a button for that ...). Carefully comment out with # in the first column all but two of the existing server lines. Then add your own server line:
```
server <ip of your server> iburst maxpoll 4
```

Hit save configuration, and restart the NTP service when it offers the dialog to do so
(note that if the restart fails you should kill the Meinberg Monitor and restart it with administrative privilege, and then restart the NTP service)

Click on the NTP Status tab

You should see three server lines, yours and two others from the internet. From this you will be able to see how your server compares to the internet servers. If your GPS has a good lock and the software is all configured and connected properly you should have a server that is lower delay and less jitter than the internet servers by about a factor of 10. I generally see under 3 milliseconds of delay and much less than 1 millisecond of jitter. The offset will head toward zero, it may take awhile but it should get well under 1 millisecond. This is from a wireless Pi time server to a wired desktop client on the same network. To a wireless laptop on the same network the numbers are about double. It may take awhile to settle down to these values.

**Debugging**
When things go wrong…
more...

**Adding a Web Server**
When we are off network it is useful to have a collection of software and files available. Using a web server on the time server is an easy way to make these accessible. To add a web server to a Raspberry Pi server enable Apache with:

```
sudo apt install apache2 -y
```

Make a directory at /var/www/html to contain files:

```
sudo mkdir /var/www/html/<dir>
sudo chown pi /var/www/html/<dir>
```

Populate it with files and directories as desired. Windows Secure Copy (WinSCP) is a good tool to use to populate and curate these files.
Visit it on the web by pointing a web server at <hostname>/<dir>

**Upgrading to a larger SD card and making Backups**
Since adding a web file repository server functionality the 8GB card was getting too full, so put a 32G card in a USB adapter and plug it into the Pi. Run the Pi SD Card Copier from the graphical menu. Specify the source and destination devices and click the 'use new partition uuids'. In a few minutes a bootable backup copy will be made of the system. This technique is also useful to make a backup copy of the SD card in case it gets corrupted. In retrospect I should have started this build with a larger SD card and reserved the smaller 8GB one for a Pi Zero project. This Pi3 has become a more capable tool with the web server and benefits from the larger uSD card. Making a backup card is desirable in any case. 16GB is likely sufficient. With a bunch of files already loaded the 32G is only 16% used.

**Next**
Moving to the Adafruit Ultimate GPS Hat
Adding the DS3231 Precision Real Time Clock
Use long pin headers on the GPS Hat so the RTC can plug directly in

# Appendix 3 - “Small Time”: Pi Zero W, GPSD, Chrony

Alan W6AKB

Pi Zero W and a pair of pre-prototype GPS stacks

The Pi 3B+ covered in Appendix 2 was the development system, this is the “production” model for field use. It is optimized for low power and small size. The I/O connector is the same as the Pi 3B+, and it is using the same operating system so the same instructions should work.  The smaller memory footprint may be an issue. (It wasn’t). We’ll start with the bare board and get the system going, this verifies that the new board is working.

**Hardware List**
Raspberry Pi Zero W
Header Pins 2x20 to install on Pi (or Pi with pins preinstalled)
8 or 16GB Micro SD Card and USB Adapter for initial programming
Adafruit Ultimate GPS Hat with supplied female socket (for testing)
CR1220 lithium coin cell for GPS backup

Raspberry Pi Power Supply
Mini HDMI to HDMI cable
HDMI Monitor
Micro USB to USB hub
USB Keyboard and Mouse

To start with, load the Raspbian 32 bit OS onto a micro SD as in Appendix 2, install it in the Pi, connect up the mini-HDMI adapter to the monitor, and plug in the USB hub with Keyboard and Mouse. Hook up the power supply (1.5 amps recommended) and fire it up.. A green LED near the power connector should come on, it flickers when there is SD card activity. So during software updates it is quite busy. Otherwise it just stays on. The Pi Zero W is a slower CPU than the Pi 3B+ used in the previous build (Appendix 2), so things proceed at a lesser pace. Especially when it comes to software updates.

