Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Diseño e implementación de un servidor TSA preciso para sistemas con pocos recursos

Autora: Dahna Colás Auberson

Director: Jorge Dávila Muro

MADRID, ENERO 2019

ÍNDICE

RESUMEN

El objetivo principal de este proyecto es implementar una Autoridad de Sellado del Tiempo (TSA) que junto con una fuente de tiempo precisa sea capaz de emitir sellados de tiempo tan precisos como para diferenciar dos documentos que se hayan sellado con sólo una diferencia de unos microsegundos. Además, el sistema donde esté implementado tendrá pocos recursos computacionales, por lo que habrá que tener especial cuidado con los recursos utilizados, teniendo que ser éstos lo más sencillos posibles, pero sin dejar de cumplir los estándares que hagan del sistema una TSA y fuente de tiempo fiables, siendo esta última servida mediante el protocolo NTP. Además, se van a estudiar los fundamentos básicos de los Sistemas Globales de Navegación por Satélite (GNSS) y de cómo funciona su comunicación con los dispositivos GPS. También se van a comprender los estándares usados. A lo largo del proyecto se van a explicar las configuraciones tenidas en cuenta, haciendo énfasis en el procesamiento de la señal PPS.

ABSTRACT

The main objective of this project is to implement a Time Stamping Authority (TSA) that working with an accurate time source be able to issue time stamps so precise as to differentiate two documents that have been timestamped with only a difference of a few microseconds. In addition, the system where it is implemented will have few computational resources, so it will be necessary to take special care with the resources used, having to be as simple as possible, but still comply with the standards that make the system a reliable TSA and time source, using NTP protocol to serve the time. Furthermore, the basic fundamentals of Global Navigation Satellite Systems (GNSS) and how their communication with GPS devices works will be studied, including the protocols used. Throughout the project the configurations taken into account will be explained, emphasizing the processing of the PPS signal.

1. INTRODUCCIÓN

Este trabajo consistirá en el desarrollo de un sistema capaz de garantizar que un documento no ha sido modificado desde el instante en que el sistema lo sella digitalmente. A este sistema se le llama Time Stamping Authority (TSA). Para la exactitud del tiempo, el sistema estará sincronizado directamente con una fuente primaria de tiempo (servidor stratum-0), siendo en este caso una constelación de 32 satélites. Para dichas funcionalidades, se deberá implementar una arquitectura PKI para llevar a cabo el papel de autoridad de sellado del tiempo y una arquitectura NTP para la sincronización con la fuente primaria de tiempo. Además, el sistema final deberá funcionar en un dispositivo con pocos recursos computacionales.

Objetivos:

1. Desarrollar un servidor con la arquitectura PKI que ejerza de TSA.
2. Implementar la arquitectura NTP para mantener el tiempo del servidor sincronizado con un servidor stratum-0 mediante un receptor GPS.
3. Hacer que ambas arquitecturas trabajen conjunta y correctamente y en un dispositivo con pocos recursos computacionales.

2. ESTADO DEL ARTE

2.1. Obtención del tiempo mediante un GPS

GPS (Global Positioning System) [1][2][3] es un sistema que proporciona servicios de posicionamiento, navegación y hora y está compuesto por tres partes:

- Satélites: son como las estrellas en las constelaciones, siempre se sabe la posición en la que están. Orbitan a 20.200 Km de altura con una trayectoria sincronizada para cubrir toda la superficie de la Tierra, con un período de 12 horas. En casi cualquier punto de la Tierra siempre habrá al menos 8 satélites visibles.

- Estaciones (en la Tierra): se aseguran de que los satélites estén en su sitio (los monitorizan) y mandan datos de corrección a los receptores.

- Receptores: conectan con los satélites y las estaciones.

Cada satélite GPS dispone de cuatro relojes atómicos, dos de Cesio y dos de Rubidio, con una precisión de 1 segundo en 300 millones de años, que están sincronizado con una estación en tierra, para que así todos los satélites tengan la misma hora. Esta precisión la consiguen operando internamente a 9,192,631,770 Hz. Esta hora no está corregida para que coincida con la rotación de la Tierra (el tiempo UTC); no se corrigen ni los segundos intercalares ni ninguna otra pequeña variación que se requiera periódicamente por la variación en el giro de la Tierra. Por eso los mensajes de los GPS incluyen un desplazamiento para poder calcular la hora UTC correctamente. [5]

Cada satélite GPS emite continuamente, a intervalos regulares, un mensaje de navegación a 50 bits por segundo, en la frecuencia transportadora de microondas de aproximadamente 1.600 MHz. La radio FM, en comparación, se emite a entre 87,5 y 108,0 MHz y las redes Wi-Fi funcionan a alrededor de 5000 MHz y 2400 MHz. Más concretamente, todos los satélites emiten a 1575,42 MHz (esta es la señal L1) y 1227,6 MHz (la señal L2). Actualmente hay más niveles a parte de L1 y L2, pero el módulo GPS que se va a usar sólo admite esas dos frecuencias.

Cada transmisión dura 30 segundos y lleva 1500 bits de datos codificados. Esta pequeña cantidad de datos está codificada con una secuencia pseudoaleatoria (PRN) de alta velocidad que es diferente para cada satélite. Los receptores GPS conocen los códigos PRN de cada satélite y por ello no sólo pueden decodificar la señal, sino que la pueden distinguir entre diferentes satélites.

Las transmisiones son cronometradas para empezar de forma precisa en el minuto y en el medio minuto tal como indique el reloj atómico del satélite. La primera parte de la señal GPS indica al receptor la relación entre el reloj del satélite y la hora GPS. La siguiente serie de datos proporciona al receptor información de órbita precisa del satélite.

El mensaje de transmitido por el satélite contiene información sobre su estado, de manera que puede deducirse si es defectuoso o no, las órbitas de los demás satélites de su constelación (almanaque), su posición (efemérides) y la hora de acuerdo con el reloj atómico primario que tenga a bordo[4], que es interceptada por el receptor GPS, el cual hace sus cálculos en base a la información recibida, al tiempo que ha tardado el mensaje en llegar y a los datos de corrección recibidos por la estación en tierra, para así conseguir su posición u hora.

El protocolo NTP no es usado por los satélites para enviar la información de su hora, es el receptor GPS o el sistema conectado al GPS el que tiene que hacer la conversión para adecuarla al protocolo. La hora del satélite GPS está basada en la misma escala de tiempo que se usa en el protocolo NTP (UTC, Universal Coordinated Time), por lo que no hay problemas para hacer la conversión.

Los relojes atómicos son muy precisos pero muy caros, por eso la mayoría de los relojes son de cuarzo, que son mucho menos precisos, por lo que necesitan sincronizar su hora periódicamente con relojes precisos para conseguir la precisión deseada.

Las señales PPS se usan para la medición precisa del tiempo. Algunos receptores GPS (como el usado en este proyecto) tienen un pin reservado para este propósito. Las salidas PPS tienen una estabilidad mucho mayor que puede ser de exacta mejor que 1 ns. La señal no especifica el tiempo, sino que especifica exactamente cuándo empieza el segundo, lo que hace que un sistema pueda ser una fuente de tiempo stratum-1. [6]

2.2. Causas de los errores en medición

[7] La troposfera e ionosfera pueden cambiar la velocidad de propagación de la señal del satélite si las condiciones atmosféricas no son buenas. La solución está en que el dispositivo GPS use 2 frecuencias distintas para minimizar el error en la velocidad. Medido en distancia, el error puede llegar a ser de 5 metros. (Un error de 1 milisegundo puede resultar en un error de 300 kilómetros. [8])

El efecto multitrayectoria ocurre cuando la señal del satélite rebota en edificios y montañas cercanas, haciendo que el dispositivo GPS detecte la señal 2 veces. Esto sólo causa errores de 1 metro.

GDOP (Geometric Dilution of Precision) o PDOP (Position Dilution of Precision) es el error causado por la posición de los satélites desde el punto de vista del receptor. Cuanto más "esparcidos" estén los satélites, más precisa será la información recibida, habrá mejor GDOP.

2.3. Problema de la relatividad

[9][10] Hay que tener en cuenta que el tiempo no corre en la Tierra igual que en los satélites en órbita. Según la Relatividad General y Especial, hay diferencias entre un reloj atómico en un satélite y uno idéntico en la Tierra:

La Relatividad general enuncia que al haber en la Tierra una atracción gravitacional más fuerte, un reloj allí parecerá que corre más lento que uno a bordo de un satélite. Esto hace que los satélites vayan adelantados 45 microsegundos al día. Sin embargo, la Relatividad Especial dice que un reloj en un satélite corre más lento respecto a un observador en la Tierra debido a que éste ve los satélites en movimiento respecto a él. Esto hace que los relojes en los satélites vayan 7 microsegundos retrasados al día.

La combinación de ambas teorías se reduce a que los relojes en los satélites van 38 microsegundos al día adelantados. Para compensar esta diferencia, los relojes de los satélites están diseñados para "hacer tic tac" con una frecuencia más baja. También los receptores GPS harán cualquier cálculo de tiempo relativista requerido por el satélite mediante los datos que transmita éste.

2.4. Timestamping

El sellado del tiempo es el proceso por el cual se sella un documento digitalmente para demostrar que éste ha existido en un determinado instante del tiempo. Esto se lleva a cabo a través de una autoridad de sellado de tiempo que sigue el estándar ANSI ASC X9.95. [11] Un servidor TSA actúa como una tercera parte de confianza para asegurar la existencia en un determinado instante de tiempo de documentos importantes que requieran de esta seguridad como lo son las transacciones, facturas electrónicas, documentos del Gobierno, etc.

3. TRABAJOS PREVIOS

Hay bastantes proyectos encontrados en la web que implementan un servidor NTP en una Raspberry Pi, pero pocos hay que reciban la hora de un stratum-0 y además con una precisión del orden del microsegundo.

Los proyectos en los que se ha conseguido tal precisión [12][13] han usado la señal hardware PPS (Pulse Per Second) del GPS (necesitando tener un kernel con el modo PPS activado) y han hecho las modificaciones oportunas en el archivo ntp.conf (en un sistema Linux con el paquete ntp instalado [14]). Para comprobar dicha precisión han usado la herramienta ntpq.

