GPS-based Network Time Protocol Server On Raspberry Pi

by Gordon Gibby KX4Z

file: 2023/JUN/GPSrpisetup.odt
file: qsl.net/nf4rc/2023/GPSrpisetup.pdf

Our club wanted a stand-alone GPS-based network time protocol (NTP) server to easily allow FT8/FT4 type communications on a private network without Internet (e.g. remote Field Day). In a previous document, creating a Windows-based GPS using an inexpensive GPS USB dongle was documented. Unfortunately, the available laptop, an inexpensive Evolve 3, had USB-driver issues and would randomly corrupt the networking.

If your raspberry pi ntp server is already in place, you will probabaly want to skip to **Section II - Use** and avoid some of the nitty-gritty details in **Section I**.

I. INITIAL CONSTRUCTION OF GPS-BASED NTP SERVER

Raspberry Pi 2

With raspberry pi prices coming back down out of the stratosphere, I pursued creating the same simple system on that Linux-based platform. I had available one Raspberry Pi 2 A truly superb GPS-based NTP server requires the 1 pulse per second (PPS) output that some advanced GPS antennas and several Adafruit projects provide, but for our purposes, a simple system with generic u-blox-based GPS dongles would suffice. A similar installation should work on a Pi 3 or Pi 4. The Pi 2 does not have built-in WIFI. It also does not by itself have a "real time clock." I utilized a simple Tenda home router in "wireless repeater" mode to allow it to reach the local network by WIFI.

NOTE: If the raspberry pi ntp-server is operating without a real-time clock, its internal clock must be within approximately 4 hours (approx 14,400 seconds) in order for `gpsd/ntp` to lock to gps-derived accurate time. Without a real time clock, it may simply start after a power-off period, at the "time" it was last powered, which might well be more than 4 hours off. One way to correct the time is with the command

```
sudo date -s HHmm
```

e.g. `sudo date -s 1130`

I found a useful youtube from Ron Nutter that got me started: https://www.youtube.com/watch?v=sLSSg_-mwuE However, I soon needed a lot more information.

First, I added packages that provide support for a gps antenna (gpsd, a "daemon" background process) and for ntp (which includes the ntp daemon). In the following commands, "sudo" allows the command to proceed as if from root:

```
sudo apt-get install gpsd gpsd-clients python-gps
sudo apt-get install ntp
```

In my case, the "python" portion didn't seem to work. I tried adding the python system using raspberry pi "add software" options but this didn't seem to help and I don't think I ended up using python anyway.)

Executing

```
sudo lusb
```

both before and after inserting the USB connector from the GPS dongle allows you to verify that the operating system observed it. Look for a new device in the after execution that includes u-blox somewhere.

Then immediately executing

```
cat /var/log/syslog | more
```

and looking for information on a new `dev` will help you find how it was labeled by the operating system. In my case it appeared to be both

`ttyAMA0` and also

`gps0` (note no "tty")

which can both be found in the subdirectory `/dev`

There is a command that will allow you to read the output from the GPS dongle:

```
stty -F /dev/XXX ispeed 4800 && cat </dev/XXX
```

where you insert for XXX the device names, such as `gps0` or `ttyAMA0`. This worked both ways for me.

To start the gpsd (daemon) Nutter recommends:

```
sudo gpsd /dev/ttyACM0 -n -F /var/run/gpsd.sock
```

but it isn't apparent in his video, how this would make it begin again automatically on each reboot. To accomplish that I used:

```
sudo service gpsd start
```

```
sudo service gpsd enable
```

It turns out that this creates the necessary files for systemd to start this program (at least on my raspberry pi 2 operating system). There are other ways involving systemctl to find which services are running and to arrange for services to start at boot, restart, and stop.[1] You can check for the existence and performance of the gpsd daemon several ways:

`cgps` executed in a terminal windows will give you a nice display, using data captured from port 2947 from `gpsd`

