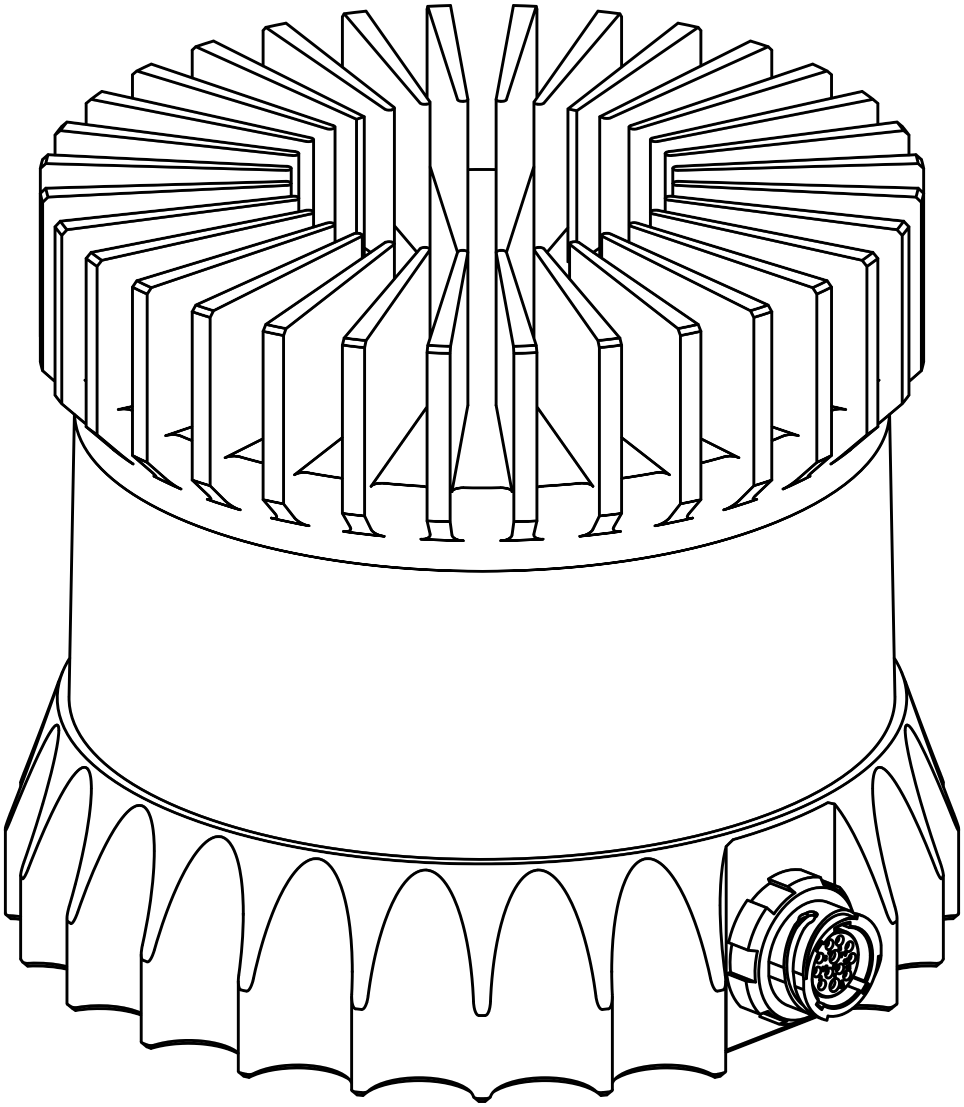


Software User Guide

Release v1.13.0

OS1-16/64 High Resolution Imaging Lidar

Nov 07, 2019

Software User Guide

1 Introduction

The OS1 family of sensors offer a market leading combination of price, performance, reliability and SWaP (Size, Weight, and Power). They are designed for indoor/outdoor all-weather environments and long lifetime. As the smallest high performance lidar on the market, the OS1 can be directly integrated into vehicle fascias, windshields, side mirrors, and headlight clusters. The OS1 family of sensors consist of two models, the OS1-16 and OS1-64, with differing resolution, but of identical mechanical dimensions.

→

HIGHLIGHTS

- Fixed resolution per frame operating mode
- Camera-grade intensity, ambient, and range data
- Multi-sensor crosstalk immunity
- Simultaneous and co-calibrated 2D and 3D output
- Industry leading intrinsic calibration
- Example client code available

For the purposes of this document, the term "OS1" refers to the family of sensors, and only where there is a difference in performance will each model be referred to by its specific model designation.

2 Safety & Legal Notices

The OS1-16 and OS1-64 are Class 1 laser products per IEC 60825-1:2014 and operate in the 850nm band.

FDA 21CFR1040 Notice: OS1-16 and OS1-64 comply with FDA performance standards for laser products except for deviations pursuant to Laser Notice No. 50, dated July 26th, 2001.

CLASS 1
Laser product

CAUTION
SHARP EDGES
Handle with care

WARNING: The OS1 is a sealed unit, and is not user-serviceable.

Your use of the OS1 is subject to the Terms of Sale that you signed with Ouster or your distributor/integrator. Included in these terms is the prohibition on removing or otherwise opening the sensor housing, inspecting the internals of the sensor, reverse-engineering any part of the sensor, or permitting any third party to do any of the foregoing.

"Ouster" and "OS1" are both registered trademarks of Ouster, Inc. They may not be used without express permission from Ouster, Inc.

If you have any questions about the above points, contact us at legal@ouster.io.

3 Drivers & Interface

By default, when newly provided power by the Interface Box, the sensor will start-up and then automatically start taking measurements, request an IP address, and stream UDP data packets to the configured destination address. Settings can be modified using a simple plaintext protocol over TCP.

Ouster provides sample code for connecting to the sensor, visualizing the output data, and interfacing with the popular ROS robotics suite. The source code repository can be found at: www.github.com/ouster-lidar/ouster_example

3.1 Network Configuration

Before attempting to configure and stream data from the sensor, please ensure that it is reachable over the network from the client PC. The OS1 requires a network that can provide data throughput of approximately 129 Mbps between client and the sensor, and a DHCP server to reliably connect and stream data. Gigabit Ethernet hardware is recommended.

In a typical network environment, the OS1 should obtain a DHCP lease and be reachable over the network a few moments after being plugged in. If your network is set up to provide DNS for DHCP clients, you should be able to check for connectivity using e.g.:

```
$ ping -c1 os1-991900123456
PING os1-991900123456 (10.5.5.94) 56(84) bytes of data.
64 bytes from os1-991900123456 (10.5.5.94): icmp_seq=1 ttl=64 time=0.163 ms

--- os1-991900123456 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.163/0.163/0.163/0.000 ms
```

where "991900123456" is the serial number printed on the top of the sensor.

Running A Local DHCP Server

If the sensor is plugged directly into the client machine, you will have to install and run a local DHCP server or use the IPV4 override mechanism described below. A common choice is *dnsmasq*, which is available for a variety of platforms.

Note: When connecting directly to a client machine, hostnames require a '.local' appended to them. e.g.: os1-991900123456.local

To connect to the sensor using a local dnsmasq instance on Linux:

1. Identify the Ethernet interface to be used on the client (Linux) machine, e.g., enp6s0f1
2. Check that the sensor is **not** plugged in to the Ethernet interface on the client machine
3. Make sure that the Ethernet interface is "down" and not yet configured:

```
$ ip addr flush dev enp6s0f1
$ ip addr show dev enp6s0f1
2: enp6s0f1: <BROADCAST,MULTICAST> ... state DOWN group default qlen 1000
```

4. Assign a static IP to the chosen interface:

```
$ sudo ip addr add 10.5.5.1/24 dev enp6s0f1
```

5. Connect an Ethernet cable between the sensor and the designated Ethernet interface on the client machine. Power-on the sensor. Ensure that the link is now "up":

```
$ sudo ip link set enp6s0f1 up
$ ip addr show dev enp6s0f1
2: enp6s0f1: <BROADCAST,MULTICAST,UP> ... state UP group default qlen 1000
link/ether xx:xx:xx:xx:xx:xx brd ff:ff:ff:ff:ff:ff
inet 10.5.5.1/24 scope global enp6s0f1
   valid_lft forever preferred_lft forever
```

6. Run dnsmasq to listen for DHCP requests on the chosen interface:

```
$ sudo dnsmasq -C /dev/null -kd -F 10.5.5.50,10.5.5.100 -i enp6s0f1 --bind-dynamic
```

7. Within 10-15 seconds, you should see the DHCP negotiation take place:

```
dnsmasq-dhcp: DHCP, IP range 10.5.5.50 -- 10.5.5.100, lease time 1h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface enp6s0f1
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 127.0.1.1#53
dnsmasq: read /etc/hosts - 7 addresses
dnsmasq-dhcp: DHCPDISCOVER(enp6s0f1) xx:xx:xx:xx:xx:xx
dnsmasq-dhcp: DHCPOFFER(enp6s0f1) 10.5.5.94 xx:xx:xx:xx:xx:xx
dnsmasq-dhcp: DHCPREQUEST(enp6s0f1) 10.5.5.94 xx:xx:xx:xx:xx:xx
dnsmasq-dhcp: DHCPACK(enp6s0f1) 10.5.5.94 xx:xx:xx:xx:xx:xx os1-991900123456
```

8. Check connectivity via the assigned IP address:

```
$ ping -c1 10.5.5.94
PING 10.5.5.94 (10.5.5.94) 56(84) bytes of data.
64 bytes from 10.5.5.94: icmp_seq=1 ttl=64 time=0.404 ms

--- 10.5.5.94 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.404/0.404/0.404/0.000 ms
```

9. Alternatively check connectivity via the local hostname:

```
$ ping -c1 os1-991900123456.local
PING os1-991900123456.local (10.5.5.94) 56(84) bytes of data.
64 bytes from os1-991900123456.local (10.5.5.94): icmp_seq=1 ttl=64 time=0.404 ms

--- os1-991900123456.local ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.404/0.404/0.404/0.000 ms
```

10. See the documentation for your operating system on how to make these changes persistent, e.g., by using a network configuration daemon like NetworkManager.

3.2 HTTP Interface

The lidar sensor hosts an HTTP server that implements a versioned RESTful API to enable programmatic command and control of the sensor.

This HTTP interface is a convenient way to query the sensor for various information. The query can be done from the command line, programmatically, or from within a web browser.

All of the responses are JSON objects and best handled with an HTTP client that can interpret and present JSON objects.

→

Note: See the *HTTP API Reference* for more information on usage of individual resources.