En el proyecto de satsignal, los recursos usados usando un módulo GPS Adafruit MTK3339, obteniendo una precisión de 20 microsegundos, han sido: 160 de 512Mb de memoria RAM y un 3% de uso de la CPU [15]. Además, también ha tenido en cuenta las condiciones ambientales (orientación, temperatura, clima…)

En cuanto a la arquitectura TSA, al ser sólo software no tiene tantas complicaciones. Principalmente habrá que seguir los estándares establecidos para implementarla. OpenSSL proporciona una librería, *openssl ts* [16][17], que puede ser usada tanto en el lado del cliente como del servidor y que sigue el Protocolo de Sellado del Tiempo (RFC 3161).

4. ESTÁNDARES

4.1. Network Time Protocol Version 4

Network Time Protocol (NTP) [18] es un protocolo usado para sincronizar relojes de sistemas informáticos a través de Internet sobre UDP. La versión 4 ha incluido mejoras en los algoritmos que aumentan la precisión a las decenas de microsegundos en condiciones idóneas y correcciones de diseño e implementación de la previa versión del protocolo.

Las cabeceras de un paquete NTP son palabras de 32 bits que se pueden subdividir en campos. El paquete contiene tres componentes: la cabecera, uno o más campos de extensión opcionales y opcionalmente un código de autenticación de mensaje (MAC).

A continuación se procede a describir los campos de la cabecera de un paquete NTP.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="2">LI</td><td colspan="3">VN</td><td colspan="3">Mode</td><td colspan="8">Stratum</td><td colspan="8">Poll</td><td colspan="8">Precision</td></tr>
<tr><td colspan="32">Root Delay</td></tr>
<tr><td colspan="32">Root Dispersion</td></tr>
<tr><td colspan="32">Reference Identifier</td></tr>
<tr><td colspan="32">Reference Timestamp (64 bits)</td></tr>
<tr><td colspan="32">Origin Timestamp (64 bits)</td></tr>
<tr><td colspan="32">Receive Timestamp (64 bits)</td></tr>
<tr><td colspan="32">Transmit Timestamp (64 bits)</td></tr>
<tr><td colspan="32">Extention Field 1 (variable)</td></tr>
<tr><td colspan="32">Extention Field 2 (variable)</td></tr>
<tr><td colspan="32">Key Identifier</td></tr>
<tr><td colspan="32">dgst (128 bits)</td></tr>
</table>

Descripción de las variables del paquete NTP:

- Leap Indicator (LI): entero sin signo de dos bits que indica que indica si hay que modificar el último minuto del último día del presente mes. Los valores posibles son:

Valor	Significado
0	sin modificación
1	el último minuto tiene 61 segundos
2	el último minuto tiene 59 segundos
3	reloj desincronizado

- Version (VN): entero sin signo de 3 bits indicando la versión del NTP.
- Mode: entero sin signo de 3 bits indicando el modo de la siguiente manera:

Valor	Significado
0	reservado
1	simétrico activo
2	simétrico pasivo
3	cliente
4	servidor
5	broadcast
6	reservado para mensajes de control de NTP
7	reservado para uso privado

Cabe decir que los modos 6 y 7 suelen estar desactivados en casi todos los servidores, ya que éstos son vulnerables a ciertos ataques [19][20]. En los parches actuales está solventado.

- Stratum: entero sin signo de 8 bits representando el nivel del servidor local, los valores definidos son:

Stratum	Significado
0	no especificado o inválido
1	servidor primario (equipado con receptor GPS)

2-15	servidor secundario (mediante NTP)
16	no sincronizado
17-255	reservado

- Poll: entero con signo de 8 bits que indica el intervalo máximo de tiempo en segundos entre dos mensajes sucesivos, expresado en log2. Es decir, si poll=4 se está refiriendo a 2^4=16 segundos.
- Precision: entero con signo representando la precisión del reloj en segundos, expresado en log2.
- Root Delay: longitud de 32 bits. Retraso total de ida y vuelta al reloj de referencia.
- Root Dispersion: longitud de 32 bits. Es la mejor estimación del error actual del reloj del sistema respecto del reloj de referencia, basándose en el comportamiento que ha tenido hasta ahora.
- Reference Identifier: código de 32 bits identificando el servidor o reloj de referencia y la interpretación depende del nivel del servidor.
- Reference Timestamp: longitud de 64 bits. Hora en la que el reloj del sistema se configuró o corrigió por última vez, en formato de tiempo NTP.
- Origin Timestamp: longitud de 64 bits. Hora del cliente a la cual la petición salió hacia el servidor, en formato de tiempo NTP.
- Receive Timestamp: longitud de 64 bits. Hora del servidor a la cual la petición llegó del cliente, en formato de tiempo NTP.
- Transmit Timestamp: longitud de 64 bits. Hora del servidor a la cual la respuesta salió hacia el cliente, en formato de tiempo NTP.
- Destination Timestamp: hora del cliente a la cual la respuesta llegó del servidor, en formato de tiempo NTP. Este campo no está incluido en la cabecera, se determina en la llegada del paquete y está disponible en la estructura de datos del paquete en el búfer.
- Extension Field n: los campos de extensión contienen un mensaje de petición o uno de respuesta.

- Key Identifier: entero sin signo de 32 bits usado por el cliente y el servidor para designar una clave de MD5 de 128 bits.
- Message Digest: hash MD5 de 128 bits calculado con la clave.

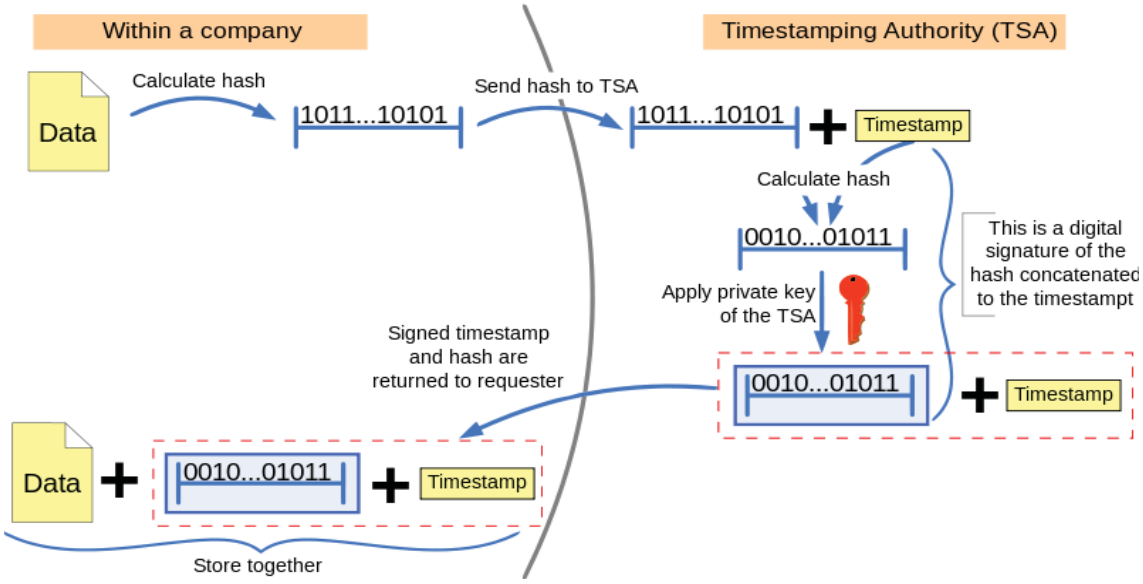
4.2. Time-Stamp Protocol

Time-Stamp Protocol (TSP) [21][22]es un protocolo criptográfico para certificar que un documento no ha sido cambiado desde su creación o modificación una vez que se ha sellado con una marca de tiempo usando un certificado X.509 y una infraestructura de clave pública. Esta marca de tiempo es una prueba de existencia de un documento en un determinado instante de tiempo. El servicio de sellado de tiempo opera como una tercera parte de confianza.

Para crear un sellado de tiempo, la entidad solicitante calcula un hash de la información original, este hash se envía a la TSA en un mensaje de petición de token de sellado de tiempo. La TSA concatena el hash con la marca de tiempo y calcula un nuevo hash, el cual es firmado digitalmente con la clave privada del TSA y es enviado junto a la marca de tiempo a la entidad solicitante. Una vez recibida la respuesta, el solicitante debe verificar si el sellado ha fallado. Si no es así, debe de verificar con la clave pública de la TSA que la respuesta está debidamente sellada. Entre los campos a verificar se encuentra el campo del tiempo, que se debe de validar con una referencia de tiempo local de confianza si se dispone de una, y el valor del campo *nonce*, un número aleatorio largo generado por el cliente que se incluye en la petición, para prevenir ataques de seguridad.

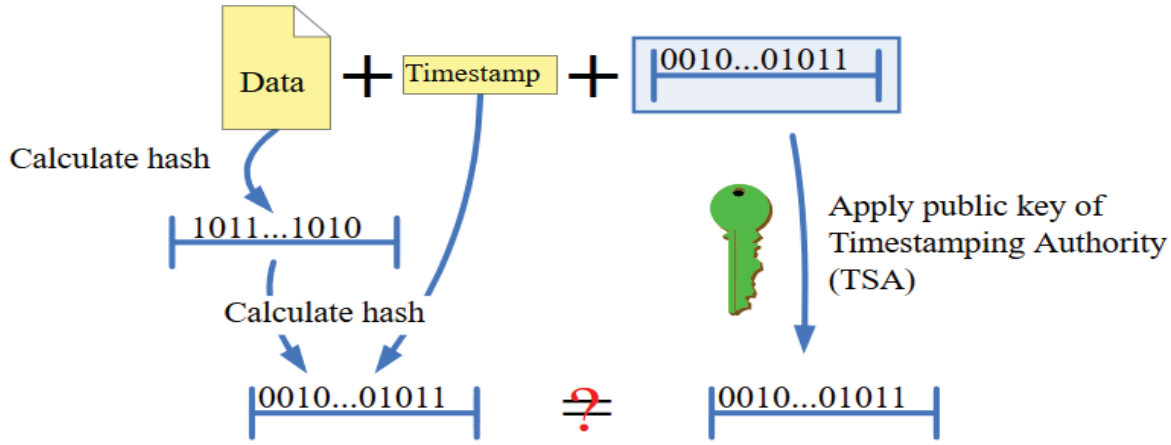
Cuando se haya comprobado que la respuesta es válida, el solicitante almacena el hash de respuesta y la marca de tiempo junto a la información original.

La siguiente imagen ilustra el proceso de sellado por parte de la TSA.