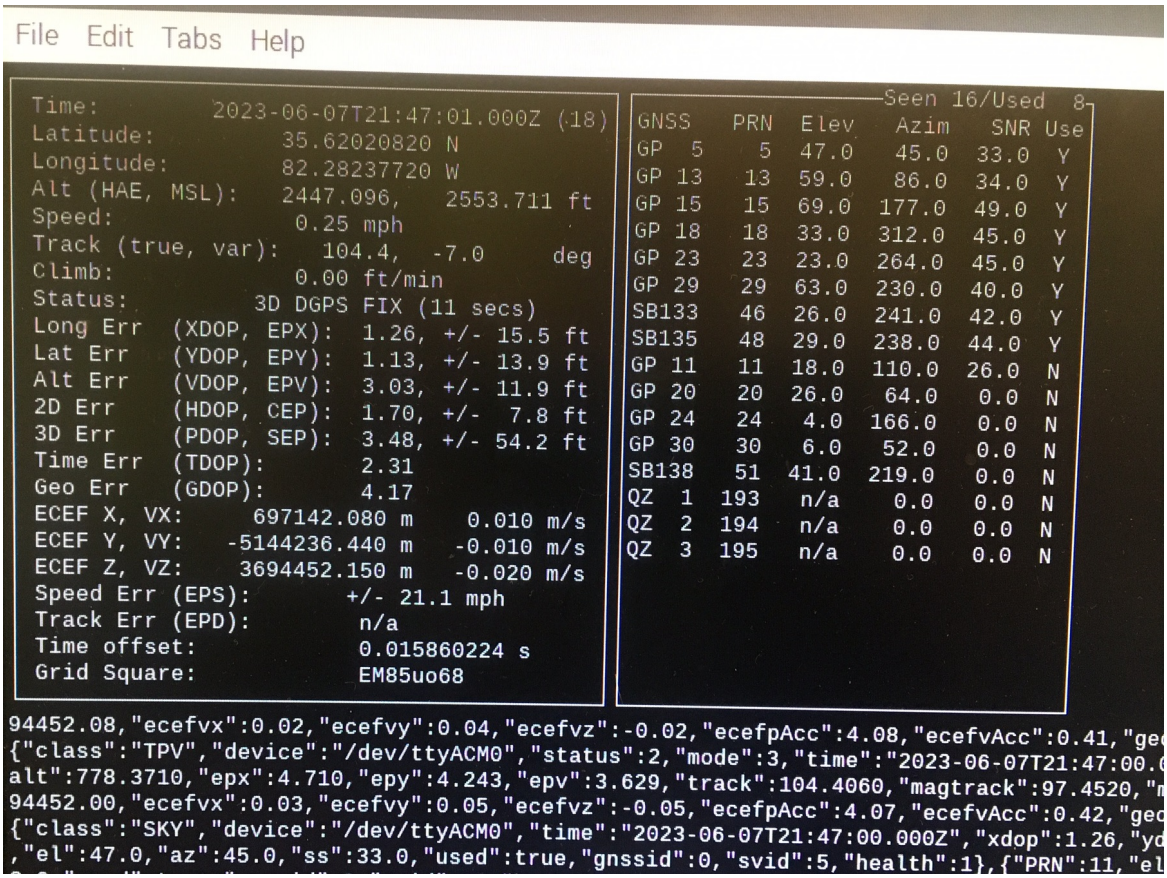

Figure: `cgps` output. Note that it shows a 3D FIX, and also gives the Grid Square. 8 Satellites are being used.

`gpsmon` executed in a terminal window will also give you a nice display, pulling directly from the dongle's USB port: (here named ACM0, in "abstract control model" nomenclature for the port)

1 Check out this informative article: https://www.redhat.com/sysadmin/systemd-commands#:~:text=To check a service's status,systemctl status service-name command.