Clients and Tools

Many readily available tools exist to facilitate communication with the HTTP API.

- *Web browser*
- *Command line*
- *Programmatic access*

Web browser

The simplest way to access the API is to issue a `GET` request using the web browser.

For example, to view the network configuration of a sensor load the following URL in a web browser:

http://os1-991900123456/api/v1/system/network

Note: When connecting a sensor directly to a client machine, hostnames require a '.local' appended to them. e.g.: http://os1-991900123456.local/api/v1/system/network

Extensions aid in viewing the JSON objects. Firefox includes a JSON formatter by default and the Chrome web store offers the JSON Formatter.

REST clients are also available to send HTTP requests using things other than the `GET` method. The Chrome web store offers the Advanced REST client.

Command line

For automated access from the command line a few tools exist for HTTP clients as well as JSON formatters.

The httpie tool serves as an HTTP client as well as JSON formatter. Example usage:

```
$ http http://os1-991900123456/api/v1/system/network
HTTP/1.1 200 OK
content-length: 260
content-type: application/json; charset=UTF-8

{
    "carrier": true,
    "duplex": "full",
    "ethaddr": "bc:0f:a7:00:01:2c",
    "hostname": "os1-991900123456",
    "ipv4": {
        "addr": "192.0.2.123/24",
        "link_local": "169.254.245.183/16",
        "override": null
    },
    "ipv6": {
        "link_local": "fe80::be0f:a7ff:fe00:12c/64"
    },
    "speed": 1000
}
```

Curl is a popular and high performance command line tool (backed by `libcurl`), example usage is as follows:

```
$ curl -s http://os1-991900123456/api/v1/system/network
{"ipv6": {"link_local": "fe80::be0f:a7ff:fe00:12c/64"}, "ethaddr": "bc:0f:a7:00:01:2c", "ipv4": {"override
↪": null, "link_local": "169.254.245.183/16", "addr": "192.0.2.123/24"}, "speed": 1000, "duplex": "full",
↪"carrier": true, "hostname": "os1-991900123456"}
```

The unformatted JSON output is hard to read, but this can be improved with the command line jq formatter:

```
$ curl -s http://os1-991900123456/api/v1/system/network | jq -S
{
  "carrier": true,
  "duplex": "full",
  "ethaddr": "bc:0f:a7:00:01:2c",
  "hostname": "os1-991900123456",
  "ipv4": {
    "addr": "192.0.2.123/24",
    "link_local": "169.254.245.183/16",
    "override": null
  },
  "ipv6": {
    "link_local": "fe80::be0f:a7ff:fe00:12c/64"
  },
  "speed": 1000
}
```

Programmatic access

Several popular libraries are available for interfacing with the sensor via common programming languages:

- C - libcurl
- Python - requests

Response Codes

The HTTP API uses HTTP response codes to provide result status information. The sensor will return HTTP response codes that comply with the HTTP standards.

The `http` command shows the response code as part of the returned response in the console.

Common response code classes:

→

2XX Request was success (e.g., when a `GET` command succeeded)

4XX Client request error not allowed (e.g., when a `POST` on `system/time` is attempted)

5XX Server errors

3.3 TCP API Command Set

Querying Sensor Info and Intrinsic Calibration

The sensor can be queried and configured using a simple plaintext protocol over TCP on port 7501.

An example session using the unix netcat utility is shown below:

```
$ nc os1-991900123456 7501
get_sensor_info
{"status": "RUNNING", "image_rev": "ousteros-image-prod-aries-v1.13.0-20190828230129",
 "base_pn": "000-101323-03", "prod_sn": "991900123456", "proto_rev": "v1.1.1",
 "base_sn": "101837000752", "prod_line": "OS-1-64", "build_rev": "v1.13.0",
 "prod_pn": "840-101855-02", "build_date": "2019-08-28T22:58:18Z"}
```

A sensor may have one of the following statuses:

Status	Description
INITIALIZING	When the sensor is booting and not yet outputting data.
UPDATING	When the sensor is updating the FPGA firmware on the first reboot after a firmware upgrade.
RUNNING	When the sensor has reached the final running state where it can output data.
ERROR	Check error codes in the errors field for more information.
UNCONFIGURED	An error with factory calibration that requires a return.

If sensor is in an ERROR or UNCONFIGURED state, please contact Ouster support with the two diagnostic files found at http://os1-9919xxxxxxxx/diag for support.

The following commands will return sensor configuration and calibration information:

Table1: Sensor Configuration and Calibration

Command	Description	Response Example

get_config_txt	Returns JSON-formatted sensor configuration.	{
		"timestamp_mode": "TIME_FROM_INTERNAL_OSC",
		"multipurpose_io_mode": "OFF",
		"nmea_leap_seconds": 0,
		"lidar_mode": "1024x10",
		"sync_pulse_in_polarity": "ACTIVE_HIGH",
		"nmea_in_polarity": "ACTIVE_HIGH",
		"sync_pulse_out_polarity": "ACTIVE_HIGH",
		"udp_ip": "192.0.2.123",
		"nmea_ignore_valid_char": 0,
		"auto_start_flag": 1,
		"sync_pulse_out_pulse_width": 10,
		"nmea_baud_rate": "BAUD_9600",
		"sync_pulse_out_angle": 360,
		"sync_pulse_out_frequency": 1,
		"udp_port_imu": 7503,
		"udp_port_lidar": 7502,
		"azimuth_window": [0, 36000]
		}

get_sensor_info	Returns JSON-formatted sensor metadata: serial number, hardware and software revision, and sensor status.	{
		"status": "RUNNING",
		"image_rev": "ousteros-image-prod-aries-v1.13.0-20190828230129",
		"base_pn": "000-101323-03",
		"prod_sn": "991900123456",
		"proto_rev": "v1.1.1",
		"base_sn": "101837000752",
		"prod_line": "OS-1-64",
		"build_rev": "v1.13.0",
		"prod_pn": "840-101855-02",
		"build_date": "2019-08-28T22:58:18Z"
		}

Continued on next page

Table 1 - continued from previous page

<table>
<tr><th>Command</th><th>Description</th><th>Response Example</th></tr>
<tr><td>get_time_info</td><td>Returns JSON-formatted sensor timing configuration and status of udp timestamp, sync_pulse_in, and multipurpose_io.</td><td><pre>
{
   "timestamp": {
      "time": 302.96139565999999,
      "mode": "TIME_FROM_INTERNAL_OSC",
      "time_options": {
         "sync_pulse_in": 0,
         "internal_osc": 302,
         "ptp_1588": 309
      }
   },
   "sync_pulse_in": {
      "locked": 0,
      "diagnostics": {
         "last_period_nsec": 0,
         "count_unfiltered": 1,
         "count": 0
      },
      "polarity": "ACTIVE_HIGH"
   },
   "multipurpose_io": {
      "mode": "OFF",
      "sync_pulse_out": {
         "pulse_width_ms": 10,
         "angle_deg": 360,
         "frequency_hz": 1,
         "polarity": "ACTIVE_HIGH"
      },
      "nmea": {
         "locked": 0,
         "baud_rate": "BAUD_9600",
         "diagnostics": {
            "io_checks": {
               "bit_count": 1,
               "bit_count_unfilterd": 0,
               "start_char_count": 0,
               "char_count": 0
            },
            "decoding": {
               "last_read_message": "",
               "date_decoded_count": 0,
               "not_valid_count": 0,
               "utc_decoded_count": 0
            }
         },
         "leap_seconds": 0,
         "ignore_valid_char": 0,
         "polarity": "ACTIVE_HIGH"
      }
   }
}
</pre></td></tr>
</table>

Continued on next page

Table 1 - continued from previous page

Command	Description	Response Example

`get_beam_intrinsics`	Returns JSON-formatted beam altitude and azimuth offsets, in degrees.	`{`
		`"beam_altitude_angles": [`
		`16.926,`
		`16.313,`
		`"...",`
		`-16.078,`
		`-16.689`
		`],`
		`"beam_azimuth_angles": [`
		`3.052,`
		`0.857,`
		`"...",`
		`-0.868,`
		`-3.051`
		`]`
		`}`

`get_imu_intrinsics`	Returns JSON-formatted IMU transformation matrix needed to adjust to the Sensor Coordinate Frame.	`{`
		`"imu_to_sensor_transform": [`
		`1,`
		`0,`
		`0,`
		`6.253,`
		`0,`
		`1,`
		`0,`
		`-11.775,`
		`0,`
		`0,`
		`1,`
		`7.645,`
		`0,`
		`0,`
		`0,`
		`1`
		`]`
		`}`

Continued on next page

Table 1 - continued from previous page

Command	Description	Response Example

`get_lidar_intrinsics`	Returns JSON-formatted lidar transformation matrix needed to adjust to the Sensor Coordinate Frame.	`{`
		`  "lidar_to_sensor_transform": [`
		`    -1,`
		`    0,`
		`    0,`
		`    0,`
		`    0,`
		`    -1,`
		`    0,`
		`    0,`
		`    0,`
		`    0,`
		`    1,`
		`    36.18,`
		`    0,`
		`    0,`
		`    0,`
		`    1`
		`  ]`
		`}`

Continued on next page

Table 1 - continued from previous page

Command	Description	Response Example

`get_alerts <START_CURSOR>`	Returns JSON-formatted sensor diagnostic information.
	The `log` list contains Alerts when they were activated or deactivated. An optional `START_CURSOR` argument specifies where the log should start.
	The `active` list contains all currently active alerts.
		`{`
		`  "next_cursor": 2,`
		`  "active": [`
		`    {`
		`      "category": "UDP_TRANSMISSION",`
		`      "msg": "Received an unknown error`
		`↪when trying to send lidar data UDP packet;`
		`↪closing socket.",`
		`      "realtime": "1569631015375767040",`
		`      "cursor": 0,`
		`      "id": "0x01000017",`
		`      "active": true,`
		`      "msg_verbose": "192.0.2.123:7502",`
		`      "level": "WARNING"`
		`    },`
		`  ],`
		`  "log": [`
		`    {`
		`      "category": "UDP_TRANSMISSION",`
		`      "msg": "Received an unknown error`
		`↪when trying to send lidar data UDP packet;`
		`↪closing socket.",`
		`      "realtime": "1569631015375767040",`
		`      "cursor": 0,`
		`      "id": "0x01000017",`
		`      "active": true,`
		`      "msg_verbose": "192.0.2.123:7502",`
		`      "level": "WARNING"`
		`    },`
		`    {`
		`      "category": "UDP_TRANSMISSION",`
		`      "msg": "Received an unknown error`
		`↪when trying to send IMU UDP packet; closing`
		`↪socket.",`
		`      "realtime": "1569631015883802368",`
		`      "cursor": 1,`
		`      "id": "0x0100001a",`
		`      "active": false,`
		`      "msg_verbose": "192.0.2.123:7503",`
		`      "level": "WARNING"`
		`    }`
		`  ]`
		`}`

Querying Active or Staged Parameters

Sensor configurations / operating modes can also be queried over TCP. Below is the latest command format:

`get_config_param active <parameter>` will return the current active configuration parameter values.

get_config_param staged <parameter> will return the parameter values that will take place after issuing a reinitialize command or after sensor reset.