Para verificar el sello de tiempo hace falta validar la firma digital del TSA. Para ello se necesita la petición o documento original, la respuesta sellada y la clave pública de la TSA.

La siguiente imagen ilustra el proceso de verificación del sello de la TSA.



4.3. Pulse-Per-Seconds API Version 1

Este estándar define una API [23] para el uso de la utilidad PPS en sistemas basados en UNIX en el que comenta que la forma conveniente de proporcionar la señal PPS a un sistema es conectar dicha señal a un pin de control de módem en una interfaz de línea serie a la computadora. La salida de código de tiempo de la fuente de tiempo es transmitida a la computadora sobre la misma línea serie. La computadora detecta la

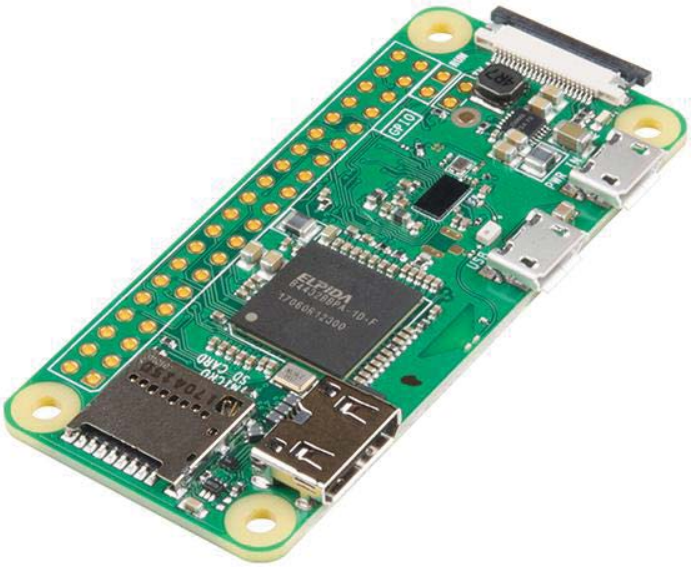
señal en el pin recibiendo una interrupción y registra la marca de tiempo cuando atienda la interrupción.

Una de las funciones más importantes que proporciona esta API es time_pps_kcbind(), la cual asocia un kernel consumer a la interfaz PPS instanciada que haya. Esta función devolverá un error si no se ha activado el PPS kernel consumer en la configuración del kernel.

5. HARDWARE USADO

El hardware usado es una Raspberry Pi Zero W [24] que hace de servidor TSA y NTP de bajos recursos computacionales y un módulo GPS que se utiliza para para obtener la hora precisa.

De las características más relevantes para este proyecto de la Raspberry es que cuenta con WiFi, CPU de 1GHz con un solo procesador lógico de un núcleo, 512MB de RAM y el procesador BCM2835 ARM Peripherals [25]. Adicionalmente tiene una memoria MicroSD de 16GB que almacena el sistema operativo junto al código necesario para poner en funcionamiento el servidor.



El chip BCM2835 ARM Peripherals cuenta con 2 puertos serie principales (UART0) y un pin preparado para señales precisas, como lo es la señal PPS. Estos son los pins GPIO usados [26]:

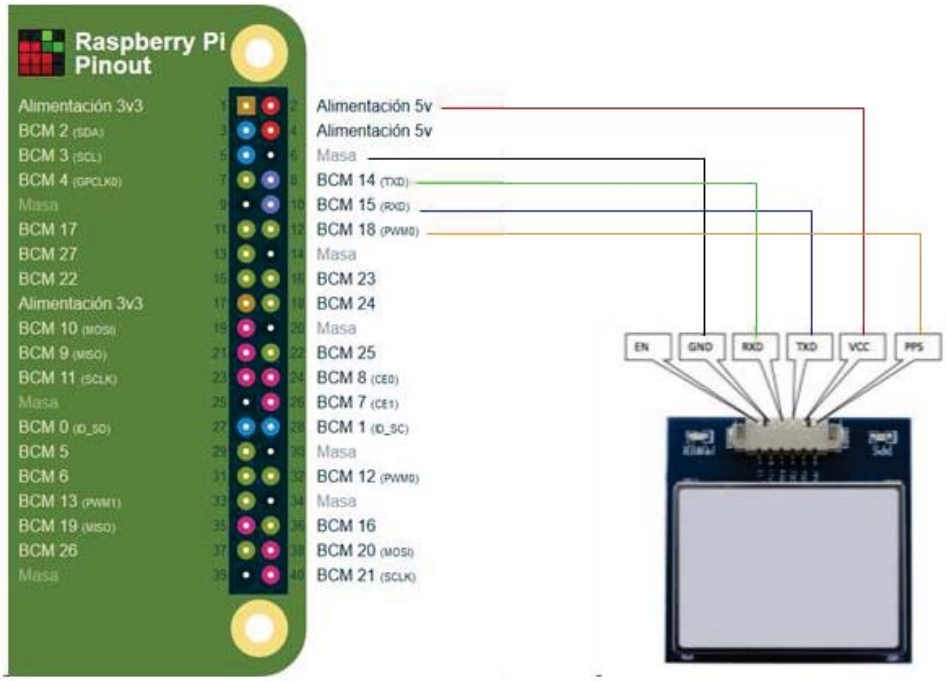
- Pin físico 2: pin de alimentación de 5V y de hasta 1.5A.
- Pin físico 6: pin de masa.
- Pin físico 8: pin de transmisión UART (cable verde del receptor).
- Pin físico 10: pin de recepción UART (cable azul del receptor).
- Pin físico 12: es el pin PWM (modulación por ancho de pulsos) que se utiliza para las señales precisas.

El módulo GPS es un FZ0041 G28U7FTTL comprado en DIYmall. Sus especificaciones son facilitadas en la página de Amazon que también vende el módulo [27]. Ahí se especifica que tiene un chip U-BLOX UBX-G7020-KT [28] [29], un receptor cuyo objetivo es procesar señales transmitidas por los satélites con alto rendimiento y bajo consumo energético gracias a su arquitectura de molde único y algoritmos mejorados. El chip soporta los GNSS GPS, GLONASS, QZSS y SBAS.

El rango de suministro de voltaje es entre 1.4 V a 3.6 V y el soporte de E/S de 1.8 V y 3.0 V hace el chip compatible con una amplia variedad de aplicaciones.

5.1. Conexión de los componentes hardware

La conexión entre la Raspberry y el módulo GPS consta de 5 cables y se muestra en el siguiente diagrama:



El pin 2 de 5V de la Raspberry se conecta al pin VCC del GPS.

El pin 6 de masa de la Raspberry se conecta al pin GND del GPS.

El pin 8 TXD de la Raspberry se conecta al pin RXD de recepción del GPS.

El pin 10 RXD de la Rapberry se conecta al pin TXD de transmisión del GPS.

El pin 12, que se utiliza para conseguir tiempos muy precisos se conecta al pin PPS del GPS para así obtener la señal PPS con alta precisión.

6. CONFIGURACIÓN SOFTWARE

El sistema operativo instalado en la raspberry es Raspbian Stretch Lite (2018-11-13) [30]

6.1. KERNEL

Antes de instalar el kernel que viene por defecto, se va a descargar su código fuente y se van a configurar ciertas características que permitan la correcta sincronización de la hora. Se va a hacer cross-compiling; es decir, compilar el kernel en una plataforma distinta al sistema destino, por razones de optimización del tiempo de compilación. Las instrucciones que se han seguido son las oficiales de la página web de raspberry [31] [32].

Una vez descargado el kernel y las herramientas necesarias para el cross-compiling, se aplica la configuración básica:

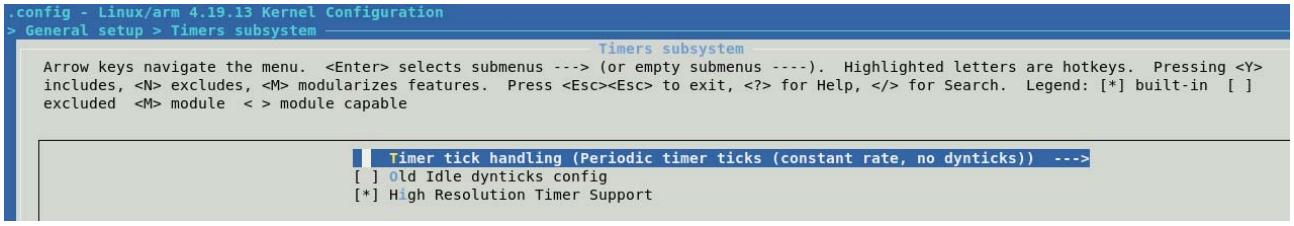
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig

Después se abre el menú de configuración:

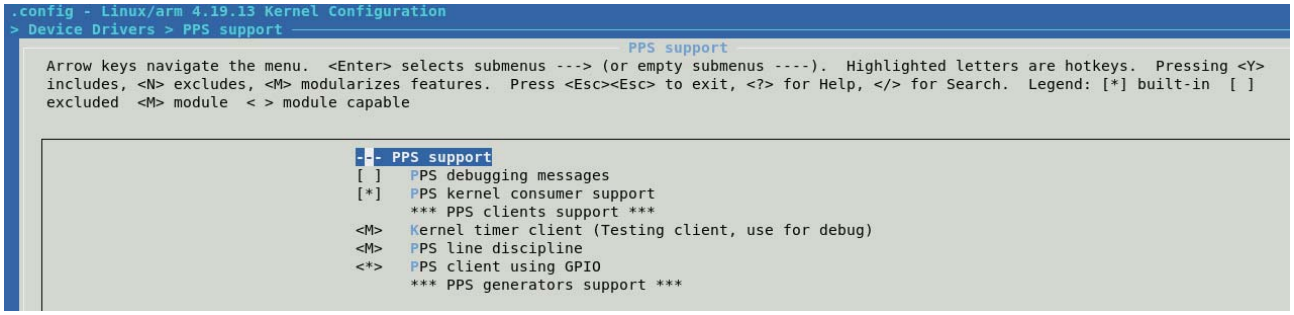
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig

Va a ser modificado lo siguiente:

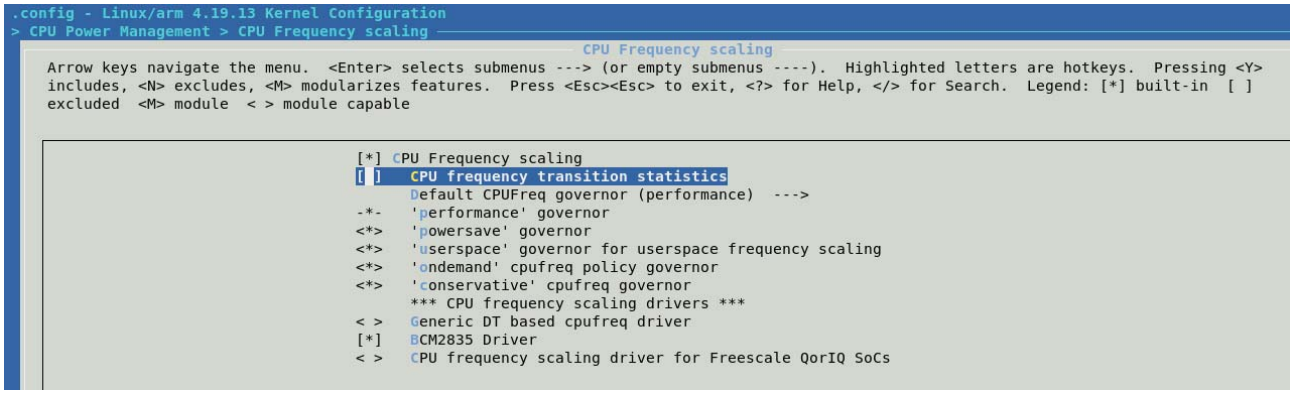
- En Timers subsystem se va a desactivar todo lo que haga referencia a dyntics y se van a activar los ticks periódicos y el soporte para relojes de alta precisión.