Shut down X? For the moment I'm leaving it running. It is useful to use and doesn't seem to be taking too many resources. These Pi's have quite a lot available, even this small one.

The GPS I'm going to install initially here is the Adafruit Ultimate GPS Hat. It is convenient to develop with (plug and play) however it is double the size of the ZeroW so later I'll change to something more compact but using the same pinout so there should be no software changes required. The GPSD program will handle various GPS's so I don't need to worry about the differences between them for basic operations. There is some GPS tuning that might be useful that will depend on which chipset it is, but we'll get to that later. The pinout we're using is PPS on GPIO4 and Serial on the Hardware Serial lines as with the Pi 3B+. So we have to disable the Bluetooth and configure the PPS to the proper pin. I'll program the network to give this Pi ZeroW server a slightly different network address so I can run both servers at once and compare the timing.

Did the rest of the software config with one exception - left the pool line in chrony.conf alone so chrony will sync with the debian pool and serve this time initially because the GPS is not yet installed. Setup the desktop client to include the new pizerotime server ip number in its config file. Restarted things and observed the desktop Meinberg NTP status. It is picking up the new server with a delay of 6.2 milliseconds compared to 2.5 milliseconds for the pi3 server. Jitter is 0.8 mS compared to 0.5 mS, but this changes so not precise. Generally the delay and jitter are about 2-3 times worse on the Zero W compared to the 3B+. At any rate the pi0time server is up and running, but without GPS time signals. The Pi Zero W server is using 2.4 ghz wireless while the Pi 3B+ is using 5 ghz on a dual band mesh local network.

The Pi Zero W and the Ultimate GPS Hat both need the connectors soldered. Use antistatic procedures during this process and make sure to put them on correctly, referring to the appropriate documentation. Plug the cables back into the Zero and plug the hat in, ensuring nothing is shorted under the hat. Apply power and verify that it comes up.

The pins were soldered on the Zero W board, and the socket soldered to the Adafruit Ultimate GPS Hat. I fired it up and had some trouble with the keyboard, suggesting there is a power shortage. The supply is large so if there is a shortage it has to do with onboard limitations. The GPS did not lock up indoors even though it ran a long time. I plugged on the amplified antenna and it locked up in a few minutes. Later I tested it without the antenna and it relocked in 4 seconds. The very first time a GPS is operated it helps to have good signals to get it started and to be very patient. The signals with the amplified antenna were definitely stronger. The backup battery on the Ultimate GPS is good for a couple hundred days of standby, so if you are powering it off for months take the battery out. If you don't have the amplified antenna take the unit outside and away from the house so it can see a large swath of sky. Give it enough time to get a lock and the full GPS Nav message, at least 20 and preferably 30 or more minutes if the signals are not good.

After it got a lock the Zero started serving PPS time. So now we have two PPS time servers running. I moved them a few feet apart initially as GPS receivers can interfere with one another. Later I ran them about six inches apart and they were fine. Signal strengths were generally similar on the same satellite.

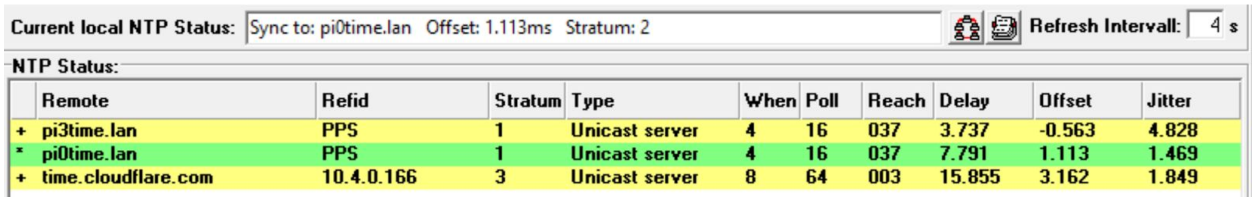
This is working now but mechanically the Ultimate GPS Hat is not a great fit for the Zero W. It would be more at home on a full sized Pi. Perhaps I should move it to 3B+ and retire the KS0216 which has a defective battery.