An example session using the unix netcat utility is shown below:

```
$ nc os1-991900123456 7501
get_config_param active lidar_mode
1024x10
```

The following commands will return sensor active or staged configuration parameters:

Table2: Sensor Configurations

get_config_param	Command Description	Response
udp_ip	Returns the ip address to which the sensor sends UDP traffic.	"" (default)
udp_port_lidar	Returns the port number of lidar UDP data packets.	7502 (default)
udp_port_imu	Returns the port number of IMU UDP data packets.	7503 (default)
lidar_mode	Returns a string indicating the horizontal resolution and rotation frequency [Hz].	One of 512x10, 1024x10, 2048x10, 512x20, or 1024x20
timestamp_mode	Returns the method used to timestamp measurements. See Section 5.2 for a detailed description of each option.	One of TIME_FROM_INTERNAL_OSC, TIME_FROM_PTP_1588, TIME_FROM_SYNC_PULSE_IN
nmea_in_polarity	Returns the polarity of NMEA UART input $GPRMC messages. See Section 5.2 NMEA use case. Use ACTIVE_HIGH if UART is active high, idle low, and start bit is after a falling edge.	One of ACTIVE_HIGH (default) or ACTIVE_LOW
nmea_ignore_valid_char	Returns 0 if NMEA UART input $GPRMC messages should be ignored if valid character is not set, and 1 if messages should be used for time syncing regardless of the valid character.	0 (default) or 1
nmea_baud_rate	Returns BAUD_9600 (default) or BAUD_115200 for the expected baud rate the sensor is attempting to decode for NMEA UART input $GPRMC messages.	One of BAUD_9600, BAUD_115200
nmea_leap_seconds	Returns the number of leap seconds that will be added to the UDP timestamp when calculating seconds since 00:00:00 Thursday, 1 January 1970. For Unix Epoch time, this should be set to 0.	0 (default)

Table3: Sensor Modes

Command	Command Description	Response
multipurpose_io_mode	Returns the configured mode of the MULTIPURPOSE_IO pin. See Section 5.3 for a detailed description of each option.	One of OFF (default), INPUT_NMEA_UART, OUTPUT_FROM_INTERNAL_OSC, OUTPUT_FROM_SYNC_PULSE_IN, OUTPUT_FROM_PTP_1588, or OUTPUT_FROM_ENCODER_ANGLE
sync_pulse_out_polarity	Returns the polarity of SYNC_PULSE_OUT output, if sensor is using this for time synchronization.	One of ACTIVE_HIGH or ACTIVE_LOW (default)
sync_pulse_out_frequency	Returns the output SYNC_PULSE_OUT pulse rate in Hz.	1 (default)
sync_pulse_out_angle	Returns the output SYNC_PULSE_OUT pulse rate defined in rotation angles.	360 (default)
sync_pulse_out_pulse_width	Returns the output SYNC_PULSE_OUT pulse width in ms.	10 (ms, default)
auto_start_flag	Returns 1 if sensor is on auto start, and 0 if not. Normal operation is to use auto start. If not in auto start, the sensor must be manually commanded in order to operate.	1 (default)

Setting Configuration Parameters

set_config_param <parameter> <value> will set new values for configuration parameters, which will take effect after issuing reinitialize command, or after sensor reset.

reinitialize will reinitialize the sensor so the staged values of the parameters will take effect immediately.

write_config_txt will write new values of active parameters into a configuration file, so they will persist after sensor reset. In order to permanently change a parameter in the configuration file, first use set_config_param to update the parameter in a staging area, then use reinitialize to make that parameter active. Only after the parameter is made active will write_config_txt capture it to take effect on reset.

set_udp_dest_auto will automatically determine the sender's IP address at the time the command was sent, and set it as the destination of UDP traffic. This takes effect after issuing a reinitialize command. Using this command has the same effect as using set_config_param udp_ip <ip address>.

An example session using the unix netcat utility is shown below:

```
$ nc os1-991900123456 7501
set_config_param lidar_mode 512x20
set_config_param
set_udp_dest_auto
set_udp_dest_auto
reinitialize
reinitialize
write_config_txt
write_config_txt
```

The following commands will set sensor configuration parameters:

Table4: Setting Config Params

set_config_param	Command Description	Response
udp_ip <ip address>	Set the <ip address> to which the sensor sends UDP traffic. On boot, the sensor will not output data until this is set. If the IP address is not known, this can also be accomplished with the set_udp_dest_auto command (details above). The sensor supports unicast, IPv4 broadcast (255.255.255.255), and IPv6 multicast (ff02::01) addresses.	set_config_param on success, error: otherwise
udp_port_lidar <port>	Set the <port> on udp_ip to which lidar data will be sent (7502, default).	set_config_param on success, error: otherwise
udp_port_imu <port>	Set the <port> on udp_ip to which IMU data will be sent (7503, default).	set_config_param on success, error: otherwise

Continued on next page

Table 4 - continued from previous page

set_config_param	Command Description	Response
lidar_mode <mode>	Set the horizontal resolution and rotation rate of the sensor. Valid modes are 512x10, 1024x10, 2048x10 512x20, 1024x20. Each 50% the total number of points gathered is reduced (e.g., from 2048x10 to 1024x10) extends range by 15-20%.	set_config_param on success, error: otherwise
timestamp_mode <mode>	Set the method used to timestamp measurements. Valid modes are TIME_FROM_INTERNAL_OSC, TIME_FROM_SYNC_PULSE_IN, or TIME_FROM_PTP_1588.	set_config_param on success, error: otherwise
sync_pulse_in_polarity <1/0>	Set the polarity of SYNC_PULSE_IN input, which controls polarity of SYNC_PULSE_IN pin when timestamp_mode is set in TIME_FROM_SYNC_PULSE_IN.	set_config_param on success, error: otherwise
nmea_in_polarity <1/0>	Set the polarity of NMEA UART input $GPRMC messages. See Section 5.2 NMEA use case. Use ACTIVE_HIGH if UART is active high, idle low, and start bit is after a falling edge.	set_config_param on success, error: otherwise
nmea_ignore_valid_char <1/0>	Set 0 if NMEA UART input $GPRMC messages should be ignored if valid character is not set, and 1 if messages should be used for time syncing regardless of the valid character.	set_config_param on success, error: otherwise
nmea_baud_rate <rate in baud/s>	Set BAUD_9600 (default) or BAUD_115200 for the expected baud rate the sensor is attempting to decode for NMEA UART input $GPRMC messages.	set_config_param on success, error: otherwise
nmea_leap_seconds <s>	Set an integer number of leap seconds that will be added to the UDP timestamp when calculating seconds since 00:00:00 Thursday, 1 January 1970. For Unix Epoch time, this should be set to 0.	set_config_param on success, error: otherwise
multipurpose_io_mode <mode>	Configure the mode of the MULTIPURPOSE_IO pin. Valid modes are OFF, INPUT_NMEA_UART, OUTPUT_FROM_INTERNAL_OSC, OUTPUT_FROM_SYNC_PULSE_IN, OUTPUT_FROM_PTP_1588, or OUTPUT_FROM_ENCODER_ANGLE.	set_config_param on success, error: otherwise

Table5: Setting Sync

set_config_param	Command Description	Response
sync_pulse_out_polarity <1/0>	Set the polarity of SYNC_PULSE_OUT output, if sensor is set as the master sensor used for time synchronization.	set_config_param on success, error: otherwise
sync_pulse_out_frequency <rate in Hz>	Set output SYNC_PULSE_OUT rate. Valid inputs are integers > 0 Hz, but also limited by the criteria described in Section 5.3 of this user manual.	set_config_param on success, error: otherwise
sync_pulse_out_angle <angle in deg>	Set output SYNC_PULSE_OUT rate defined by rotation angle. Valid inputs are integers up to 360 degrees but also limited by the criteria described in Section 5.3 of this user manual.	set_config_param on success, error: otherwise
sync_pulse_out_pulse_width <width in ms>	Set output SYNC_PULSE_OUT pulse width in ms, in 1 ms increments. Valid inputs are integers greater than 0 ms, but also limited by the criteria described in Section 5.3 of this user manual.	set_config_param on success, error: otherwise

Table6: Reinitialize, Write Configuration, & Auto Destination UDP

Command	Command Description	Response
reinitialize	Restarts the sensor. Changes to lidar, multipurpose_io, and timestamp modes will only take effect after reinitialization.	reinitialize on success
write_config_txt	Make the current settings persist after reboot.	write_config_txt on success
set_udp_dest_auto	Set the destination of UDP traffic to the IP address that issued the command.	set_udp_dest_auto on success

3.4 Lidar Data Format

By default UDP data is forwarded to Port `7502`. Lidar data packets consist of 16 azimuth blocks and are always 12608 Bytes in length. The packet rate is dependent on the output mode. Words are 32 bits in length and little endian.