- En Device drivers → PPS support activar PPS kernel consumer support y PPS client using GPIO.



- En CPU Power Management → CPU Frequency scaling establecer performance como el gobernador por defecto, así la raspberry siempre se ejecutará a su máxima frecuencia (1 000 MHz), disminuyendo los errores de sincronización del reloj debido a los cambios de frecuencia.



Compilar e introducir en la raspberry siguiendo las instrucciones oficiales.

Antes de iniciar la raspberry hay que configurarla para deje libres sus puertos UART principales. Esto es debido a que este modelo incorpora bluetooth, el cual usa los puertos UART. Para desactivar el bluetooth y configurar el módulo pps-gpio se va a crear la siguiente configuración en *boot*/config.txt:

```
# Mirar los logs de DT con: vcdbg log msg
dtdebug=on

# Para que corra con el kernel que se ha recompilado
kernel=kernel_pps.img

# Deshabilitar BT y dejar los UART PL011 para el GPS
dtoverlay=pi3-disable-bt

#Activar pps para el GPIO 18
dtoverlay=pps-gpio,gpiopin=18
```

También hay que desactivar la consola en los puertos serie en *boot*/cmdline.txt quitando console=serial0,115200, que también puede aparecer como console=ttyAMA0,115200.

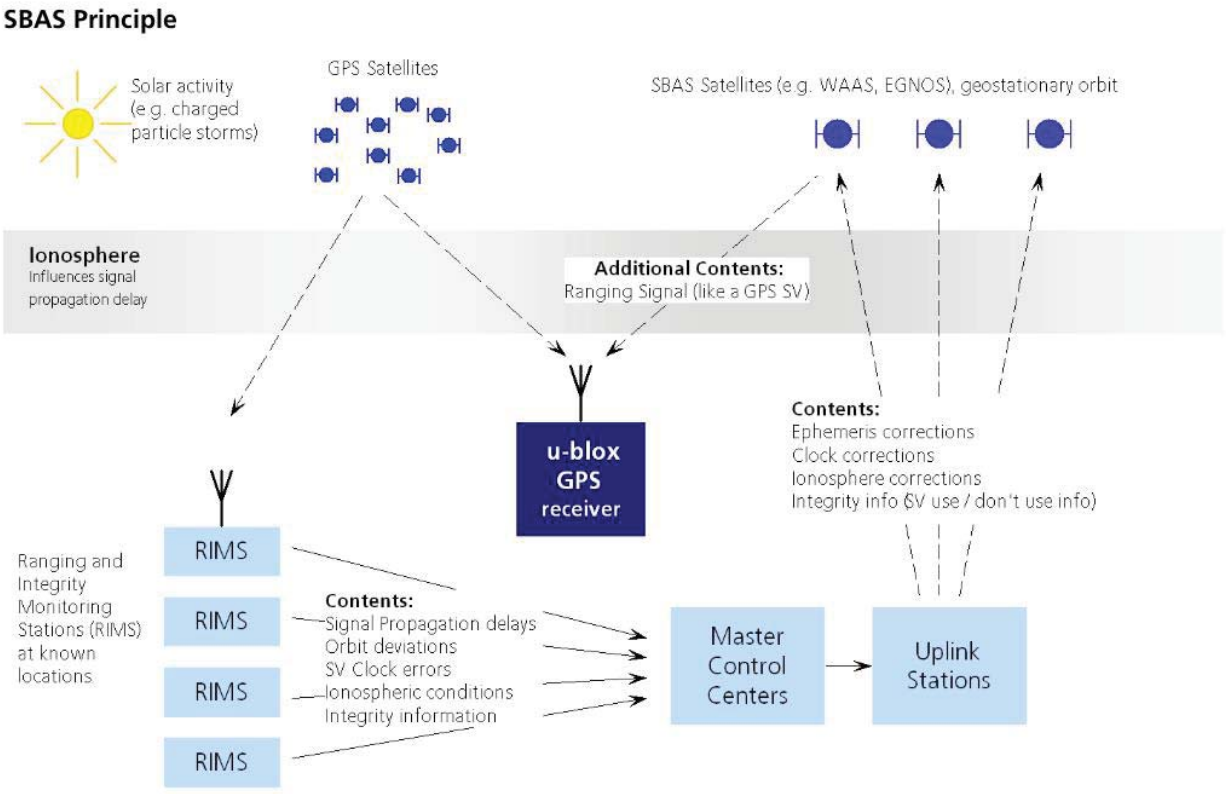
El servicio alternativo al NTP que usa el sistema es systemd-timesyncd, por lo que habrá que deshabilitarlo para que no entre en conflicto con NTP: systemctl disable systemd-timesyncd.service.

6.2. U-BLOX

El receptor u-blox [33] puede hacer un seguimiento de varios sistemas de navegación global por satélite (GNSS). Estos son los GNSS que detecta:

GPS (Global Positioning System): es el GNSS más expandido y el que usa el receptor por defecto. A fecha de octubre de 2018 tenía 31 satélites operativos. [34]

SBAS (Satellite-based augmentation systems): como dice su nombre, hace referencia a los GNSS de aumento. Estos sistemas usan satélites adicionales para aumentar la precisión, fiabilidad y disponibilidad del GNSS GPS (actualmente no hay SBAS para otros GNSS), proporcionando información extra al proceso de cálculo. Estos sistemas no conforman un sistema de navegación individual global, ya que no disponen de suficientes satélites como para poder estar disponibles mundialmente. Para ello se necesitan al menos 24 satétiles. En el caso del SBAS de Europa, EGNOS [35], tiene sólo dos satélites en órbita. La siguiente imagen resume el funcionamiento de un SBAS:



GLONASS: es el sistema de navegación de Rusia. Es una alternativa al uso de GPS. La precisión si se usan sólo satélites GLONASS es peor que si se usan sólo satélites GPS. Esto es debido a que hay menos satélites GLONASS (24 actualmente [36]) que satélites GPS (32 actualmente [37]), además de que su velocidad de transmisión es menor. También cabe decir que las señales de GLONASS se transmiten en una frecuencia distinta (1602.00MHz) a la de todas las demás (1575.42MHz).

El protocolo NMEA identifica los satélites con un número de dos dígitos, reservando de 1 a 32 para GPS, de 33 a 64 para SBAS y de 65 a 96 para GLONASS.

Es posible comunicarse con el receptor GPS mediante sentencias NMEA. Éstas son del tipo $XXYYY,arg0,arg1,...,argN*checksum<CR><LF>:

- XX es el identificador de con quién se va a comunicar. En el receptor usado hay dos valores posibles:
    - GP: para hablar con el receptor GPS
    - P: para hablar con el propietario, que en este caso es u-blox. Se usa para enviar los mensajes no estándar.

- YYY es el identificador del mensaje, el cual permite saber qué tipo de mensaje va a tener la sentencia.

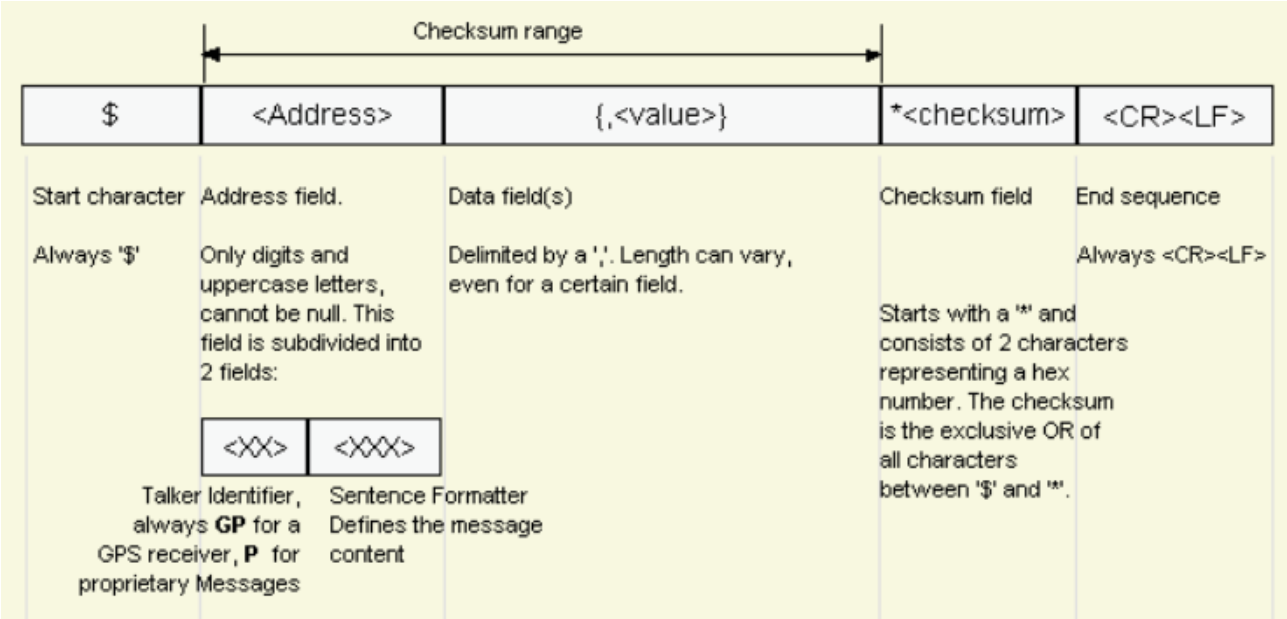
  En el caso de que XX sea GP, YYY puede tener estos valores:

  - GLL, RMC: para los mensajes que sean relativos al estado del receptor.
  - GGA: mensajes relativos a la calidad de recepción.
  - GSA: mensajes relativos al modo de navegación.
  - GLL, RMC, VTG, GNS: mensajes relativos a la posición.
  - GSV: satélites a la vista

  En el caso de que XX sea P, YYY tendrá el valor de UBX. En este caso arg0 será un número de dos dígitos que identificará el tipo de mensaje PUBX.

  argN: contienen los argumentos del mensaje.

  checksum: consiste en dos caracteres hexadecimales resultantes de hacer un XOR de todos los caracteres comprendidos entre $ y *.