For the Zero W it would be nice to find a GPS Hat that is the same size. Looking around the internet I have seen a couple but they were not current commercial products. So perhaps the thing to do here is fabricate a small plug in board that takes a GPS module that is physically compatible with the Zero. The genuine USA Adafruit Ultimate GPS Module is small enough for this and has the slot for the coin battery as well as a socket for an optional external amplified antenna. Alternatively a low cost Chinese GPS module could be used. Five connections are required. I would use a short female socket on it to leave the other end of the GPIO connector open for a DS3231 Real Time Clock Module. I'll cover that combination in a separate Appendix here, but the goal there is to have a backup clock for the Pi that doesn't require time to lock up or sky access to work. In field work we may be in a situation where the GPS doesn't work for whatever reason, so this combination will give a backup time source for the Zero that it can serve at a higher stratum level when the GPS module is not locked up. The GPS can be used to set and discipline the DS3231 clock and then it can be used for weeks with adequate time accuracy for our needs

## Appendix 4 - Meinberg NTP Client for Windows

Alan W6AKB

The Meinberg company makes commercial NTP time servers, and as part of their business model they provide NTP clients for Windows at no charge. They provide GUI installer packages for the NTP service and associated programs that have been compiled from the original public NTP source code. So this is the "real NTP code". They have variants for different Windows operating system versions. Their web pages at meinbergglobal.com have more information and the download page. They also have a graphical monitor program and a cheat sheet for the NTP configuration options.



Current local NTP Status: Sync to: pi0time.lan Offset: 1.113ms Stratum: 2

Refresh Intervall: 4 s

NTP Status:

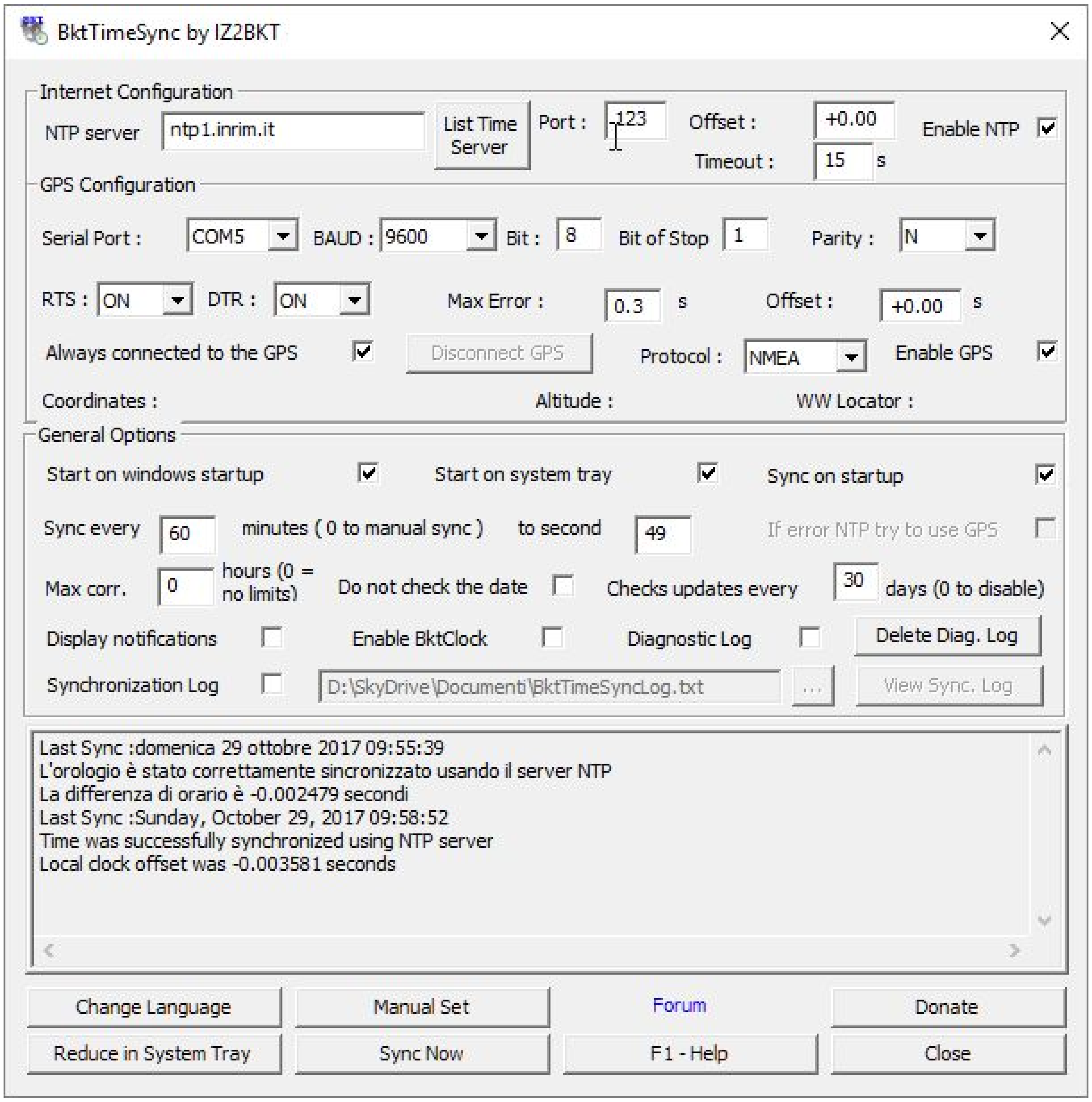
| | Remote | Refid | Stratum | Type | When | Poll | Reach | Delay | Offset | Jitter |
|---|---|---|---|---|---|---|---|---|---|---|
| + | pi3time.lan | PPS | 1 | Unicast server | 4 | 16 | 037 | 3.737 | -0.563 | 4.828 |
| * | pi0time.lan | PPS | 1 | Unicast server | 4 | 16 | 037 | 7.791 | 1.113 | 1.469 |
| + | time.cloudflare.com | 10.4.0.166 | 3 | Unicast server | 8 | 64 | 003 | 15.855 | 3.162 | 1.849 |

Here Meinberg is running on a 2020 Dell XPS 13 looking at the two time servers covered in Appendices 2 and 3. The Dell is on WiFi as are both time servers so the delays are somewhat larger than when viewed from the wired desktop. Here we see delays of 3.7 and 7.8 milliseconds. The offset drifts around as NTP follows the selected server which is the green bar. The jitter is the recent variation in delay, and this snapshot was taken soon after a restart of the NTP daemon on the laptop so it was still settling in. The Reach of 037 is an octal representation of the last 8 polls to each server, and here it shows that 5 polls have been made and they were all successful. There's a config file that by default pulls servers in from the pool but in this case I have edited the file and added my two servers and commented most of the defaults out leaving just one. NTP picks one primary and two secondary servers and I want to force it to show my two plus one other so I can compare the three offsets and have confidence in the correct configuration of my GPS based time servers. They are presently configured only for the GPS and PPS and each other so they won't fall back on the internet and from the Refid here we can see they are presently locked to the PPS which makes them stratum 1 servers. I've also configured maxpoll to 4 for my servers so they are polled every 16 seconds while the internet servers are at the defaults with a longer poll cycle as required by open NTP server policy.

I have noticed that after I wake up the laptop it often fails to connect to my local servers while it will connect to the internet servers. There's a button to restart NTP and that resolves it.