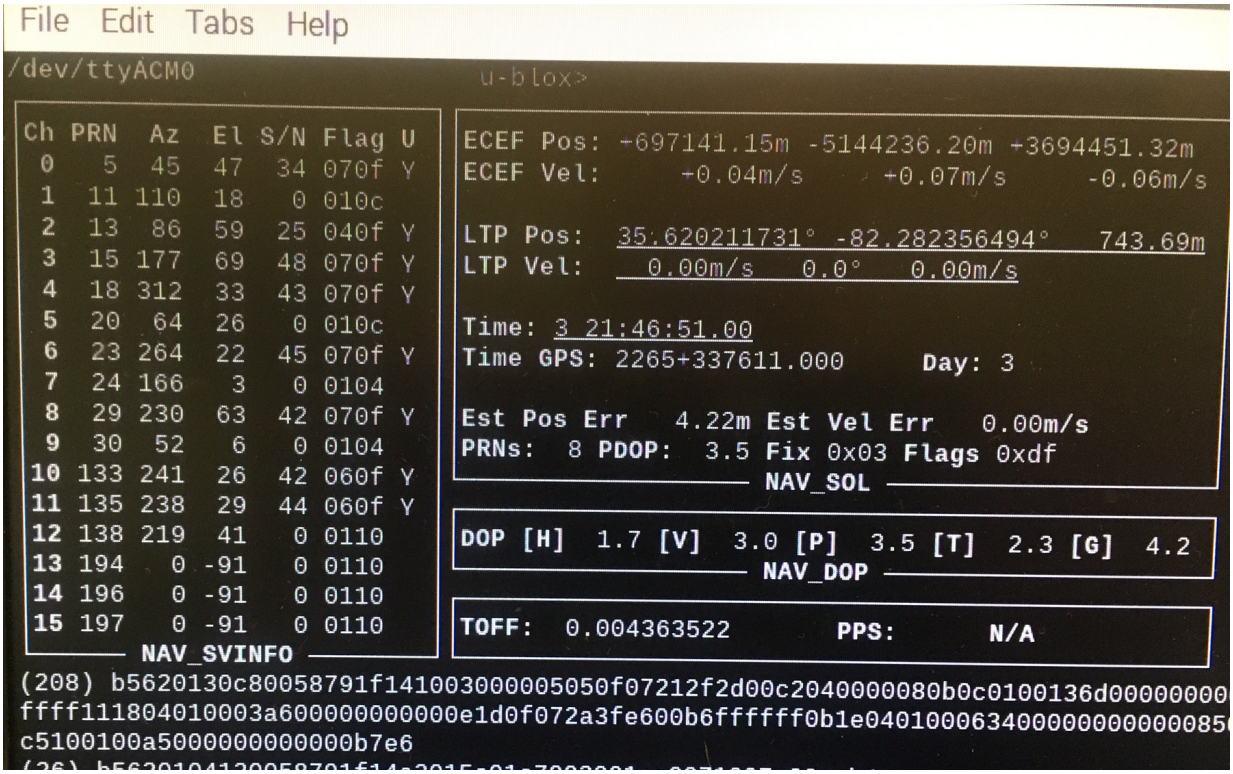

Figure: gpsmon output. Note on the top line it shows which device is providing the data. In thetop box on the right you can see the Fix is at 0x03 status. In the bottom box on the right, there is no pulse-per-second (PPS) intput. 8 Satellites are being used.

I also used this command many, many times to see if the service were running and how it was working:

```
service gpsd status
```