Ejemplo:

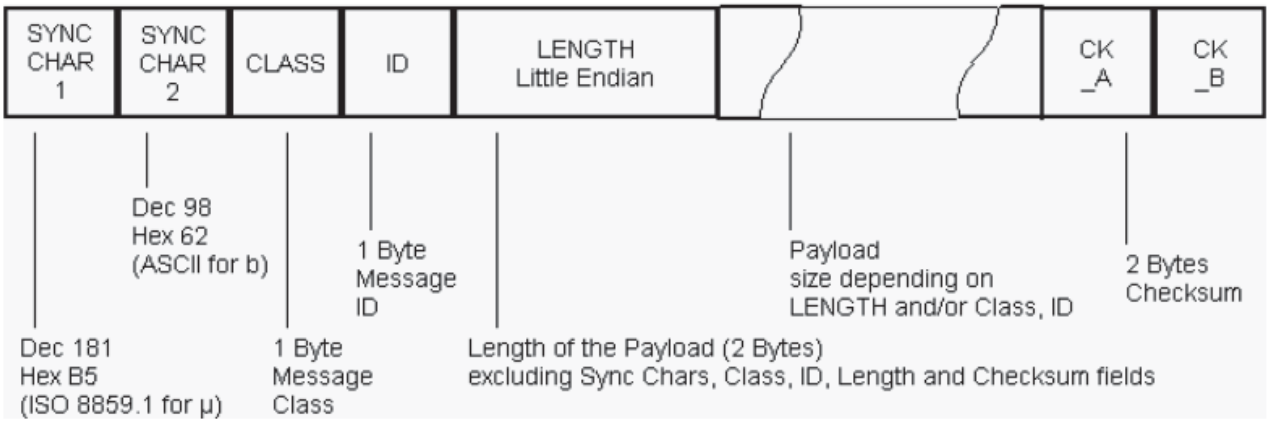
Una sentencia válida para enviar es $PUBX,00*33. 33 es el checksum realizado a "PUBX,00", el cual se puede calcular con este sencillo programa en python al cual se le pasa como parámetro "PUBX,00":

```
import sys

sentence = sys.argv[1]
checksum = 0
for e in sentence:
   checksum ^= ord(e)

print hex(checksum)
```

La salida por pantalla será 0x33.

Para comunicarse con el receptor, estos mensajes se pueden enviar directamente por el puerto al que esté conectado. Para comprobar que el dispositivo recibe correctamente los mensajes que se le envía se puede hacer con una simple prueba: por un lado se escucha al puerto mediante el comando de terminal "cat /dev/ttyAMA0 | grep UBX" y en otra terminal se ejecuta "echo -en "\$PUBX,00*33\r\n" > /dev/ttyAMA0". Al poco de enviar el mensaje el receptor contestará con un mensaje PUBX,00 de respuesta:
$PUBX,00,180343.00,4034.53127,N,00400.19011,W,951.890,G3,12,13,0.931,230.90,0.112,,2.06,2.50,2.12,**6**,0,0*7A. Éste contiene información acerca de la posición. El campo resaltado indica el número de satélites usados activamente (puede llevar un seguimiento de más satélites pero no usar sus datos).

Otra forma de comunicación es el protocoolo del propio fabricante, UBX, el cual consiste en mensajes en binario en los que cada parámetro ocupa un byte. La tecnología de posicionamiento u-blox es totalmente configurable usando este protocolo. Así es su estructura:



Los dos primeros bytes son los identificadores del protocolo: 0xb5 0x62, y no se tienen en cuenta para la suma de comprobación. Para calcular los dos bytes del checksum se ha usado este código en python:

```
import sys

# packet (without header 0xB5,0x62)
packet = sys.argv[1][1:-1].split(',')

print packet

CK_A,CK_B = 0, 0
for i in packet:
  CK_A = CK_A + int(i.strip(), 16)
  CK_B = CK_B + CK_A

CK_A = CK_A & 0xFF
CK_B = CK_B & 0xFF

print "UBX packet checksum:", ("0x%02X,0x%02X" % (CK_A,CK_B))
```

Un ejemplo de input es "[06,16,01,06,00,00,00,01,00,08]".

Los mensajes de este protocolo se envían al receptor de la forma:

```
echo -en "\xb5\x62\x06\x16\x01\x06\x00\x00\x00\x01\x00\x08\x2c\x3f"
> /dev/ttyAMA0
```

U-blox proporciona un software para facilitar la comunicación, pero sólo corre en Windows., además de que no está preparado para la arquitectura ARM de la raspberry.

Visto cómo es posible counicarse con el receptor, se va a configurar para usar EGNOS (European Geostationary Navigation Overlay Service) como sistema de aumento (SBASS) que va a complementar al sistema de navegación GPS. Los tipos de corrección que va usar SBAS son los siguientes:

- Corrección de perturbaciones a corto plazo en la señal GPS (debido a problemas de su reloj, etc.)
- Correcciones relacionadas con la actividad de la Ionosfera
- Correciones a largo plazo para distintos problemas

A la fecha de publicación de las especificaciones de u-blox (marzo de 2012), SBAS puede hacer seguimiento de los satélites Inmarsat 3F2 AOR-E, Artemis e Inmarsat 3F5 IOR-W de EGNOS, cuyos PRN son 120, 124 y 126, respectivamente. Actualmente ya no es así, EGNOS tiene 3 satélites en órbita, 2 de ellos operativos y el restante en modo test, y sus PRN son 123, 136 y 120, respectivamente. [38].

Para configurar el receptor para que también haga seguimiento del servicio EGNOS se va a usar el protocolo UBX propio del fabricante para aplicar la configuración. El mensaje va a ser el siguiente:

```
echo -en "\xb5\x62\x06\x16\x01\x06\x00\x00\x00\x01\x00\x08\x2c\x3f" > /dev/ttyAMA0
```

- Con el caracter de escape \x se indica que se envía 1 byte con valor hexadecimal
- b5 62 es la cabecera que identifica un mensaje UBX
- 06 16 es el identificador del mensaje de configuración SBAS (CFG-SBAS)
- 01 habilita SBAS
- 06 indica que se realicen correcciones diferenciales y que se use información de integridad
- 00 está obsoleto

- 00 00 01 00 08 indica qué PRN buscar. Aquí se le está indicando que busque los PRN 123 y 136

- 2c 3f checksum

El receptor está configurado por defecto para aceptar tanto mensajes NMEA como UBX, pero si se duda, con esta sentencia se configura una velocidad de transmisión de 9600 baudios y permite tanto la entrada como salida de los protocolos UBX y NMEA:

```
echo -en "\$PUBX,41,1,0003,0003,9600,0*14\r\n" > /dev/ttyAMA0
```

7. SERVIDOR

7.1. TSA

Como se puede observar con el comando lscpu, sólo hay una CPU con un core que sólo soporta un thread, así que implementar multithreading en la raspberry no es lo recomendado (aun así se ha intentado, pero ha resultado en un servivdor totalmente inestable, incluso con sólo dos threads), se ha implementado un servidor HTTP monoproceso.

```
root@raspberrypi:~# lscpu
Architecture:          armv6l
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
Model:                 7
Model name:            ARMv6-compatible processor rev 7 (v6l)
CPU max MHz:           1000.0000
CPU min MHz:           700.0000
BogoMIPS:              997.08
Flags:                 half thumb fastmult vfp edsp java tls
root@raspberrypi:~#
```

Se ha usado el mínimo número de recursos. Todos los usados en el programa son librerías del sistema más el módulo de timestamping de la librería criptográfica openssl, el cual cumple con el estándar RFC3161 [39] [40].

El programa consiste en tres partes:

- tsa_server.c aquí se encuentra el bucle principal de escucha y se inicia el contexto de timestamping para usar en las peticiones entrantes.
- tsa_query_processor.c donde se parsea la petición HTTP y se lee el fichero tsq (timestamp query) recibido.
- tsa_timestamper.c hace uso de la librería ts.h para realizar el sellado.

El servidor va a necesitar su propio certificado exclusivo para sellar, junto a su clave privada. Esto se va a obtener mediante una Autoridad Certificadora (CA). Los comandos openssl para crear la clave y el certificado son los siguientes:

- Primero se crea una CA autofirmada. Primero se crea la llave y luego se usa para autofirmar el certificado de la CA:

  openssl req -new -newkey rsa:4096 -nodes -out ca.csr -keyout ca.key

  openssl x509 -trustout -signkey ca.key -days 1570 -req -in ca.csr -out ca.pem

- Se crea la llave privada de la TSA, con ella se genera una petición de certificado y finalmente esa petición se firma con la CA creada, obteniendo el certificado final de la TSA:

  openssl genrsa -out tsakey.pem 4096

  Request del certificado:

  openssl req -new -key tsakey.pem -out tsa.csr -nodes -days 1570 -config ext.config

  Se crea el certificado firmado por CA autoofirmada:

  openssl x509 -req -days 1570 -in tsa.csr -CA ../CA/ca.pem -CAkey ../CA/ca.key -set_serial 01 -out tsacrt.pem -extfile ext-only.config

El contenido de ext.config es la configuración personalizada que se tenga de openssl, y ext-only.config tan solo va a contener:

extendedKeyUsage=critical,timeStamping

Por si el cliente lo requiere, se crea una cadena de certificados que contenga el certificado de la CA y el de la TSA:

cat ca.pem > cert_chain.pem; cat tsacrt.pem >> cert_chain.pem

Eso indica que la llave que va a usar la TSA va a ser exclusivamente para sellar, no se podrá usar en ningún otro tipo de operación, como indica el estándar RFC3161.