## Appendix 5 - btkTimeSync NTP Client for Windows

btkTimeSync by Mania Radio is a free Windows NTP client that is, in many ways, similar to NetTime, Dimension4 or Meinberg.  Installation is simple, and once installed the user simply selects a ntp time server from an available list.



```
Last Sync :domenica 29 ottobre 2017 09:55:39
L'orologio è stato correttamente sincronizzato usando il server NTP
La differenza di orario è -0.002479 secondi
Last Sync :Sunday, October 29, 2017 09:58:52
Time was successfully synchronized using NTP server
Local clock offset was -0.003581 seconds
```

It is not as full featured as the Meinburg client, but, unlike Dimension4 or NetTime, it does also allows for a locally connected (optional) GPS. If the GPS is available, the client will take time references from there whether an internet connection is available or not.

# Appendix 6 - FDLog logging program and the GPS FDLog Clock

Alan W6AKB

# Appendix 7 - List of Some Pi Commands and Files

**system**

```
tail /var/log/syslog
grep chrony /var/log/syslog
dmesg
ifconfig
iwconfig
sudo nano /etc/network/interfaces (no changes)
/etc/rc.local (add iwconfig wlan0 power off)
cat /etc/os-release
```

**tools**

```
sudo
vi
nano
cat
grep
```

**systemctl**

has gpsd.service and gpsd.socket
operations status, start, stop, restart, reload, reload-or-restart, enable, disable

```
sudo systemctl enable gpsd
sudo systemctl disable gpsd
sudo systemctl unmask gpsd
```

**gpsd**

`sudo vi /etc/default/gpsd`
`gpsmon`  shows gps and pps data that gpsd has
`gpspipe -r`  pipes gps data to standard output

`cgps`
`xgps` (requires x)

`ipcrm -m` to look for shared memory

`ntpshmmon`  to see time corrections in shared memory

**chrony**

```
sudo vi /etc/chrony/chrony.conf
chronyc sources
chronyc tracking
```

**pps-tools**
apt install pps-tools
ppstest /dev/pps0

**devices**
/dev/ttyAMA0
/dev/pps0

## Appendix 8 - Pi Header Pins for GPS

These are the power, serial and PPS pins for the ZeroW and 3B+ and other Pi's
The I2C pins are also in this area and may be useful for local displays or other I/O

pin 1 3.3V power (square pad)
pin 2 5V power
pin 3 GPIO2 SDA1 I2C
pin 4 5V power (GPS 5V power)
pin 5 SCL1 I2C
pin 6 ground (GPS Ground)
-- Pi clock module uses above six pins
pin 7 gpio4 (3.3V) (PPS)
pin 8 gpio14 uart0_txd (3.3V)(GPS serial in)
pin 9 ground
pin 10 gpio15 uart0_rxd (3.3V)(GPS serial out)

## Appendix 9 - Zero W NTP Server with DS3231 RTC Backup

Add a battery backed real time clock chip and configure the NTP server to use it as a backup for those times when GPS isn't working. It should be stable to within a few milliseconds per day so adequate for many things at least for a while.

more...

## Appendix 10 - Rock Pi S NTP Server

Eric WD6CMU

This is probably the smallest time server. Perhaps Eric will document his here.

More…

## Appendix 11 - Configuring NTP Time Services for Field Ops

Alan W6AKB

On Field Day our group sets up a WiFi network in the forest. We use it to interconnect the stations logging computers. We don't generally connect this network to the internet because it doesn't need it, and we often don't have a reliable way to do that. If we have internet access it is generally by cellular or on a separate WiFi network. This is a plan for configuring the station logging network, time servers, and clients for NTP time synchronization. We assume that either there is no internet access on this network, or that there is limited and unreliable access. So time servers on this network need to be configured to get time from either the GPS or from a known accurate crystal oscillator driven clock. In this note I address the Field Day event but clearly this same plan applies to other events that stand up an isolated or potentially isolated WiFi network.