Word	Azimuth Block 0	Azimuth Block 1	...	Azimuth Block 15
(Word 0,1)	Timestamp	Timestamp	...	Timestamp
(Word 2[0:15])	Measurement ID	Measurement ID	...	Measurement ID
(Word 2[16:31])	Frame ID	Frame ID	...	Frame ID
(Word 3)	Encoder Count	Encoder Count	...	Encoder Count
(Word 4,5,6)	Channel 0 Data Block	Channel 0 Data Block	...	Channel 0 Data Block
(Word 7,8,9)	Channel 1 Data Block	Channel 1 Data Block	...	Channel 1 Data Block
	.	.		.
(Word 193, 194, 195)	Channel 63 Data Block	Channel 63 Data Block	...	Channel 63 Data Block
(Word 196)	Azimuth Data Block Status	Azimuth Data Block Status	...	Azimuth Data Block Status

Each azimuth block contains:

- Timestamp [64 bit unsigned int] - timestamp of the measurement in nanoseconds
- Measurement ID [16 bit unsigned int] - a sequentially incrementing azimuth measurement counting up from 0 to 511, or 0 to 1023, or 0 to 2047 depending on lidar_mode.
- Frame ID [16 bit unsigned int] - index of the lidar scan. Increments every time the sensor completes a rotation, crossing the zero point of the encoder.
- Encoder Count [32 bit unsigned int] - an azimuth angle as a raw encoder count, starting from 0 with a max value of 90111 - incrementing 44 ticks every azimuth angle in 2048 mode, 88 ticks in 1024 mode, and 176 ticks in 512 mode.
- Data Block [96 bits] - 3 data words for each of the 16 or 64 pixels. See Table below for full definition.
  - Range [32 bit unsigned int - only 20 bits used] - range in millimeters, discretized to the nearest 3 millimeters.
  - Signal Photons [16 bit unsigned int] - signal intensity photons in the signal return measurement are reported.
  - Reflectivity [16 bit unsigned int] - sensor signal_photon measurements are scaled based on measured range and sensor sensitivity at that range, providing an indication of target reflectivity. Calibration of this measurement has not currently been rigorously implemented, but this will be updated in a future firmware release.
  - Ambient Noise Photons [16 bit unsigned int] - ambient noise photons in the ambient noise return measurement are reported.
- Azimuth Data Block Status [32 bits]- indicates whether the azimuth block contains valid data in its channels' Data Blocks. Good = 0xFFFFFFFF, Bad = 0x0. If the Azimuth Data Block Status is bad (e.g. in the case of column data being dropped), words in the data block will be set to 0x0, but Timestamp, Measurement ID, Frame ID, and Encoder Count will remain valid.

Full Description of Data Block:

Table7: Data Block

Word	Byte 3	Byte 2	Byte 1	Byte 0
(Word 0)	unused[31:24]	unused[23:20] range_mm[19:16]	range_mm[15:8]	range_mm[7:0]
(Word 1)	signal_photons[31:24]	signal_photons[23:16]	reflectivity[15:8]	reflectivity[7:0]
(Word 2)	unused[31:24]	unused[23:16]	noise_photons[15:8]	noise_photons[7:0]

3.5 IMU Data Format

IMU UDP Packets are 48 Bytes long and are sent to Port `7503` at 100 Hz. Values are little endian.

Word	IMU and Gyro Data Block
(Word 0,1)	64-bit unsigned integer for IMU diagnostic time (ns, monotonic system time since boot)
(Word 2,3)	64-bit unsigned integer for accelerometer read time (ns, relative to *timestamp_mode*)
(Word 4,5)	64-bit unsigned integer for gyroscope read time (ns, relative to *timestamp_mode*)
(Word 6)	32-bit float for acceleration in x-axis (g)
(Word 7)	32-bit float for acceleration in y-axis (g)
(Word 8)	32-bit float for acceleration in z-axis (g)
(Word 9)	32-bit float for angular velocity about in x-axis (deg per sec)
(Word 10)	32-bit float for angular velocity about in y-axis (deg per sec)
(Word 11)	32-bit float for angular velocity about in z-axis (deg per sec)

Note that the first timestamp (Word 0,1) is for diagnostics only and is rarely used under normal operation.

The second two timestamps, (Word 2,3) and (Word 4,5), are sampled on the same clock as the lidar data, so should be used for most applications.

Ouster provides timestamps for both the gyro and accelerometer in order to give access to the lowest level information. In most applications it is acceptable to use the average of the two timestamps.

3.6 Data Rates

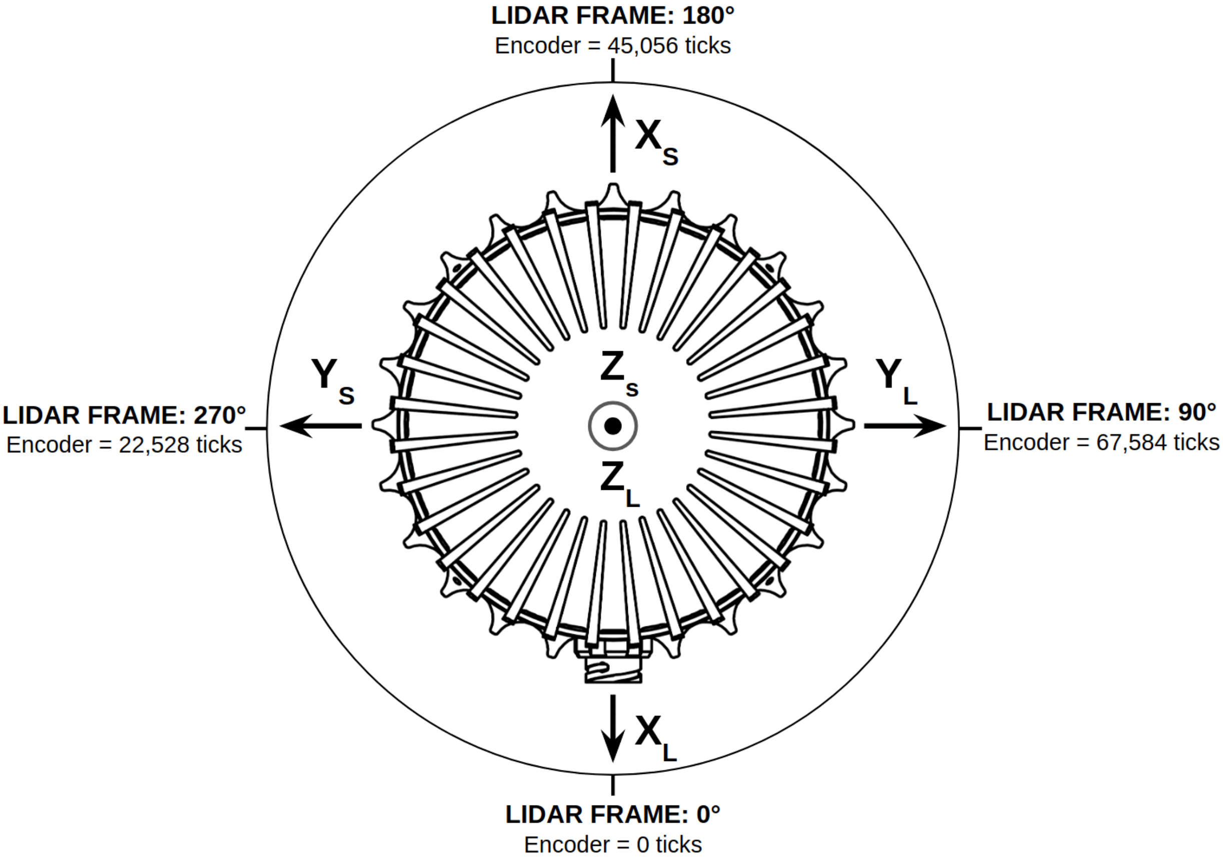
Considering the lidar packets which account for 99.6% of data coming from the sensor (IMU packets constitute the remaining 0.4%): Based on 12,608 Bytes/packet and 1280 packets/sec, in `2048x10` or `1024x20` mode the OS1 outputs 16.138 MB/s (129 Mbps). For this reason a gigabit Ethernet network is required for reliable performance.

4 Coordinate Frames

4.1 Sensor Coordinate Frame

The Sensor Coordinate Frame follows the right-hand rule convention and is defined at the center of the sensor housing on the bottom, with the x-axis pointed forward, y-axis pointed to the left and z-axis pointed towards the top of the sensor. The external connector is located in the negative x direction. The Sensor Coordinate Frame is marked in the diagram below with X_S, Y_S, Z_S.

The Lidar Coordinate Frame follows the right-hand rule convention and is defined at the intersection of the lidar axis of rotation and the lidar optical midplane (a plane parallel to Sensor Coordinate Frame XY plane and coincident with the 0 degree elevation beam angle of the lidar). The Lidar Coordinate Frame axes are arranged with the x-axis pointed at encoder angle 0, the y-axis pointed to the right and the z-axis pointed towards the top of the sensor. The external connector is located in the positive x direction. The Lidar Coordinate Frame is marked in the diagram below with X_L, Y_L, Z_L.



The Lidar Coordinate Frame's positive x-axis (0 encoder value) is opposite the Sensor Coordinate Frame's positive x-axis to center lidar data about the Sensor Coordinate Frame's positive x-axis. A single measurement frame starts at the Lidar Coordinate Frame's 0 degree position and ends at the 360 degree position. This is convenient when viewing a "range image" of the Ouster Sensor measurements, allowing the "range image" to be centered in the Sensor Coordinate Frame's positive x-axis, which is generally forward facing in most robotic systems.

The Ouster Sensor scans in the clockwise direction when viewed from the top, which is a negative rotational velocity about the z-axis. Thus, as encoder ticks increases from 0 to 90111, the actual angle about the z-axis in the Lidar Coordinate Frame will decrease.

4.2 Lidar Intrinsic Beam Angles

The intrinsic beam angles for each beam may be queried with a TCP command (see OS1 Software User Guide) and provide an azimuth and elevation adjustment to the each beam. The azimuth adjustment is referenced off of the current encoder angle and the elevation adjustment is referenced from the XY plane in the Sensor and lidar Coordinate Frames.

4.3 Lidar Range Data To XYZ Lidar Coordinate Frame

The origin and axes of the Lidar Coordinate Frame are defined by the position of the lidar lens aperture stop in the sensor and the 0º position of the rotary encoder, which is aligned with the sensor connector and the negative X axis of the Sensor Coordinate Frame.

For many applications, it is sufficient to calculate the XYZ point cloud in the Lidar Coordinate Frame using a combination of the intrinsic beam angles and the encoder reading. The intrinsic azimuth and elevation beam angles may be queried over TCP as two vectors each 64 elements long.