Con todos los certificados configurados, se va a modificar la configuración openssl que va a usar el servidoor para que haga uso de ellos e indique al cliente la precisión de su reloj. La sección que hay que modificar es [tsa]:

```
[ tsa ]

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1

# These are used by the TSA reply generation only.
dir                     = ./tsa                 # TSA root directory
serial                  = ./tsa/tsaserial       # The current serial
number
crypto_device           = builtin               # OpenSSL engine to use
for signing
signer_cert             = ./tsa/tsacrt.pem      # The TSA signing
certificate
certs                   = ./cert_chain.pem      # Certificate chain to include in
reply
signer_key              = ./tsa/tsakey.pem      # The TSA private key
signer_digest           = sha256                # Signing digest to use
digests                 = sha256, sha384, sha512  # Acceptable message
digests
accuracy                = secs:0, millisecs:0, microsecs:1
clock_precision_digits  = 6     # number of digits after dot. (optional)
ordering                = no    # Is ordering defined for timestamps?
tsa_name                = yes   # Must the TSA name be included in the reply?
ess_cert_id_chain       = yes   # Must the ESS cert id chain be included?
```

clock_precision_digits = 6 es el máximo valor que puede tener.

7.2. NTP

7.2.1. Instalación

Para convertir la raspberry es un servidor de tiempo y comunicarse con el receptor se va a usar el paquete NTP, pero con una configuración personalizada para que pueda acceder al PPS del kernel.

Primero se descargan e instalan las dependencias para que NTP funcione correctamente:

```
apt build-dep ntp
```

Se descarga el código fuente para configurarlo antes de instalarlo:

```
apt source ntp
```

Dentro de la carpeta que desempaqueta apt, ntp-4.2.8p10+dfsg, se realizan las siguientes modificaciones:

- En debian/rules modificar los parámetroos de configure:

./configure CFLAGS='$(CFLAGS)' CPPFLAGS='$(CPPFLAGS)' LDFLAGS='$(LDFLAGS)' \
**--build=armv6l-unknown-linux-gnueabihf **
**--host=armv6l-unknown-linux-gnueabihf **
--enable-NMEA --enable-ATOM --enable-linuxcaps --enable-accurate-adjtime \

Todos los demás parámetros se han dejado como estaban. El valor de build y host se obtiene de ejecutar ./sntp/libevent/build-aux/config.guess.

- Cambiar la cifra de debian/compat por un 9, ya que si no al compilar dará un error de compatibilidad.

Una vez realizadas las modificaciones se compila con dpkg-buildpackage -b. Si esta compilación se hace directamente en la raspberry, va a tardar aproximadamente una hora. Puede ser que dé una advertencia como esta: tests/ntpd/Makefile.am:141: but option 'subdir-objects' is disabled. No supone ningún problema, así que se puede ignorar.

Cuando termina dla compilación se instalan los paquetes resultantes (ntp, documentación ntp y ntpdate):

dpkg -i ntp_4.2.8p10+dfsg-3+deb9u2PPS_armhf.deb ntpdate_4.2.8p10+dfsg-3+deb9u2PPS_armhf.deb ntp-doc_4.2.8p10+dfsg-3+deb9u2PPS_all.deb

Para evitar que NTP se sobreescriba en futuras actualizaciones: apt-mark hold ntp.

7.2.2. Configuración

En /etc/default/ntp se va a modificar NTPD_OPTS:

NTPD_OPTS='-g -N -D 5'

- -g es para que el primer cambio de hora pueda ser muy grande
- -N es para que el demonio se ejecute con máxima prioridad en la CPU
- -D 5 es el nivel de debug, útil para cuando falla algo y se quiere obtener más información

El archivo de configuracción principal ubicado en *etc/ntp.conf* queda de la siguiente manera:

```
# http://doc.ntp.org/4.2.8/monopt.html
driftfile /var/lib/ntp/drift
logfile /var/log/ntp.log
leapfile /usr/share/zoneinfo/leap-seconds.list
statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats protostats sysstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable
#filegen sysstats file sysstats type day enable
filegen protostats file sysstats type day enable

# http://www.ntp.org/ntpfaq/NTP-s-config.htm#AEN2891
# Junto con la opción -D 5 de ntpd recoge más información en /var/log/ntp.log
logconfig=all

# pool.ntp.org maps to about 1000 low-stratum NTP servers.  Your server will
# pick a different set every time it starts up.  Please consider joining the
# pool: <http://www.pool.ntp.org/join.html>
# pool es.pool.ntp.org iburst

# GPS NMEA
# minpoll 4 maxpoll 4: recoge información del GPS cada 2⁴ segundos
# mode 16: lee el puerto gps0 a 9600 bps
# flag1 1: procesa señal PPS, puerto gpspps0
# flag3 1: modo kernel
server 127.127.20.0 minpoll 4 maxpoll 4 prefer mode 16
fudge 127.127.20.0 flag1 1 flag3 1

# By default, exchange time with everybody, but don't allow configuration.
restrict -4 default kod notrap nomodify nopeer noquery limited
restrict -6 default kod notrap nomodify nopeer noquery limited

# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1
restrict ::1
```

Las primeras líneas hacen referencia a la generación de estadísticas, que son útiles para monitorear el funcionamiento del receptor y su comunicación con la raspberry.

Las líneas server y fudge indican de dónde se va a obtener la hora.

- 127.127 significa que es de un recurso local y 20 indica que el protocolo de transmisión es NMEA.
- minpoll 4 establece cada cuánto como mínimo el demonio NTPD tiene que leer los datos entrantes del receptor GPS, y maxpoll el máximo. Está indicado como potencia de 2, por lo que el intervalo es 2^4=16 segundos.
- prefer: establece el receptor como el servidor preferente
- flag1 1: habilita el procesamiento de la señal pps del gps mediante el puerto gpspps0
- flag3 1: habilita la modificación del reloj del kernel.

Las sentencias strict permite servir la hora a otros usuarios, pero con limitaciones de seguridad (excepto para la máquina local), indicadas en los argumentos.

Lo único que queda para tener NTP totalmente configurado es crear links a los puertos pps0 y ttyAMA0 para que NTP pueda acceder. Una forma de hacerlo de forma permanente es añadiendo lo siguiente a /etc/udev/rules.d/99-com.rules:

KERNEL=="ttyAMA0", MODE="0666", SYMLINK+="gps0"

KERNEL=="pps0", SUBSYSTEM=="pps", MODE="0666", SYMLINK+="gpspps0"

En este punto, con el módulo GPS conectado, se puede iniciar/reiniciar NTP para aplicar las configuraciones y esperar a que se calibre correctamente.

9. PRUEBAS Y RESULTADOS

9.1 NTP

Hay que tener en cuenta que el módulo GPS necesita varias horas hasta conseguir estar bien calibrado y equilibrado con el sistema. Además tiene que tener visión directa al exterior para tener buena cobertura. Una pared o estar a más de cinco metros de la ventana puede dificultar considerablemente la visualización de los satélites. Cuando la señal PPS parpadee significará que tiene cobertura y está en funcionamiento. Desde que se enciende hasta que empieza a proporcionar datos válidos puede tardar unos 15 minutos, y unas cuantas horas en conseguir la precisión adecuada.

Para comprobar que todo está en correcto funcionamiento se puede consultar lo siguiente:

- Al principio se pueden observar valores como estos:

```
root@raspberrypi:~# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*GPS_NMEA(0)     .GPS.            0 l    3   16    7    0.000   -0.431   0.227
```

El valor que tiene reach en la imagen de arriba corresponde a los valores normales de la secuencia de inicio: 0, 1, 3, 7, 17, 37, 77, 177, 377. Esta es una de las razones por las que aún no se tienen valores con precisión.

Una vez se ha estabilizado, los valores son muy distintos:

```
root@raspberrypi:~# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
oGPS_NMEA(0)     .GPS.            0 l   10   16  377    0.000   -0.001   0.002
```

- Con el comando dmesg o leyendo /var/log/kern.log se verifica que el kernel consumer está asociado al puerto que recibe la señal PPS:

```
[   25.756549] pps pps0: bound kernel consumer: edge=0x1
[   27.679826] hardpps: PPSJITTER: jitter=310142000, limit=0
```

- En los logs de NTP se puede ver que el kernel PPS ha sido activado:

```
 1 Jan 21:43:23 ntpd[437]: Listening on routing socket on fd #22 for interface updates
 1 Jan 21:43:23 ntpd[437]: 0.0.0.0 c01d 0d kern kernel time sync enabled
 1 Jan 21:43:23 ntpd[437]: 0.0.0.0 c012 02 freq_set kernel 0.000 PPM
 1 Jan 21:43:23 ntpd[437]: 0.0.0.0 c011 01 freq_not_set
 1 Jan 21:43:23 ntpd[437]: 0.0.0.0 c016 06 restart
 1 Jan 21:43:24 ntpd[437]: 0.0.0.0 c41c 0c clock_step +10.878804 s
 1 Jan 21:43:35 ntpd[437]: 0.0.0.0 c414 04 freq_mode
 1 Jan 21:43:51 ntpd[437]: 0.0.0.0 c418 08 no_sys_peer
 1 Jan 21:48:39 ntpd[437]: kernel reports TIME_ERROR: 0x141: Clock Unsynchronized
 1 Jan 21:48:39 ntpd[437]: 0.0.0.0 c412 02 freq_set kernel -7.080 PPM
 1 Jan 21:48:39 ntpd[437]: 0.0.0.0 c415 05 clock_sync
 1 Jan 21:49:27 ntpd[437]: 0.0.0.0 041d 0d kern PPS enabled status: 0001 -> 0007
root@raspberrypi:~#
```