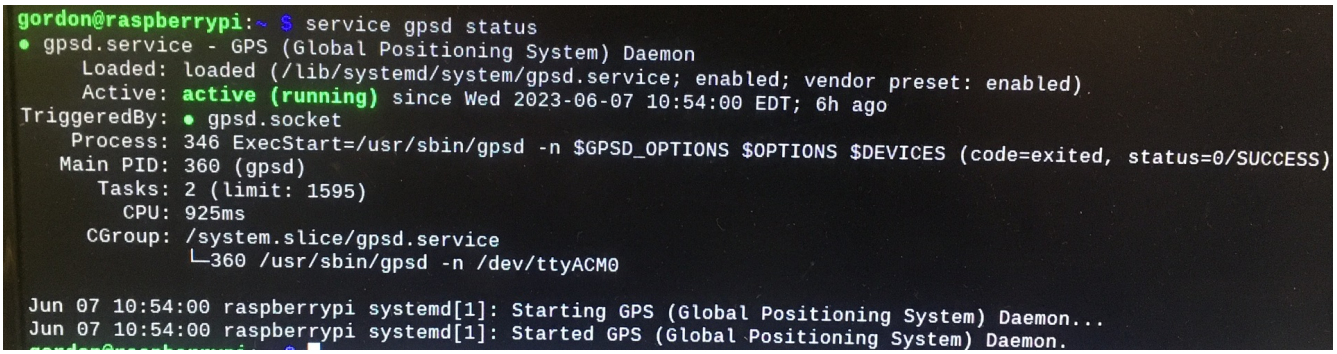

Figure: Output of `service gpsd status`. Note that it indicates the service is running, gives the processs id, shows the command by which it was started (which includes the -n option) and the device it is reading (`/dev/ttyACM0`) and shows the time it was starting and actually started.

The files that start these services are found in `/lib/systemd/system`
and there I found `gpsd.service`

To see if the gpsd is providing socket-based output connections on its customary 2947 socket, you can

```
telnet 127.0.0.1 2947
```

and verify that you get a connection and some information

There may be delays in gpsd providing information on a raspberry pi without a real time clock. There are potentially due to gpsd not ascertaining enough stability in its mathematics establishing a fix from its gps inputs. I have found much better success in getting gpsd to provide information to the ntp daemon, with two changes to the stock setup on my raspberry pi 2:

Fix the baud rate to 4800 (the default of VK-162 USB dongles)
Command gpsd to release data even if the accuracy is not established. (A feature more useful on a raspberry pi WITHOUT a real time clock that might still have significant accuracy.)

In order to accomplish these goals (which resulted in much more repeatable and quicker linkage to ntp deaemon in my case):

1. Determine how gpsd will be started by

```
cat /lib/systemd/system/gpsd.service
```

In my case there was a paragraph like this:

```
[Service]
Type=forking
EnvironmentFile=-/etc/default/gpsd
ExecStart=/usr/sbin/gpsd -n $GPSD_OPTIONS $OPTIONS $DEVICES
```

This told me the Environment file is /etc/default/gpsd

2. Edit the Environment file to add the options -s 4800 and -r (you can use nano or vi but will need to assume root powers)

```
sudo vi /etc/default/gpsd
```

Edit to achieve:

```
# GLG: added -s 4800 and -r to force 4800
# and use time even if no fix
GPSD_OPTIONS="-n -G -s 4800 -r"
```

Now verify that gpsd starts up and runs and provides information (e.g. to cgps and or gpsmon) after reboot.

MOVING TO NTP DAEMON

Once you have gpsd reliably installed and coming up properly on reboots, it is time to move to getting the ntp service running and reading data from gpsd. gpsd is supposed to have multiple techniques for providing outputs:

- socket based GPS output on 2947
- shared memory outputs on SHM0 and SHM1

In my case, the socket-based output connected fairly well after the above alterations in the calling of gpsd. Howevver, to get the shared memory outputs to work, the stock installation of ntp had to be altered to start it as root. This is necessary to gain control of the shared memory segments. ntp apparently may drop to a lower user status later, but without beginning as root, the shared memory segments cannot be utilized.

The software for ntp can be obtained as detailed above using apt-get

```
sudo apt-get install ntp
```

CONFIGURATION OF NTP: ntp.conf

Lines in /etc/ntp.conf:

Server statement to read the socket-based output of gpsd:

```
server 127.127.20.0 flag1 1 minpoll 4 maxpoll 4 refid GPS prefer
```

- The pseudo-IP address 127.127.20.0 refers to the socket-base communications.
- The settings for minpoll and maxpoll will force reading every 2^4 seconds = 16 seconds, and mark this input as GPS in a ntpq -p output. Minpoll is not supposed to be set to less than 4.
- flag1 1 sets a flag to accept input even if it radically differs from the system clock (which is likely in a non-real-time-clock raspberry pi.