Lidar data may be transformed into 3D cartesian x, y, z coordinates in the Lidar Coordinate Frame. Given:

- **encoder_count** of the azimuth block
- **range** from the data block of the *i*-th channel
- **beam_altitude_angles** and
- **beam_azimuth_angles** from **get_beam_intrinsics** in the TCP interface described in Section 3.3

the corresponding 3D point can be computed using:

$$
\begin{aligned}
r &= range \text{ mm} \\
\theta &= 2\pi\left(\frac{encoder\_count}{90112} + \frac{beam\_azimuth\_angles[i]}{360}\right) \\
\phi &= 2\pi\frac{beam\_altitude\_angles[i]}{360} \\
x &= r\cos(\theta)\cos(\phi) \\
y &= -r\sin(\theta)\cos(\phi) \\
z &= r\sin(\phi)
\end{aligned}
$$

4.4 Lidar Range Data To Sensor XYZ Coordinate Frame

For applications that require calibration against a precision mount or use the IMU data in combination with the lidar data, the xyz points should be adjusted to the Sensor Coordinate Frame. This requires a z translation and a rotation of the x,y,z points about the z axis. The z translation is the height of the lidar aperture stop above the sensor origin, which is 36.180 mm, and the data must be rotated 180 degrees around the z axis. This information can be queried over TCP in the form of an intrinsic transformation matrix:

M_lidar_to_sensor = [[X, X, X, X], [X, X, X, X], [X, X, X, X], [0, 0, 0, 1]]

Example JSON formatted query using the TCP command `get_lidar_intrinsics`:

```
{
  "lidar_to_sensor_transform": [
    -1,
    0,
    0,
    0,
    0,
    -1,
    0,
    0,
    0,
    0,
    1,
    36.18,
    0,
    0,
    0,
    1
    ]
 }
```

4.5 IMU Data To Sensor XYZ Coordinate Frame

The IMU is slightly offset in the Sensor Coordinate Frame for practical reasons. The IMU origin in the Sensor Coordinate Frame can be queried over TCP in the form of an intrinsic transformation matrix:

M_imu_to_sensor = [[X, X, X, X], [X, X, X, X], [X, X, X, X], [0, 0, 0, 1]]

Example JSON formatted query using the TCP command `get_imu_intrinsics`:

```
{
  "imu_to_sensor_transform": [
    1,
    0,
    0,
    6.253,
    0,
    1,
```

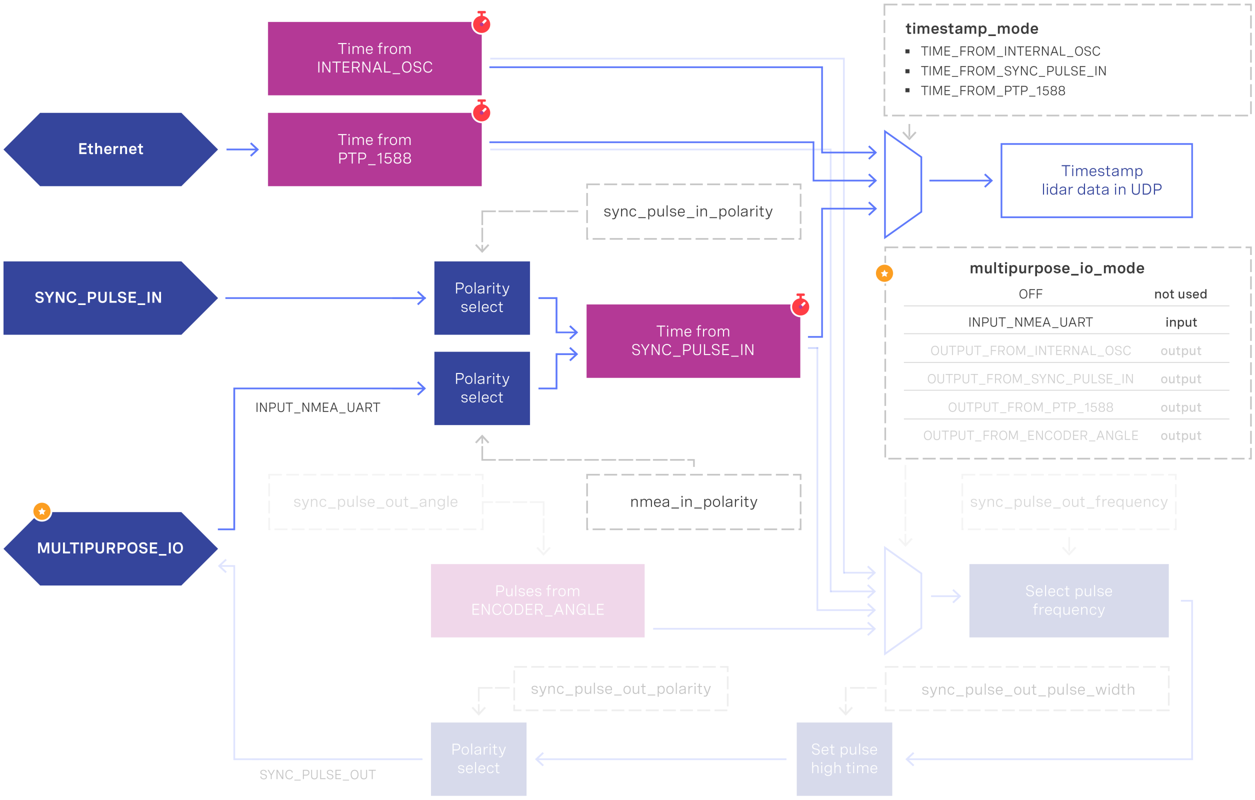
(continues on next page)

(continued from previous page)

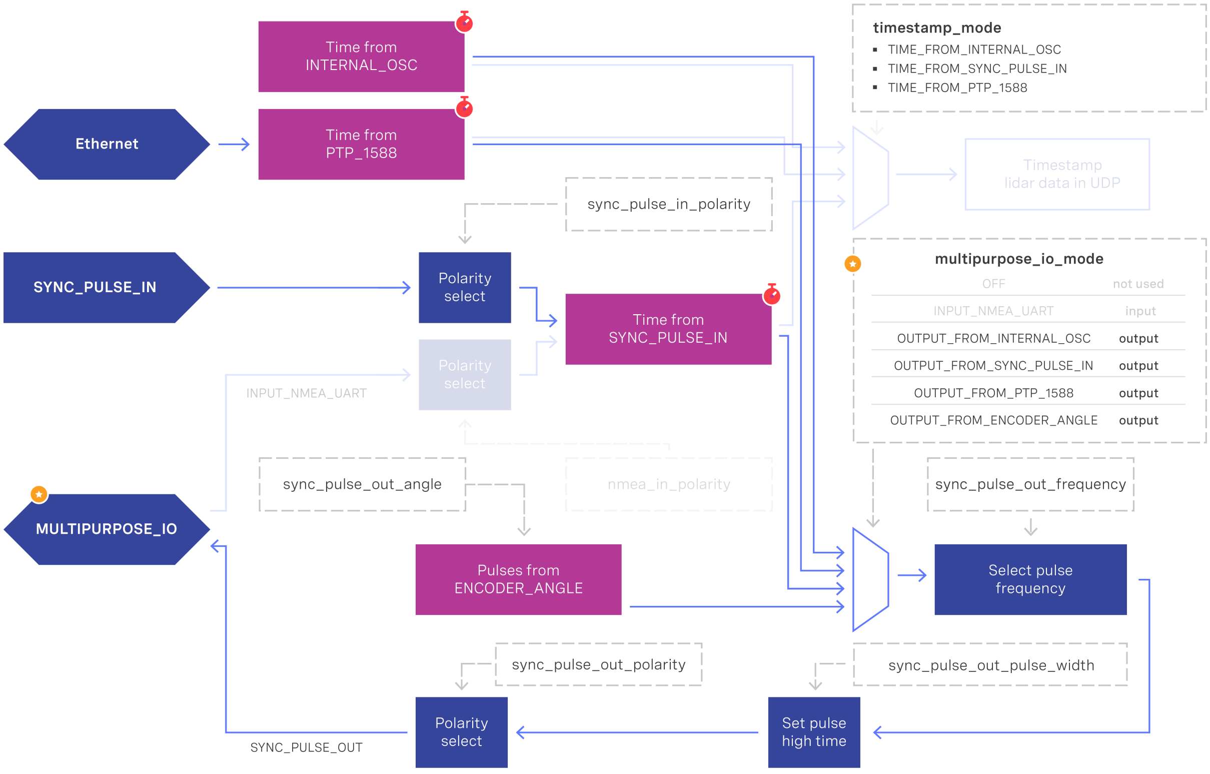
```
    0,
    -11.775,
    0,
    0,
    1,
    7.645,
    0,
    0,
    0,
    1
    ]
}
```

5 Time Synchronization

5.1 Timing Overview Diagram



Signal path with MULTIPURPOSE_IO set as input



Signal path with MULTIPURPOSE_IO set as output

5.2 Sensor Time Source

- All lidar and IMU data are timestamped to a common timer with 10 nanosecond precision.
- The common timer can be programmed to run off one of three clock sources:
  - An internal clock derived from a high accuracy, low drift oscillator.
  - An opto-isolated digital input from the external connector for timing off an external hardware trigger such as a GPS. The polarity of this input signal is programmable. For instance, both a GPS PPS pulse and a 30 Hz frame sync from an industrial camera can supply a timing signal to the OS1.
  - Using the IEEE 1588 Precision Time Protocol. PTP provides the convenience of configuring timing over a network that supports IEEE 1588 with no additional hardware signals.

Setting Ouster Sensor Time Source

The source for measurement timestamps can be configured using the `set_timestamp_mode` TCP command (see Section 3.3). The available modes are described below:

Table8: Timestamp Modes

Command	Response
TIME_FROM_INTERNAL_OSC	Use the internal clock. Measurements are time stamped with ns since power-on. Free running counter based on the OS1's internal oscillator. Counts seconds and nanoseconds since OS1 turn on, reported at ns resolution (both a second and nanosecond register in every UDP packet), but min increment is on the order of 10 ns. Accuracy is +/- 90 ppm.
TIME_FROM_SYNC_PULSE_IN	A free running counter synced to the SYNC_PULSE_IN input counts seconds (# of pulses) and nanoseconds since OS1 turn on. If multipurpose_io_mode is set to INPUT_NMEA_UART then the seconds register jumps to time extracted from a NMEA $GPRMC message read on the multipurpose_io port. Reported at ns resolution (both a second and nanosecond register in every UDP packet), but min increment is on the order of 10 ns. Accuracy is +/- 1 s from a perfect SYNC_PULSE_IN source.
TIME_FROM_PTP_1588	Synchronize with an external PTP master. A monotonically increasing counter that will begin counting seconds and nanoseconds since startup. As soon as a 1588 sync event happens, the time will be updated to seconds and nanoseconds since 1970. The counter must always count forward in time. If another 1588 sync event happens the counter will either jump forward to match the new time, or slow itself down. It is reported at ns resolution (there is both a second and nanosecond register in every UDP packet), but the minimum increment varies. Accuracy is +/- <50 us from the 1588 master.