- Puede que cuando se quiera leer el puerto UART del GPS se visualicen caracteres extraños. Esto puede ser debido a que la velocidad de recepción del puerto de la rpi no corresponde con la velocidad de transmisión del puerto del módulo. Por defecto, el módulo viene configurado a 9 600 bps, por lo que esto se podría arreglar ajustando la recepción en la rpi: stty -F dev/ttyAMA0 ispeed 9600.

```
root@raspberrypi:~# cat /dev/ttyAMA0

���HN�kH�JL�IZ{iZ*I������������
                          iJ*ZZ�IZ{iZ�yi+Z��
H��
   xO���I!
         )���H�X:(��K
                   JN���
�
�LYI1L
      ��y���KLL�����


                   N�wYYY*zI�)1�L���
                                 �

JL�H����N�wYIY*�HL��

��L��
    �JL��N�wYYY*zy1!�
```

```
root@raspberrypi:~# stty -F /dev/ttyAMA0
speed 4800 baud; line = 0;
intr = <undef>; quit = <undef>; erase = <undef>; kill = <undef>; eof
werase = <undef>; lnext = <undef>; discard = <undef>;
ignbrk -brkint -imaxbel
-opost -onlcr
-isig -iexten -echo -echoe -echok -echoctl -echoke
root@raspberrypi:~# stty -F /dev/ttyAMA0 ispeed 9600
stty: /dev/ttyAMA0: unable to perform all requested operations
root@raspberrypi:~# minicom -b 9600 -o -D /dev/ttyAMA0
-bash: minicom: command not found
root@raspberrypi:~# stty -F /dev/ttyAMA0 ispeed 9600
root@raspberrypi:~# stty -F /dev/ttyAMA0
speed 9600 baud; line = 0;
```

Aunque también se pueden aplicar otras configuraciones:

Possible UART Interface Configurations

Baud Rate	*Data Bits*	*Parity*	*Stop Bits*
4800	8	none	1
9600	8	none	1
19200	8	none	1
38400	8	none	1
57600	8	none	1
115200	8	none	1

- Con la utilidad wiringpi (apt install wiringpi) gpio readall se puede consultar el estado de los gpios. Los pertenecientes a los puertos UART, los pines físicos 8 y 10, tienen que estar en modo ALT0. Si tuviesen el modo ALT5 indicaría que se están usando los puertos UART secundarios, mini UART, los cuales no son tan eficientes ya que, por ejemplo, carecen de control de flujo y son más propensos a perder caracteres. [41].

```
root@raspberrypi:~# gpio readall
 +-----+-----+---------+------+---+-Pi ZeroW-+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+-Pi ZeroW-+---+------+---------+-----+-----+
```

- Se puede observar que el assert del PPS, depués de unas horas funcionando, se ha sincronizado con el sistema ya que las fracciones del segundo ya no varían tanto (el jitter ha reducido).

```
source 0 - assert 1546400353.999998191, sequence: 17617 - clear  0.000000000, sequence: 0
source 0 - assert 1546400355.000003178, sequence: 17618 - clear  0.000000000, sequence: 0
source 0 - assert 1546400356.000001220, sequence: 17619 - clear  0.000000000, sequence: 0
^C
root@raspberrypi:~# ppstest /dev/pps1
trying PPS source "/dev/pps1"
found PPS source "/dev/pps1"
ok, found 1 source(s), now start fetching data...
source 0 - assert 1546400362.306008346, sequence: 345 - clear  0.000000000, sequence: 0
source 0 - assert 1546400363.346003774, sequence: 346 - clear  0.000000000, sequence: 0
source 0 - assert 1546400364.386002972, sequence: 347 - clear  0.000000000, sequence: 0
source 0 - assert 1546400365.426004458, sequence: 348 - clear  0.000000000, sequence: 0
source 0 - assert 1546400366.466026463, sequence: 349 - clear  0.000000000, sequence: 0
source 0 - assert 1546400367.506015654, sequence: 350 - clear  0.000000000, sequence: 0
source 0 - assert 1546400368.546017825, sequence: 351 - clear  0.000000000, sequence: 0
source 0 - assert 1546400369.586004084, sequence: 352 - clear  0.000000000, sequence: 0
source 0 - assert 1546400370.625998326, sequence: 353 - clear  0.000000000, sequence: 0
source 0 - assert 1546400371.665991493, sequence: 354 - clear  0.000000000, sequence: 0
source 0 - assert 1546400372.705988554, sequence: 355 - clear  0.000000000, sequence: 0
source 0 - assert 1546400373.747351476, sequence: 356 - clear  0.000000000, sequence: 0
source 0 - assert 1546400374.785987798, sequence: 357 - clear  0.000000000, sequence: 0
^C
root@raspberrypi:~# ppstest /dev/gpspps0
trying PPS source "/dev/gpspps0"
found PPS source "/dev/gpspps0"
ok, found 1 source(s), now start fetching data...
source 0 - assert 1546400392.000000154, sequence: 17655 - clear  0.000000000, sequence: 0
source 0 - assert 1546400392.999998201, sequence: 17656 - clear  0.000000000, sequence: 0
source 0 - assert 1546400394.000003178, sequence: 17657 - clear  0.000000000, sequence: 0
source 0 - assert 1546400395.000000220, sequence: 17658 - clear  0.000000000, sequence: 0
source 0 - assert 1546400396.000003172, sequence: 17659 - clear  0.000000000, sequence: 0
source 0 - assert 1546400396.999999207, sequence: 17660 - clear  0.000000000, sequence: 0
source 0 - assert 1546400398.000000166, sequence: 17661 - clear  0.000000000, sequence: 0
source 0 - assert 1546400399.000001195, sequence: 17662 - clear  0.000000000, sequence: 0
source 0 - assert 1546400400.000000196, sequence: 17663 - clear  0.000000000, sequence: 0
source 0 - assert 1546400401.000000184, sequence: 17664 - clear  0.000000000, sequence: 0
source 0 - assert 1546400402.000006189, sequence: 17665 - clear  0.000000000, sequence: 0
source 0 - assert 1546400402.999996226, sequence: 17666 - clear  0.000000000, sequence: 0
^C
```

- Una vez se ha comprobado que todas las configuraciones son correctas, se examina lo más importante: la precisión y estabilidad que ha alcanzado el dispositivo GPS. Ejecutando en la terminal ntptime se obtiene:

```
root@raspberrypi:~# ntptime
ntp_gettime() returns code 0 (OK)
  time dfe71adf.d953dea4  Mon, Jan 14 2019 14:27:43.848, (.848936955),
  maximum error 9000 us, estimated error 2 us, TAI offset 37
ntp_adjtime() returns code 0 (OK)
  modes 0x0 (),
  offset 0.000 us, frequency -8.406 ppm, interval 256 s,
  maximum error 9000 us, estimated error 2 us,
  status 0x2107 (PLL,PPSFREQ,PPSTIME,PPSSIGNAL,NANO),
  time constant 4, precision 0.001 us, tolerance 500 ppm,
  pps frequency -8.406 ppm, stability 0.034 ppm, jitter 0.892 us,
  intervals 99, jitter exceeded 5, stability exceeded 0, errors 0.
```

Comparando los valores con un ejemplo de la página oficial de NTP [42] se puede concluir:

- status es correcto, indica que la señal PPS se está usando.

- jitter exceeded 5 indica que rechazó 5 pulsos cuando en realidad habían llegado a tiempo. Está dentro del margen de error.
- stability 0.034 ppm está por debajo de 0.1 ppm, indica un sistema estable.
- jitter 0.892 us indica lo que varían los pulsos entre cada segundo. Está dentro del margen de error. Si fuesen milisegundos no sería normal.
- pps frequency -8.406 ppm indica que el reloj del sistema es más rápido que la frecuencia PPS. Es un valor que también está dentro de lo normal.
- offset 0.000 us es la última corrección que se tuvo que hacer al sistema.

Hay otros comandos, como ntpq -c peer -c as -c rl o ntpq -ccv -p -crv -ckern -csysinfo, que proporcionan también información útil. En el Anexo hay definidos muchos de los términos mostrados.

9.2. TSA

Mientras el servidor TSA estaba procesando 250 peticiones que le habían llegado de golpe, se ha observado que el jitter ha ascendido como mucho a un valor de 4 microsegundos, que sigue siendo un valor normal:

```
Every 1.0s: ntptime

ntp_gettime() returns code 0 (OK)
  time dfe721d7.30aed99c  Mon, Jan 14 2019 14:57:27.190, (.190168515),
  maximum error 5000 us, estimated error 1 us, TAI offset 37
ntp_adjtime() returns code 0 (OK)
  modes 0x0 (),
  offset 0.000 us, frequency -8.343 ppm, interval 256 s,
  maximum error 5000 us, estimated error 1 us,
  status 0x2107 (PLL,PPSFREQ,PPSTIME,PPSSIGNAL,NANO),
  time constant 4, precision 0.001 us, tolerance 500 ppm,
  pps frequency -8.344 ppm, stability 0.025 ppm, jitter 3.956 us,
  intervals 106, jitter exceeded 5, stability exceeded 0, errors 0.
```

De las 250 peticiones, el servidor ha rechazado 9. Las demás las ha sellado todas correctamente.

Para realizar las pruebas para el servidor TSA se han usado los comandos de openssl ts y tsget en un script de bash que lanza n peticiones paralelas:

- Para crear una tsq (timestamp query):

  openssl ts -query -sha256 -cert -data datos.txt -out peticion.tsq

  A partir de un archivo de texto normal se ha calculado su sha256 y con él se ha construido un arcchivo binario listo para enviar al servidor.

- Para enviar la solicitud de sellado:

  tsget -h rpi:318 peticion.tsq

  tsget usa una extensión perl de la librería curl. Crea una petición HTTP con un Expect: 100 continue. La respuesta la almacena en peticion.tsr.

- Para verificar el sello:

  openssl ts -verify -in peticion.tsr -queryfile peticion.tsq -untrusted tsacrt.pem -CAfile ca.pem

  Se necesita tanto el certificado de la TSA como la CA que ha emitido ese certificado para la verificación.