Server connection in shared memory

```
server 127.127.28 0 minpoll 4 maxpoll 4 prefer
fudge 127.127.28.0 time1 0.000 refid SHM stratum 15
```

You can use both of these if you wish and pick which one you prefer with the `prefer` statement. If you are able to get the shared memory working, an advantage may be that you can still use `cgps` and `gpsmon` to observe the output of `gpsd`, without causing an EOF (end of file) stoppage of ntpd.

```
server 127.127.20.0 flag1 1 minpoll 4 maxpoll 4 refid GPS
server 127.127.28 0 flag1 1 minpoll 4 maxpoll 4 refid SHM0 prefer
```

The fudge statements refer to the use of a PPS (pulse per second) signal, which my dongle doesn't provide. You should be able to get well under 100 mSec without this.

II. USE

NOTE: If the raspberry pi ntp-server is operating without a real-time clock, its internal clock must be within approximately 4 hours (approx 14,400 seconds) in order for `gpsd/ntp` to lock to gps-derived accurate time. Without a real time clock, it may simply start after a power-off period, at the "time" it was last powered, which might well be more than 4 hours off. One way to correct the time is with the command

```
sudo date -s HHmm
```
e.g. `sudo date -s 1130`

This limitation disappears with the addition of a real-time clock to the raspberry pi. In my case I added an inexpensive DS3231-based clock that installs on 4 pins of the GPIO. There are numerous available explanations of the significant reconfiguration required to cause the raspberry to utilize the hardware clock instead of the "fake" hardware clock. After successfully integrating, the system will turn off after a power loss, and be locked in short order, starting from the real time clock time.

ntp is a well-designed system, updated over many years (with a more secured version also available) that has very significant advantages, including graded and carefully controlled updates of system clocks.

Simpler systems such as **Dimension 4**, (download: http://www.thinkman.com/dimension4/download.htm **)** use a more brute-force method for updating the clocks of computers, which can abruptly move the clock forward or backwards! This works fine most of the time, but can upset WSJT-X decoding if it occurs during a critical point.

Dimension 4 has the advantage that it is generally *extremely easy to install* and configure and works on most computers immediately. Most modern laptops will have real-time clocks as part of their hardware that are actually fairly accurate. FT8 decoding works pretty well even with timing errors in the hundreds of the milliseconds! So the typical recent laptop will not require a time update more than a few times a day. Because of the brute-force technique used by Dimension 4, for Field Day work I would suggest that computers be set up to synchronize with a working ntp server only every 30-60 minutes.

Large Time Error Synchronization Reporting
Dimension 4 has another odd characteristic. The synchronization times presented on the screen appear to indicate the clock errors that were corrected, and are often in the range of 200-500 milliseconds. In my testing, using more than one comparative technique, the actual error of the computer clock has been far less than that suggested by Dimension 4. Clock errors measured by alternative methods (such as ***https://time.is*** or **BktTimeSync**) have been generally in the range of 10%-25% of those apparently displayed by Dimension 4. In addition, I have seen time syncrhonizations by Dimension 4 that appeared to be errornously large, and improper. However, Dimension 4 is easy to install, works on almost all Windows computers, offers a large range of potential servers and doesn't make corrections

that are beyond the capability of WSJT-X to work with. My suggestions for home users would be to edit out time servers that are >500-1000 miles away, and use Dimension 4 sparingly as discussed above.

Meinberg NTP (https://www.meinbergglobal.com/english/sw/ntp.htm **)** is an alternative method of Windows time correction, but I've had problems getting its installation to work.