The network router gives out addresses for the network via DHCP. This can often be configured to assign a fixed IP address based on their unique MAC address. The DHCP assigned static IP address configuration is preferred to configuring the individual time servers, however than will also work. Here’s an example of how this might be laid out:

192.168.1.x network layout

192.168.1.1 router
192.168.1.11 time server 1
192.168.1.12 time server 2
192.168.1.13 time server 3
192.168.1.14 time server 4
192.168.1.15 time server 5

<other servers that benefit from fixed addresses>

192.168.1.100-250  DHCP Dynamic Address Pool (example)

The client machines must be running NTP compatible time clients that are adequate for digital mode time sync. This generally means the Windoze time client is insufficient since it has only a plus or minus 5 second rating. Some other time client must be loaded and configured on each machine. This should be done long before Field Day and the computer user should be familiar with configuring their chosen tool since it will probably require reconfiguration for Field Day to include the portable network’s time servers. Depending on the client the additional line(s) for Field Day servers may be left in the configuration all the time and they will effectively be ignored when there are no time servers at those addresses.

Some systems already have adequate time software installed, Unix, Linux and Macintosh for example have NTP programs already onboard such as NTPD, NTPSEC or Chrony. For Windows clients the built in programs are not sufficiently accurate as they only have a 5 second requirement for sync. So third party time sync programs like Meinberg NTP, BktTimeSync or Dimension 4 are popular. See separate Appendices for more info on these programs.

The time server configurations could refer to each other as backup. I’ve done that here with the two GPS PPS servers, each refers to the other as a server. While the GPS is unlocked they can still sync their clocks if the other server is online, and they can serve time as well. Once their own GPS locks up they will switch to that and start serving it.

## Appendix 12 - A Partial Survey of NTP Clients for Windows

**Meinberg NTP**

Uses the real public NTP source code, compiled for windows, and a graphical monitor/interface to it
Uses 3 servers from a larger pool, follows one and tracks two more for verification
Extremely configurable and sophisticated client
Syncs once per minute (configurable)
Warps time smoothly (can be configured to jump when the difference is large)
Develops clock compensations

**BktTimeSync**
Uses one server
Syncs once per hour
Can read local GPS but not PPS, only accurate to about 300 mS from GPS (with tuning)

**Dimension 4**
More??

## Appendix 13 - Ublox Clone Modules

I’m just starting to research the Ublox Clone Module issues. The really low cost Ublox modules are likely clones rather than licensed units. If the code on the module is a QR code it is almost certainly fake. If it is a data matrix code it may or may not be fake. Attempting a firmware upgrade may reveal the fake. One video said that in the clone modules the actual GPS chip is real, but they leave out the updatable firmware chip so it is stuck at whatever firmware revision is inside the chip. In the long term this may be important as new changes are needed to the code. At some point the GPS module may have to be replaced since it cannot be updated with new firmware. This is a good reason to make it plug-in and facilitate swapping the module for testing or upgrading later.

## About the Authors -

Alan Biocca W6AKB (formerly WB6ZQZ) - Often known as “The Antenna Launcher Guy” (see www.AntennaLaunchers.com) Alan is actually a Computer Scientist and Electrical Engineer retired from a long career developing and supporting accelerator control and data acquisition systems for Science at Lawrence Berkeley National Lab. He has been a licensed Amateur Radio Operator since he was very young. He likes to design and make things and does a lot of CAD design for 3D printing. Field Day and portable operation has always been a passion of his. He also likes to play tennis and camp in the mountains. He can be reached via email through w6akb at ARRL.net. Please send him any feedback you have on this document.

Frank Kibbish N6OI (formerly WB6MRQ) is a retired Silicon Valley professional who spent the last 19 years of his career as a Business Systems Analyst, supporting IT Sales and Marketing software systems.  Licensed in 1975 as WN6MRQ, he enjoys multiple areas of amateur radio, including DX, digital modes (HF, VHF and UHF), public service (36 years as a Cal Fire Volunteer In Prevention), and contesting. Frank also enjoys building his own shack accessories, this NTP server being just one example.  Outside of ham radio, he enjoys camping, international travel, and playing the occasional round of golf.  He can be reached at n6oi at ARRL dot net.

Eric W WD6CMU?

## Bibliography and Links

https://photobyte.org/raspberry-pi-stretch-gps-dongle-as-a-time-source-with-chrony-timedatectl/

More…

eof