El script informará de todos los sellados fallidos y el tiempo transcurrido. Esta es la salida del script una vez ha enviado 250 peticiones y las ha verificado:

```
Host:
        192.168.1.135:318
Num. ficheros a los que realizar timestamp:
        250

Preparando peticiones... 250 de 250
Lanzando peticiones... 250 de 250

Todas las peticiones lanzadas. Esperando.
Hecho.

Tiempo desde que se empiezan a lanzar las 250 peticiones hasta que se recibe respuesta de todas:
102539 ms

Verificando respuestas...
        Verificando 250 de 250
Total veificaciones fallidas: 0

Resumen:
        Tiempo para 250 sellados:       102539 ms
        Sellados erroneos:      0
```

Con el comando openssl ts -query -in query.tsq -text se puede ver el contenido de una timestamp query en formato legible:

```
Version: 1
Hash Algorithm: sha256
Message data:
    0000 - 43 55 a4 6b 19 d3 48 dc-2f 57 c0 46 f8 ef 63 d4   CU.k..H./W.F..c.
    0010 - 53 8e bb 93 60 00 f3 c9-ee 95 4a 27 46 0d d8 65   S...`.....J'F..e
Policy OID: unspecified
Nonce: 0x4F996B2E0AC9AAD9
Certificate required: yes
Extensions:
```

Con el comando openssl ts -reply -in reply.tsr -text se puede ver el contenido de una timestamp response en formato legible. En el campo Time stamp se pueden ver los 6 dígitos de precisión:

```
Status info:
Status: Granted.
Status description: unspecified
Failure info: unspecified

TST info:
Version: 1
Policy OID: tsa_policy1
Hash Algorithm: sha256
Message data:
    0000 - 43 55 a4 6b 19 d3 48 dc-2f 57 c0 46 f8 ef 63 d4   CU.k..H./W.F..c.
    0010 - 53 8e bb 93 60 00 f3 c9-ee 95 4a 27 46 0d d8 65   S...`.....J'F..e
Serial number: 0x05
Time stamp: Jan 14 14:57:04.302791 2019 GMT
Accuracy: unspecified seconds, unspecified millis, 0x01 micros
Ordering: no
Nonce: 0x4F996B2E0AC9AAD9
TSA: DirName:/C=ES/ST=Madrid/L=Madrid/O=TFG/OU=TFG/CN=tsaServer
Extensions:
```

10. CONCLUSIONES

Con los resultados obtenidos tras el desarrollo del proyecto se concluye que por un precio reducido se ha podido implementar un sistema de sellado de tiempo preciso capaz de ser ejecutado eficientemente en sistemas con pocos recursos computacionales, como lo es la Raspberry Pi Zero W usada como servidor.

Con la implementación del sistema diseñado se ha hecho el uso conjunto de la TSA y del servicio NTP sin generar conflictos y cumpliendo las funcionalidades esperadas de ambas partes, incluso cuando el servidor TSA ha tenido una carga elevada.

Además, no se ha empleado ninguna librería de terceros que no sea de confianza; las librerías usadas son propias del sistema y lo relacionado con el timestamping y NTP se ha obtenido de repositorios oficiales, todo ello cumpliendo con los estándares TSP y NTP.

De este proyecto también se puede destacar que desde sus inicios ha sido diseñado y posteriormente desarrollado para ser multiplataforma, ya que ha sido desarrollado en C, un lenguaje que está presente en todos los sistemas operativos más usados en servidores y es uno de los más eficientes si se siguen las buenas prácticas.

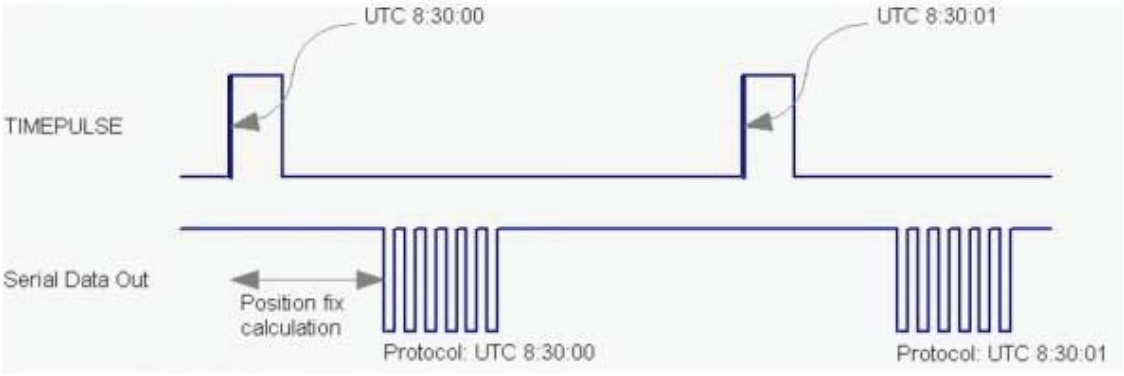
Es un proyecto que puede llegar a ser ampliado en diversas líneas, ya que está lleno de detalles que admiten múltiples configuraciones, como lo es el kernel, al cual se le podría desactivar todo desde un principio y examinar cada uno de sus módulos exhaustivamente para personalizar completamente su funcionamiento.

11. ANEXO

11.1. Definiciones

11.1.1 PPS (Pulse-per-second)

Es una señal eléctrica que genera un dispositivo una vez por segundo con un flanco de subida o bajada muy brusco, consiguiendo así una alta precisión [43]. Dicha señal es comúnmente usada junto al tiempo actual para mantener el reloj de un sistema precisamente sincronizado. Dicha señal es generada por relojes atómicos, receptores GNSS y otros dispositivos que generen señales precisas.



11.1.2. Receptor GNSS (Global Navigation Satellite System)

El Sistema Global de Navegación por Satélite [44] es una constelación de satélites artificiales que transmite información de posicionamiento y tiempo de gran exactitud a cualquier parte del mundo y a cualquier hora. Las constelaciones más importantes son:

- GALILEO: sistema desarrollado por la Unión Europea para uso civil
- GPS: sistema de posicionamiento desarrollado por el Departamento de Defensa Estados Unidos
- GLONASS: sistema desarrollado a cargo del Ministerio de Defensa de la Federación Rusa
- BEIDOU: sistema controlado por China

11.1.3. Precisión

La precisión del sistema en segundos redondeado a la siguiente potencia de dos.

11.1.4. Root delay

Es el retraso total de ida y vuelta a la fuente de referencia principal en segundos.

11.1.5. Root dispersión

Valor mayor que cero que indica en segundos el error máximo relativo a la fuente de referencia principal.

11.1.6. Offset

La diferencia de tiempo entre dos relojes relativos al reloj de referencia seleccionado. Representa la cantidad a ajustar el reloj local para parearlo al reloj de referencia.

11.1.7. sys_jitter

Es el error ligado en el desplazamiento y debe ser menor de 1ms.

11.1.8. clk_jitter

Variación a corto plazo en la frecuencia del reloj. Debería ser menos de 1 ms.

11.1.9. clk_wander

Variación a largo plazo en la frecuencia del reloj. Debería ser menos de 1 PPM.

11.1.10. TAI

Siglas del francés que significan Tiempo Atómico Internacional y es un estándar de tiempo de coordenadas atómicas de alta precisión. Es la base del Tiempo Universal Coordinado (UTC) aunque TAI está adelantado 37 segundos. El tiempo se consigue haciendo una media de 400 relojes atómicos. En el comando ntp es la diferencia de hora entre TAI y UTC (TAI-UTC).

12. BIBLIOGRAFÍA

[1] https://www.gps.gov/systems/gps/space/

[2] http://www.physics.org/article-questions.asp?id=55

[3] https://spaceplace.nasa.gov/gps/en/

[4] http://www.gps-basics.com/faq/q0120.shtml

[5] http://www.timetoolsglobal.com/information/gps-ntp-server/

[6] https://en.wikipedia.org/wiki/Pulse-per-second_signal

[7] http://gisgeography.com/gps-accuracy-hdop-pdop-gdop-multipath/

[8] http://www.gpsntp.com/ntpserver/using-satellite-signals-in-gps-ntp-server.php

[9] http://www.physics.org/article-questions.asp?id=55

[10] http://www.astronomy.ohio-state.edu/~pogge/Ast162/Unit5/gps.html

[11] https://en.wikipedia.org/wiki/Trusted_timestamping

[12] http://www.satsignal.eu/ntp/Raspberry-Pi-NTP.html

[13] https://aardvarklabs.wordpress.com/2013/12/29/ntp-server-using-raspberry-pi-and-vp-oncore-gps-module-part-3/

[14] https://wiki.archlinux.org/index.php/Network_Time_Protocol_daemon

[15] http://www.satsignal.eu/mrtg/performance_raspi-1.php

[16] https://www.openssl.org/docs/manmaster/man1/ts.html

[17] https://github.com/openssl/openssl/tree/master/crypto/ts

[18] https://tools.ietf.org/html/rfc5905

[19] https://www.cvedetails.com/cve/CVE-2016-9310/

[20] https://www.kb.cert.org/vuls/id/568372/

[21] https://tools.ietf.org/html/rfc3161

[22] https://es.wikipedia.org/wiki/Sellado_de_tiempo_confiable

[23] https://tools.ietf.org/html/rfc2783

[24] https://www.raspberrypi.org/products/raspberry-pi-zero-w/

[25]https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[26] https://es.pinout.xyz/

[27] https://www.amazon.com/G28U7FTTL-UBX-G7020-KT-Monitoring-Navigation-DIYmall/dp/B015R62YHI/ref=cm_cr_arp_d_product_top?ie=UTF8

[28] https://www.u-blox.com/en/product/ubx-g7020-series

[29] https://www.u-blox.com/sites/default/files/products/documents/UBX-G7020_ProductSummary_%28UBX-13003349%29.pdf

[30] https://www.raspberrypi.org/downloads/raspbian/

[31] https://www.raspberrypi.org/documentation/linux/kernel/building.md

[32] https://www.raspberrypi.org/documentation/linux/kernel/configuring.md

[33] https://www.u-blox.com/sites/default/files/products/documents/u-blox7-V14_ReceiverDescriptionProtocolSpec_%28GPS.G7-SW-12001%29_Public.pdf

[34] https://www.navcen.uscg.gov/?Do=constellationStatus

[35] https://egnos-user-support.essp-sas.eu/new_egnos_ops/egnos-system/about-egnos

[36] https://www.glonass-iac.ru/en/GLONASS

[37] https://www.glonass-iac.ru/en/GPS

[38] https://egnos-user-support.essp-sas.eu/new_egnos_ops/egnos_ops/?q=egnos_system_realtime

[39] https://www.openssl.org/docs/manmaster/man1/ts.html

[40] https://github.com/openssl/openssl/tree/master/crypto/ts

[41] https://www.raspberrypi.org/documentation/configuration/uart.md

[42] http://www.ntp.org/ntpfaq/NTP-s-config-adv.htm#Q-CONFIG-ADV-PPS-VERIFY

[43] http://pos.mgb-tech.com/insightpps/

[44] https://nagarvil.webs.upv.es/receptores-gnss-receptores-gps/

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Mon Jan 14 23:56:34 CET 2019
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)