BktTimeSync by IZ2BKT (https://www.maniaradio.it/en/bkttimesync.html) is another alternative that offers the intriguing ability to directly connect to a GPS USB dongle or to use a NTP server, and my installation of it went smoothly and seemed to show relatively tiny corrections when utilized every minute on my Windows laptop:

```
Last Sync :Friday, June 09, 2023 09:41:00 --> Local clock offset was 0.003921 seconds
Last Sync :Friday, June 09, 2023 09:42:00 --> Local clock offset was -0.024262 seconds
Last Sync :Friday, June 09, 2023 09:43:00 --> Local clock offset was 0.002726 seconds
Last Sync :Friday, June 09, 2023 09:44:20 --> Local clock offset was 0.001159 seconds
Last Sync :Friday, June 09, 2023 09:45:00 --> Local clock offset was -0.048861 seconds
Last Sync :Friday, June 09, 2023 09:46:00 --> Local clock offset was 0.000858 seconds
Last Sync :Friday, June 09, 2023 09:47:00 --> Local clock offset was 0.132379 seconds
Last Sync :Friday, June 09, 2023 09:48:00 --> Local clock offset was -0.005601 seconds

Last Sync :Friday, June 09, 2023 09:49:00 --> Local clock offset was -0.002681 seconds
Last Sync :Friday, June 09, 2023 09:50:00 --> Local clock offset was -0.009764 seconds
Last Sync :Friday, June 09, 2023 09:51:00 --> Local clock offset was -0.003562 seconds
Last Sync :Friday, June 09, 2023 09:52:00 --> Local clock offset was 0.003060 seconds
Last Sync :Friday, June 09, 2023 09:53:00 --> Local clock offset was 0.003771 seconds
Last Sync :Friday, June 09, 2023 09:54:00 --> Local clock offset was 0.003717 seconds
Last Sync :Friday, June 09, 2023 09:55:00 --> Local clock offset was 0.015109 seconds
Last Sync :Friday, June 09, 2023 09:56:00 --> Local clock offset was -0.149080 seconds
Last Sync :Friday, June 09, 2023 09:57:00 --> Local clock offset was -0.008584 seconds
Last Sync :Friday, June 09, 2023 09:58:00 --> Local clock offset was 0.002330 seconds
Last Sync :Friday, June 09, 2023 09:59:00 --> Local clock offset was 0.031725 seconds
Last Sync :Friday, June 09, 2023 10:00:00 --> Local clock offset was 0.095348 seconds
Last Sync :Friday, June 09, 2023 10:01:00 --> Local clock offset was 0.001434 seconds
Last Sync :Friday, June 09, 2023 10:02:00 --> Local clock offset was 0.021001 seconds
Last Sync :Friday, June 09, 2023 10:03:00 --> Local clock offset was -0.007266 seconds
Last Sync :Friday, June 09, 2023 10:04:00 --> Local clock offset was 0.003996 seconds
Last Sync :Friday, June 09, 2023 10:05:00 --> Local clock offset was 0.003926 seconds
Last Sync :Friday, June 09, 2023 10:06:00 --> Local clock offset was 0.004539 seconds
Last Sync :Friday, June 09, 2023 10:39:20 --> Local clock offset was -0.244170 seconds
(Note this was after a 33 minute gap caused by my keeping their log open for inspection)
Last Sync :Friday, June 09, 2023 10:40:07 --> Local clock offset was 0.002891 seconds
Last Sync :Friday, June 09, 2023 10:41:00 --> Local clock offset was 0.001427 seconds
Last Sync :Friday, June 09, 2023 10:42:00 --> Local clock offset was 0.002897 seconds
Last Sync :Friday, June 09, 2023 10:43:00 --> Local clock offset was 0.004410 seconds
```

III. MONITORING PERFORMANCE

As discussed above, Dimension 4 operating on a computer being disciplined by the ntp server may show outsized estimates of time corrections.

Using https://time.is/ (if connected to the internet) offers a comparison technique against a completely separate atomic source. This will not reset the raspberry clock and provides a very simple way to observe `npt` working to bring the clock to the correct time.