If configuring the sensor to synchronize time from an external sync pulse, the pulse polarity can be specified as described in Section 3.3. Pulse-in frequency is assumed to be 1 Hz. For example, the below commands will set the sensor to expect an active low pulse and configure the seconds timetamp to be pulse count since sensor startup:

- set_config_param timestamp_mode TIME_FROM_SYNC_PULSE_IN
- set_config_param sync_pulse_in_polarity ACTIVE_LOW
- reinitialize

If desired to configure the multipurpose-io port of the sensor to accept an external NMEA UART message, the multipurpose_io_mode parameter must be set to INPUT_NMEA_UART as described in Section 5.3. Once a valid UART message is recieved by the sensor, the seconds timetamp will snap to the latest timestamp recieved. The expected NMEA UART message is configurable as described in Section 3.3. For example, the below commands will set the sensor to accept an NMEA UART message that is active high with a baud rate of 115200 bits per second, add 27 additional leap seconds, and accept messages even with a valid character not set:

- set_config_param multipurpose_io_mode INPUT_NMEA_UART
- set_config_param nmea_in_polarity ACTIVE_HIGH
- set_config_param nmea_baud_rate BAUD_115200
- set_config_param nmea_leap_seconds 27

- `set_config_param nmea_ignore_valid_char 1`
- `reinitialize`

5.3 External Trigger Clock Source

Additionally, the OS1 can be configured to output a SYNC_PULSE_OUT signal from a variety of sources. See example commands in Section 3.3. Pulses will always be evenly spaced.

This can be enabled through the `multipurpose_io_mode` configuration parameter.

Configuration	Response
`OFF`	Do not output a SYNC_PULSE_OUT signal.
`INPUT_NMEA_UART`	Reconfigures the MULTIPURPOSE_IO port as an input. See Section 5.2 for more information.
`OUTPUT_FROM_INTERNAL_OSC`	Output a SYNC_PULSE_OUT signal synchronized with the internal clock.
`OUTPUT_FROM_SYNC_PULSE_IN`	Output a SYNC_PULSE_OUT signal synchronized with a SYNC_PULSE_IN provided to the unit.
`OUTPUT_FROM_PTP_1588`	Output a SYNC_PULSE_OUT signal synchronized with an external PTP IEEE 1588 master.
`OUTPUT_FROM_ENCODER_ANGLE`	Output a SYNC_PULSE_OUT signal with a user defined rate in an integer number of degrees.

When the sensor's `multipurpose_io_mode` is set to `OUTPUT_FROM_INTERNAL_OSC`, `OUTPUT_FROM_SYNC_PULSE_IN`, or `OUTPUT_FROM_PTP_1588`, then `sync_pulse_out_frequency` (Hz) parameter can be used to define the output rate. It defaults to 1 Hz. It should be greater than 0 Hz and maximum `sync_pulse_out_frequency` is limited by the criterion below.

When the sensor is set to `OUTPUT_FROM_ENCODER_ANGLE`, then the `sync_pulse_out_angle` (deg) parameter can be used to define the output pulse rate. This allows the user to output a SYNC_PULSE_OUT signal when the encoder passes a specified angle, or multiple of the angle, indexed from 0 crossing, in degrees. It should be an integer between 0 and 360 degrees, inclusive. However, the minimum `sync_pulse_out_angle` is also limited by the criterion below.

In all modes, the output pulse width is defined by `sync_pulse_out_pulse_width` (ms).

Note: If `sync_pulse_out_pulse_width` x `sync_pulse_out_frequency` is close to 1 second, the output pulses will not function (will not return to 0). For example, at 10 Hz rotation and a 10 ms pulse width, the limitation on the number of pulses per rotation is 9.

EXAMPLE COMMANDS: Here are example commands and their effect on output pulse when `lidar_mode` is `1024x10`, and assuming `sync_pulse_out_pulse_width` is 10 ms.

→

Command	Response
`set_config_param multipurpose_io_mode OUTPUT_FROM_SYNC_PULSE_IN set_config_param sync_pulse_out_pulse_width 10 set_config_param sync_pulse_out_frequency 1 reinitialize`	The output pulse frequency is 1 Hz. Each pulse is 10 ms wide. `sync_pulse_out_pulse_width` and `sync_pulse_out_frequency` commands are optional because they just re-command the default values
`set_config_param multipurpose_io_mode OUTPUT_FROM_SYNC_PULSE_IN set_config_param sync_pulse_out_frequency 50 reinitialize`	The output pulse frequency is 50 Hz. Each pulse is 10 ms wide.
`set_config_param multipurpose_io_mode OUTPUT_FROM_ENCODER_ANGLE set_config_param sync_pulse_out_angle 360 reinitialize`	The output pulse frequency is 10 Hz, since the sensor is in 10 Hz mode (10 rotations per second) and the angle is set to 360 degrees, a full rotation. Each pulse is 10 ms wide.
`set_config_param multipurpose_io_mode OUTPUT_FROM_ENCODER_ANGLE set_config_param sync_pulse_out_angle 45 reinitialize`	The output pulse frequency is 80 Hz, since the sensor is in 10 Hz mode (10 rotations per second) and the angle is set to 45 degrees. Each full rotation will have 8 pulses. Each pulse is 10 ms wide.

5.4 NMEA Message Format

The Ouster Sensor expects a standard NMEA $GPRMC UART message. Data (called a sentence) is a simple ASCII string starting with a '$' character and ending with a return character. Fields of the sentence are separated with a ',' character, and the last field (a checksum) is separated by a '*' character.

The max character length of a standard message is 80 characters; however, the Ouster Sensor can support non-standard messages up to 85 characters (see Example 2 below).

The Ouster Sensor will deliver time in the UDP packet by calculating seconds since 00:00:00 Thursday, 1 January 1970. `nmea_leap_seconds` by default is 0, meaning this calculation will not take into account any leap seconds. If `nmea_leap_seconds` is 0 then the reported time is Unix Epoch time. As of February, 2019 Coordinated Universal Time (UTC) lags behind International Atomic Time (TAI) by an offset of 37 seconds (10 seconds from the initial UTC offset when UTC was introduced in 1972 + 27 leap seconds announced in the intervening years). Therefore, setting `nmea_leap_seconds` to 37 in February of 2019 would make the timestamps match the TAI standard.

`nmea_in_polarity` by default is `ACTIVE_HIGH`. This means that a UART start bit will occur directly after a falling edge. If using RS-232, the UART signal may be inverted (where a start bit occurs directly after a rising edge). In this case, `nmea_in_polarity` should be set to `ACTIVE_LOW`.

Example 1 Message:

```
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A
```

→

Field	Description
$GPRMC	Recommended Minimum sentence C
123519	Fix taken at 12:35:19 UTC
A	Status A=active or V=Void
4807.038	Latitude 48 deg 07.038'
N	Latitude cardinal reference
01131.000	Longitude 11 deg 31.000'
E	Longitude cardinal reference
022.4	Speed over the ground in knots
084.4	Track angle in degrees True
230394	Date - 23rd of March 1994
003.1	Magnetic Variation
W	Magnetic cardinal reference
A	[Optional] A=autonomous, D=differential, E=Estimated, N=not valid, S=Simulator
*6A	The checksum data, always begins with *

Example 2 Message:

```
$GPRMC,042901.00,A,3745.871698,N,12224.825960,W,0.874,327.72,130219,13.39,E,A,V*60
→
```

Field	Description
$GPRMC	Recommended Minimum sentence C
042901.00	Fix taken at 4:29:01 UTC
A	Status A=active or V=Void
3745.871698	Latitude 37 deg 45.871698'
N	Latitude cardinal reference
12224.825960	Longitude 12 deg 24.825960'
W	Longitude cardinal reference
0.874	Speed over the ground in knots
327.72	Track angle in degrees True
130219	Date - 13th of February 2019
13.39	Magnetic Variation
E	Magnetic cardinal reference
A	[Optional] A=autonomous, D=differential, E=Estimated, N=not valid, S=Simulator
*60	The checksum data, always begins with *

6 Updating Firmware

Sensor firmware can be updated with an Ouster-provided firmware file from www.ouster.com/resources (or directly from the deployment engineering team) by accessing the sensor over http - e.g., http://os1-991900123456.local/ and uploading the file as prompted.

Always check your firmware version before attempting an update. Only update to a equal or higher version number. Do not `roll back` firmware to lower numbered versions without having been instructed to do so by Ouster deployment engineering.

7 Troubleshooting

Starting from firmware v1.11, the sensor HTTP server page http://os1-991900123456.local/ has `Dashboard`, `Diagnostics`, `Documentation` and `Reset Configuration` buttons:

- `Dashboard`: Current page that lists some basic sensor information, and allows sensor firmware upgrade.
- `Diagnostics`: Diagnostic information and system journal that can be downloaded and included when contacting Ouster for service.
- `Documentation`: Sensor User Guide
- `Reset Configuration`: Sensor factory configuration that can be reset to if desired. This will erase any custom configuration that you set on the sensor previously.

Many initial problmes with the OS1 are associated with the sensor not properly being assigned an IP address by a network switch or DHCP server on a client computer. Check your networking settings, the steps in Section 3, and that all wires are firmly connected if you suspect this problem. Note that if the sensor is not connected via gigabit Ethernet, it will stop sending data and will output an error code if it fails to achieve a 1000 Mb/s+ full duplex link.

To check for hardware errors, use the `get_alerts` command as described in Section 3.3.

If the watchdog is triggered (when various temperature limits are exceeded or uplink/downlink has failed), an alert code will be appended to the end of TCP command `get_alerts`. The sensor has a limited-size buffer that will record the first few alerts detected by the sensor.