Another comparison is to use the computer to observe FT8/FT4 signals using WSJT-X and watch the "DT" column, which appears to present the absolute value of time difference between your clock and received signals. When well-timed, these will tend to be near 0. It is normal to find a few outliers of a much as a second! Not everyone has an accurate clock.

Watching the ntpd daemon

It is safe to use `ntpq -p` to watch the ntp daemon. My understanding of the cryptic display is as follows:

Parameter	Meaning
Remote/refid	Remote source and assigned id If asterisked, being followed successfully
When	seconds since interrogated
Poll	Seconds between polling
Reach	octal shift buffer indicating successes in interrogating (see below)
Delay	
Offset	Appears to be in milliseconds
Jitter	Appears to be in milliseconds

REACH should remaind at 377 (8 bit rotary FIFO display) to indidate the ntpd daemon has had success on all of the last 8 interrogations of its time source (that is, gpsd). If this declines, it is possible that ntpd has observed an EOF (end of file) on the source from gpsd and will require stopping and restarting. Check `service ntp status` for the existence of the EOF.

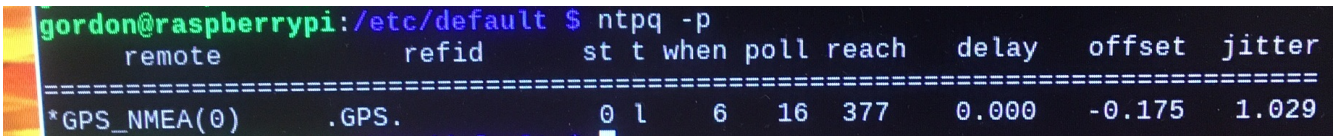

Figure. Illustration of `ntpq -p` output. Reach is 377 indicating all recent time queries to the source (GPS) have been successful. Jitter may be measured in milliseconds; <16 is said to be acceptable.

Avoid initiating `cgps` or `gpsmon` if you are locking to the socket-based `gpsd` output. I have seen this adversely affect the stream to ntpd, causing an EOF to be observed and requiring `ntp` to be restarted. It appears safe to use these if you are locking to the shared memory connection.

It is also safe to use `service ntp status` to observe `ntp`.

ntpq -c peer -c as -c rl

This command was my primary method for evaluating details of how the ntpd daemon was locking onto its source. In my case, I had deleted all internet-based sources purposefully. As you can see in the Figure below, my ntpd is locked (*) onto the SHM(0) Shared Memory Segment 0. It is polling every 64 seconds, has gottten a solid response all of the last 8 tries (reach = 0x377) and has negligible offset (-3.4 mSec) and jitter. The "reftime" is the time the internal clock was last updated by the source, which only occurs every poll=64 seconds in this case.

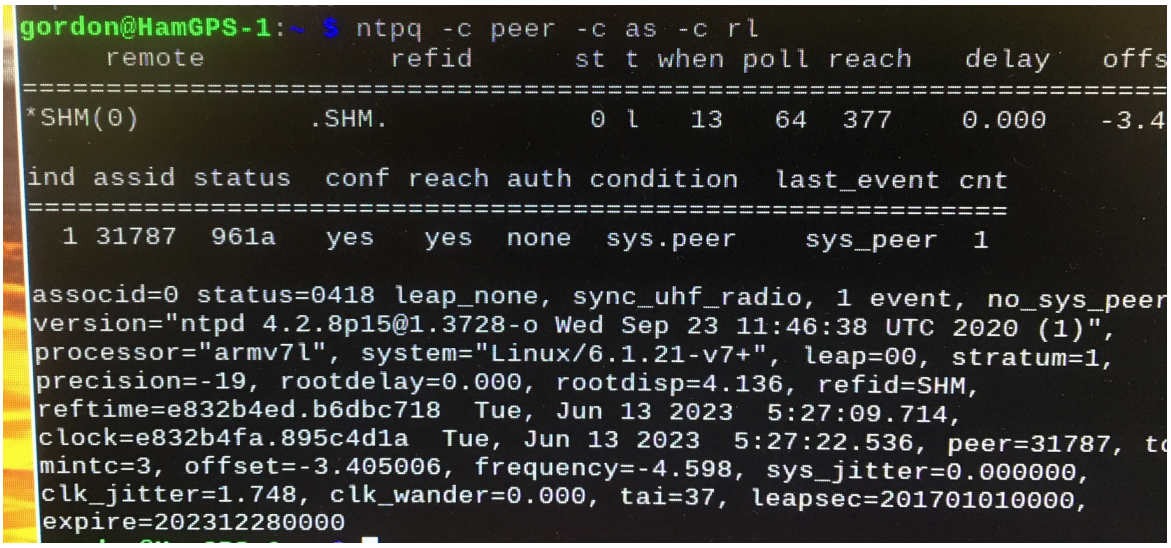

FIGURE: Display of shared memory segment lock.

ntpdate

`ntpdate` will likely require installation on your raspberry system. Use the "add software" options to find a late-model package. Once installed, using `-q` option (to prevent time-synching to the listed source) allows you to compare your current ntp-server clock to a different ntp server.

If you are connected to the Internet, this allows you to compare your gps-based time server, to a trusted external server. The Figure below shows a time offset from Wisconsin server in the range of +/- 70 milliseconds (probably just reflecting network delays).

You can also use this to study the difference between one gps-based ntp server, and another such system!

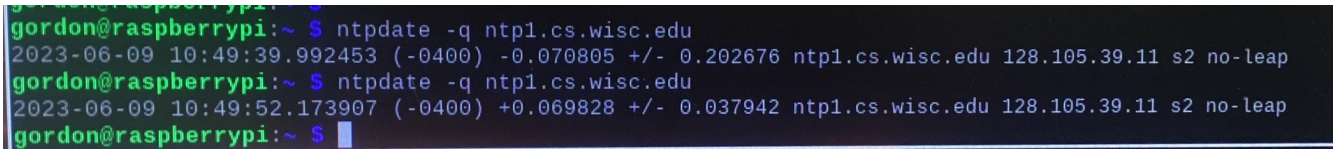

FIGURE. `ntpdate -q <other server>` comparison

IV. USEFUL RASPBERRY PI LINUX COMMANDS

Useful command	applicable services	example / comments
`sudo service <servicename> status`	gpsd ntp	sudo service gpsd status sudo service ntp status
`sudo service <name> start`	gpsd ntp	
`sudo service <name> restart`		
`gpsmon`	(reads directly from the serial port)	get out with CTRL-c
`cgps`	reads data from the gpsd system	Will also give you your grid square! get out with CTRL-c
`sudo systemctl enable <name>`	(sets service to run at boot time)	
`ntpq -p`	read status of peers	Example GPS_NMEA(0) . GPS..... gives information onfest jitter etc
`sudo ntpshmmon`		get out with CTRL-c (without sudo it will not be allowed to read the memory segments)
`ntpq -c peer -c as -c rl`	Note that is an alphabetical character "l" in the last option, not the number 1. This command will hep you see connections working (*) or not into ntpd, offset (in milliseconds), jitter (milliseconds) etc. Note the reftime is the time of last synchronization.	
`ntpdate -q <other server>`	The command ntpdate -q <address> will output information about time difference between your machine and the time source specified in <address>. The -q parametr is important, as without it ntpdate will try to sync time to the specified machine instead of just outputting the difference.	

REFERENCE DOCUMENTS

https://docs.ntpsec.org/latest/ntpd.html

Excellent information for gpsd:
https://www.multitech.net/developer/software/mlinux/using-mlinux/gpsd/#:~:text=GPSD configuration file is located,chip used on the device.

Useful reference for getting gpsd working:
https://gpsd.gitlab.io/gpsd/installation.html

Possibleproduct to get the 1PPS:
https://www.adafruit.com/product/2324