The alerts reported have the following format:

```
{
   "category": "Category of the alert: e.g. OVERTEMP, UDP_TRANSMISSION",
   "level": "Level of alert: e.g. NOTICE, WARNING, ERROR",
   "realtime": "The timestamp of the alert in nanoseconds",
   "active": "Whether the alert is active or not: <true/false>",
   "msg": "A description of the alert",
   "cursor": "The sequential number of the alert, starting from 0 counting up",
   "id": "The hexadecimal identification code of the alert: e.g. 0x01000017",
   "msg_verbose": "Any additional verbose description that the alert may present"
}
```

Example showing active and logged forced temperature sensor failures occuring at timestamps 1569712873477772800, 1569712879991844096, 1569712884968876544 (nanoseconds). The first logged error then resolves itself at 1569713260229536000. The example has been JSON formatted:

```
{
   "active": [
      {
         "category": "OVERTEMP",
         "level": "ERROR",
         "realtime": "1569712879991844096",
         "active": true,
```

(continues on next page)

(continued from previous page)

```
         "msg": "Unit internal temperature out of bounds; please ensure proper heat sinking.",
         "cursor": 1,
         "id": "0x01000001",
         "msg_verbose": ""
      },
      {
         "category": "OVERTEMP",
         "level": "ERROR",
         "realtime": "1569712884968876544",
         "active": true,
         "msg": "Unit internal temperature out of bounds; please ensure proper heat sinking.",
         "cursor": 2,
         "id": "0x01000002",
         "msg_verbose": ""
      }
   ],
   "next_cursor": 4,
   "log": [
      {
         "category": "OVERTEMP",
         "level": "ERROR",
         "realtime": "1569712873477772800",
         "active": true,
         "msg": "Unit internal temperature out of bounds; please ensure proper heat sinking.",
         "cursor": 0,
         "id": "0x01000000",
         "msg_verbose": ""
      },
      {
         "category": "OVERTEMP",
         "level": "ERROR",
         "realtime": "1569712879991844096",
         "active": true,
         "msg": "Unit internal temperature out of bounds; please ensure proper heat sinking.",
         "cursor": 1,
         "id": "0x01000001",
         "msg_verbose": ""
      },
      {
         "category": "OVERTEMP",
         "level": "ERROR",
         "realtime": "1569712884968876544",
         "active": true,
         "msg": "Unit internal temperature out of bounds; please ensure proper heat sinking.",
         "cursor":2 ,
         "id": "0x01000002",
         "msg_verbose": ""
      },
      {
         "category": "OVERTEMP",
         "level": "ERROR",
         "realtime": "1569713260229536000",
         "active": false,
         "msg": "Unit internal temperature out of bounds; please ensure proper heat sinking.",
```

(continues on next page)

(continued from previous page)

```
         "cursor": 3,
         "id": "0x01000000",
         "msg_verbose": ""
      }
   ]
}
```

→

Version v1.6.0

Date 2018-08-16

Description

- Add `get_sensor_info` command gives `prod_line` info.

→

Version v1.7.0

Date 2018-09-05

Description

- No TCP command change.

→

Version v1.8.0

Date 2018-10-11

Description

- `get_sensor_info` command gives `INITIALIZING`, `UPDATING`, `RUNNING`, `ERROR` and `UNCONFIGURED` status.

→

Version v1.9.0

Date 2018-10-24

Description

- No TCP command change.

→

Version v1.10.0

Date 2018-12-11

Description

- Remove all references of `pulse_mode`.
- Add `get_alerts`, `pps_rate` and `pps_angle` usage commands and expected output.
- Remove TCP commands prior to v1.5.1.

→

Version v1.11.0

Date 2019-03-25

Description

- Add section on HTTP API commands.
- TCP Port now hardcoded to 7501; port is no longer configurable.
- Update to SYNC_PULSE_IN and MULTIPURPOSE_IO interface and configuration parameters (see details below).

Details on interface changes:

Configuration parameters name changes:

- `pps_in_polarity` changed to `sync_pulse_in_polarity`
- `pps_out_mode` changed to `multipurpose_io_mode`
- `pps_out_polarity` changed to `sync_pulse_out_polarity`
- `pps_rate` changed to `sync_pulse_out_frequency`
- `pps_angle` changed to `sync_pulse_out_angle`
- `pps_pulse_width` changed to `sync_pulse_out_pulse_width`

New configuration parameters:

- `nmea_in_polarity`
- `nmea_ignore_valid_char`
- `nmea_baud_rate`
- `nmea_leap_seconds`

Configuration parameters option changes:

- **timestamp_mode**
  - `TIME_FROM_PPS` changed to `TIME_FROM_SYNC_PULSE_IN`
- **multipurpose_io_mode (formerly pps_out_mode)**
  - `OUTPUT_PPS_OFF` changed to `OFF`
  - `OUTPUT_FROM_PPS_IN_SYNCED` changed to `OUTPUT_FROM_SYNC_PULSE_IN`
  - Removed `OUTPUT_FROM_PPS_DEFINED_RATE`
  - Added `INPUT_NMEA_UART`

TCP command changes:

- **Added commands:**
  - `get_time_info`
- **Changed commands:**
  - `get_config_txt` (returned dictionary keys match parameter changes)
- **Removed commands:**
  - `set_pps_in_polarity`
  - `get_pps_out_mode`
  - `set_pps_out_mode`
  - `get_timestamp_mode`
  - `set_timestamp_mode`

Polarity changes:

- `sync_pulse_in_polarity` was corrected to match parameter naming.
- `sync_pulse_out_polarity` was corrected to match parameter naming.

→

Version v1.12.0

Date

Description

- Corrected IMU axis directions to match sensor coordinate frame. See section Section 4.1 for details on sensor coordinate frame. This change inverts IMU X, Y, and Z axis relative to v1.11.0.

→

Version v1.13.0

Date

Description

- Add TCP command `get_udp_dest_auto`
- TCP command `get_alerts`, includes more descriptive errors for troubleshooting
- Packet Status now called Azimuth Data Block Status and is calculated differently
- Packets with bad CRC are now dropped upstream and replaced with `0` padded packets to ensure all packets are sent for each frame.
- Return format of TCP command `get_time_info` updated.
- Removed reference to window_rejection_enable

8 HTTP API Reference

HTTP API developer reference guide. This documents the interface for HTTP API and is accessible via `/api/v1` on the sensor hosted HTTP server.

API Resources

- `system/firmware`
- `system/network`
- `system/time`

8.1 `system/firmware`

`GET system/firmware`

```
GET /api/v1/system/firmware HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
Host: 192.0.2.123
content-type: application/json; charset=UTF-8

{
  "fw": "ousteros-image-prod-aries-v1.11.0"
}
```

→

Response JSON Object

- `fw` (`string`) - Running firmware image name and version.

Status Codes

- 200 OK - No error

8.2 `system/network`

`GET system/network`

Get the system network configuration.

```
GET /api/v1/system/network HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8

{
   "carrier": true,
   "duplex": "full",
   "ethaddr": "bc:0f:a7:00:01:2c",
   "hostname": "os1-991900123456",
   "ipv4": {
      "addr": "192.0.2.123/24",
      "link_local": "169.254.245.183/16",
      "override": null
   },
   "ipv6": {
      "link_local": "fe80::be0f:a7ff:fe00:12c/64"
   },
   "speed": 1000
}
```

→

Response JSON Object

- `carrier` (`boolean`) - State of Ethernet link, `true` when physical layer is connected.
- `duplex` (`string`) - Duplex mode of Ethernet link, `half` or `full`.
- `ethaddr` (`string`) - Ethernet hardware (MAC) address.
- `hostname` (`string`) - Hostname of the sensor, also used when requesting *DHCP* lease.
- `ipv4` (`object`) - See *ipv4 object*
- `ipv6.link_local` (`string`) - Link-local IPv6 address.
- `speed` (`integer`) - Ethernet physical layer speed in Mbps, should be 1000 Mbps.

Status Codes

- 200 OK - No error

`GET system/network/ipv4`

Get the IPv4 network configuration.

```
GET /api/v1/system/network/ipv4 HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8

{
  "addr": "192.0.2.123/23",
  "link_local": "169.254.245.183/16",
  "override": null
}
```

→

Response JSON Object

- `addr` (`string`) - Current global or private IPv4 address.
- `link_local` (`string`) - Link-local IPv4 address.
- `override` (`string`) - Static IP override value, this should match `addr`. This value will be `null` when unset and operating in *DHCP* mode.

Status Codes

- 200 OK - No error

`GET system/network/ipv4/override`

Get the current IPv4 static IP address override.

```
GET /api/v1/system/network/ipv4/override HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8

null
```

→

Response JSON Object

- `string` - Static IP override value, this should match `addr`. This value will be `null` when unset and operating in *DHCP* mode.

Status Codes

- 200 OK - No error

`PUT system/network/ipv4/override`

Override the default dynamic behavior and set a static IP address.

Note: The sensor will reset the network configuration after a short sub second delay (to allow for the HTTP response to be sent). After this delay the sensor will only be reachable on the newly set IPv4 address.

The sensor needs to be reachable either by dynamic *DHCP* configuration or by an existing static IP override from the host reconfiguring the sensor.

Warning: If an unreachable network address is set, the sensor will become unreachable.

Static IP override should only be used in special use cases. The dynamic *DHCP* configuration is recommended where possible.

```
PUT /api/v1/system/network/ipv4/override HTTP/1.1
Content-Type: application/json
Host: 192.0.2.123

"192.0.2.100/24"
```

→

Request JSON Object

- `string` - Static IP override value with subnet mask

Response JSON Object

- `string` - Static IP override value that system will set after a short delay.

Status Codes

- 200 OK - No error

`DELETE system/network/ipv4/override`

Delete the static IP override value and return to dynamic configuration.

Note: The sensor will reset the network configuration after a short sub second delay (to allow for the HTTP response to be sent). After this delay the sensor will only be reachable on the newly set IPv4 address.

The sensor may be unreachable for several seconds while a *DHCP* lease is obtained from a network *DHCP* server.

```
DELETE /api/v1/system/network/ipv4/override HTTP/1.1
Host: 192.0.2.123
```

→

Status Codes

- 204 No Content - No error, no content

8.3 system/time

GET system/time

Get the system time configuration for all timing components of the sensor.

```
GET /api/v1/system/time HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8

{
  "ptp": {
    "current_data_set": {
      "mean_path_delay": 37950,
      "offset_from_master": -211488,
      "steps_removed": 1
    },
    "parent_data_set": {
      "gm_clock_accuracy": 33,
      "gm_clock_class": 6,
      "gm_offset_scaled_log_variance": 20061,
      "grandmaster_identity": "001747.fffe.700038",
      "grandmaster_priority1": 128,
      "grandmaster_priority2": 128,
      "observed_parent_clock_phase_change_rate": 2147483647,
      "observed_parent_offset_scaled_log_variance": 65535,
      "parent_port_identity": "001747.fffe.700038-1",
      "parent_stats": 0
    },
    "port_data_set": {
      "announce_receipt_timeout": 3,
      "delay_mechanism": 1,
      "log_announce_interval": 1,
      "log_min_delay_req_interval": 0,
      "log_min_pdelay_req_interval": 0,
```

(continues on next page)

(continued from previous page)

```
      "log_sync_interval": 0,
      "peer_mean_path_delay": 0,
      "port_identity": "bc0fa7.fffe.00012c-1",
      "port_state": "SLAVE",
      "version_number": 2
    },
    "time_properties_data_set": {
      "current_utc_offset": 37,
      "current_utc_offset_valid": 1,
      "frequency_traceable": 1,
      "leap59": 0,
      "leap61": 0,
      "ptp_timescale": 1,
      "time_source": 32,
      "time_traceable": 1
    },
    "time_status_np": {
      "cumulative_scaled_rate_offset": 0,
      "gm_identity": "001747.fffe.700038",
      "gm_present": true,
      "gm_time_base_indicator": 0,
      "ingress_time": 1552413985821448000,
      "last_gm_phase_change": "0x0000'0000000000000000.0000",
      "master_offset": -211488,
      "scaled_last_gm_phase_change": 0
    }
  },
  "sensor": {
    "nmea": {
      "baud_rate": "BAUD_9600",
      "diagnostics": {
        "decoding": {
          "date_decoded_count": 0,
          "last_read_message": "",
          "not_valid_count": 0,
          "utc_decoded_count": 0
        },
        "io_checks": {
          "bit_count": 1,
          "bit_count_unfilterd": 0,
          "char_count": 0,
          "start_char_count": 0
        }
      },
      "ignore_valid_char": 0,
      "leap_seconds": 0,
      "locked": 0,
      "polarity": "ACTIVE_HIGH"
    },
    "sync_pulse_in": {
      "diagnostics": {
        "count": 1,
        "count_unfiltered": 0,
        "last_period_nsec": 0
```

(continues on next page)

(continued from previous page)

```
      },
      "locked": 0,
      "polarity": "ACTIVE_HIGH"
    },
    "sync_pulse_out": {
      "angle_deg": 360,
      "frequency_hz": 1,
      "mode": "OFF",
      "polarity": "ACTIVE_HIGH",
      "pulse_width_ms": 10
    },
    "timestamp": {
      "mode": "TIME_FROM_INTERNAL_OSC",
      "time": 57178.44114677,
      "time_options": {
        "internal_osc": 57178,
        "ptp_1588": 1552413986,
        "sync_pulse_in": 1
      }
    }
  },
  "system": {
    "monotonic": 57191.819600378,
    "realtime": 1552413949.3948405,
    "tracking": {
      "frequency": -7.036,
      "last_offset": 5.942e-06,
      "leap_status": "normal",
      "ref_time_utc": 1552413947.8259742,
      "reference_id": "70747000",
      "remote_host": "ptp",
      "residual_frequency": 0.006,
      "rms_offset": 5.358e-06,
      "root_delay": 1e-09,
      "root_dispersion": 0.000129677,
      "skew": 1.144,
      "stratum": 1,
      "system_time_offset": -2.291e-06,
      "update_interval": 2
    }
  }
}
```

→

Response JSON Object

- string - See sub objects for details.

Status Codes

- 200 OK - No error

`GET system/time/system`

Get the operating system time status. These values relate to the operating system clocks, and not clocks related to hardware timestamp data from the lidar sensor.

```
GET /api/v1/system/time/system HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8

{
  "monotonic": 345083.599570944,
  "realtime": 1551814510.730453,
  "tracking": {
    "frequency": -6.185,
    "last_offset": -3.315e-06,
    "leap_status": "normal",
    "ref_time_utc": 1551814508.1982567,
    "reference_id": "70747000",
    "remote_host": "ptp",
    "residual_frequency": -0.019,
    "rms_offset": 4.133e-06,
    "root_delay": 1e-09,
    "root_dispersion": 0.000128737,
    "skew": 1.14,
    "stratum": 1,
    "system_time_offset": 4.976e-06,
    "update_interval": 2
  }
}
```

→

Response JSON Object

- `monotonic` (`float`) - Monotonic time of operating system. This timestamp never counts backwards and is the time since boot in seconds.
- `realtime` (`float`) - Time in seconds since the Unix epoch, should match wall time if synchronized with external time source.
- `tracking` (`object`) - Operating system time synchronization tracking status. See chronyc tracking documentation for more information.

Status Codes

- 200 OK - No error

System `tracking` fields of interest:

→

Rms_offset Long-term average of the offset value.

System_time_offset Time delta (in seconds) between estimate of the operating system time and the current true time.

Last_offset Estimated local offset on the last clock update.

Ref_time_utc UTC Time at which the last measurement from the reference source was processed.

Remote_host This is either `ptp` if the system is synchronizing to a *PTP* time source or the address of a remote NTP server the system has selected if the sensor is connected to the Internet.

`GET system/time/ptp`

Get the status of the *PTP* time synchronization daemon.

Note: See the IEEE 1588-2008 standard for more details on the standard management messages.

```
GET /api/v1/system/time/ptp HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8

{
  "current_data_set": {
    "mean_path_delay": 30110,
    "offset_from_master": 224159,
    "steps_removed": 1
  },
  "parent_data_set": {
    "gm_clock_accuracy": 33,
    "gm_clock_class": 6,
    "gm_offset_scaled_log_variance": 20061,
    "grandmaster_identity": "001747.fffe.700038",
    "grandmaster_priority1": 128,
    "grandmaster_priority2": 128,
    "observed_parent_clock_phase_change_rate": 2147483647,
    "observed_parent_offset_scaled_log_variance": 65535,
    "parent_port_identity": "001747.fffe.700038-1",
    "parent_stats": 0
  },
  "port_data_set": {
    "announce_receipt_timeout": 3,
    "delay_mechanism": 1,
    "log_announce_interval": 1,
    "log_min_delay_req_interval": 0,
    "log_min_pdelay_req_interval": 0,
    "log_sync_interval": 0,
    "peer_mean_path_delay": 0,
    "port_identity": "bc0fa7.fffe.00012c-1",
    "port_state": "SLAVE",
    "version_number": 2
  },
  "time_properties_data_set": {
    "current_utc_offset": 37,
    "current_utc_offset_valid": 1,
    "frequency_traceable": 1,
    "leap59": 0,
    "leap61": 0,
    "ptp_timescale": 1,
    "time_source": 32,
    "time_traceable": 1
  },
  "time_status_np": {
    "cumulative_scaled_rate_offset": 0,
    "gm_identity": "001747.fffe.700038",
    "gm_present": true,
    "gm_time_base_indicator": 0,
    "ingress_time": 1551814546772493800,
```

(continues on next page)

(continued from previous page)

```
    "last_gm_phase_change": "0x0000'0000000000000000.0000",
    "master_offset": 224159,
    "scaled_last_gm_phase_change": 0
  }
}
```

→

Response JSON Object

- current_data_set (object) - Result of the PMC GET CURRENT_DATA_SET command.
- parent_data_set (object) - Result of the PMC GET PARENT_DATA_SET command.
- port_data_set (object) - Result of the PMC GET PORT_DATA_SET command.
- time_properties_data_set (object) - Result of the PMC GET TIME_PROPERTIES_DATA_SET command.
- time_status_np (object) - Result of the PMC GET TIME_STATUS_NP command. This is a linuxptp non-portable command.

Status Codes

- 200 OK - No error

Fields of interest:

→

Current_data_set.offset_from_master Offset from master time source in nanoseconds as calculated during the last update from master.

Parent_data_set.grandmaster_identity This should match the local grandmaster clock. If this displays the sensor's clock identity (derived from Ethernet hardware address) then this indicates the sensor is not properly synchronized to a grandmaster.

Parent_data_set Various information about the selected master clock.

Port_data_set.port_state This value will be SLAVE when a remote master clock is selected. See parent_data_set for selected master clock.

Port_data_set Local sensor *PTP* configuration values. Grandmaster clock needs to match these for proper time synchronization.

Time_properties_data_set *PTP* properties as given by master clock.

Time_status_np.gm_identity Selected grandmaster clock identity.

Time_status_np.gm_present True when grandmaster has been detected. This may stay true even if grandmaster goes off-line. Use port_data_set.port_state to determine up-to-date synchronization status. When this is false then the local clock is selected.

Time_status_np.ingress_time Indicates when last *PTP* message was received. Units are in nanoseconds.

Time_status_np Linux *PTP* specific diagnostic values. The Red Hat manual provides some more information on these fields

GET system/time/sensor

Get the lidar sensor time status. These values relate to the hardware timestamping mechanism of the sensor.

```
GET /api/v1/system/time/sensor HTTP/1.1
Host: 192.0.2.123
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8

{
  "nmea": {
    "baud_rate": "BAUD_9600",
```

(continues on next page)

(continued from previous page)

```
    "diagnostics": {
      "decoding": {
        "date_decoded_count": 0,
        "last_read_message": "",
        "not_valid_count": 0,
        "utc_decoded_count": 0
      },
      "io_checks": {
        "bit_count": 1,
        "bit_count_unfilterd": 0,
        "char_count": 0,
        "start_char_count": 0
      }
    },
    "ignore_valid_char": 0,
    "leap_seconds": 0,
    "locked": 0,
    "polarity": "ACTIVE_HIGH"
  },
  "sync_pulse_in": {
    "diagnostics": {
      "count": 1,
      "count_unfiltered": 0,
      "last_period_nsec": 0
    },
    "locked": 0,
    "polarity": "ACTIVE_HIGH"
  },
  "sync_pulse_out": {
    "angle_deg": 360,
    "frequency_hz": 1,
    "mode": "OFF",
    "polarity": "ACTIVE_HIGH",
    "pulse_width_ms": 10
  },
  "timestamp": {
    "mode": "TIME_FROM_INTERNAL_OSC",
    "time": 57178.44114677,
    "time_options": {
      "internal_osc": 57178,
      "ptp_1588": 1552413986,
      "sync_pulse_in": 1
    }
  }
}
```

For more information on these parameters refer to Section 3.3 for the `get_time_info` TCP command.

9 PTP Quickstart Guide

There are many configurations for a PTP network, this quick start guide aims to cover the basics by using Ubuntu 18.04 as an example. It provides configuration settings for a commercial PTP grandmaster clock and also provides directions on setting up a Linux computer (Ubuntu 18.04) to function as a PTP grandmaster.

The linuxptp project provides a suite of PTP tools that can be used to serve as a PTP master clock for a local network of sensors.

Table of Contents

9.1 Assumptions

- Command line Linux knowledge (e.g., package management, command line familiarity, etc.).
- Ethernet interfaces that support hardware timestamping.

- Ubuntu 18.04 is assumed for this tutorial, but any modern distribution should suffice.
- Knowledge of systemd service configuration and management.
- Familiarity with Linux permissions.

9.2 Physical Network Setup

Ensure the Ouster sensor is connected to the PTP master clock with at most one network switch. Ideally the sensor should be connected directly to the PTP grandmaster. Alternatively, a simple layer-2 gigabit Ethernet switch will suffice. Multiple switches are not recommended and will add unnecessary jitter.

9.3 Third Party Grandmaster Clock

A dedicated grandmaster clock should be used for the highest absolute accuracy often with a GPS receiver.

It must be configured with the following parameters which match the *linuxptp* client defaults:

- Transport: `UDP IPv4`
- Delay Mechanism: `E2E`
- Sync Mode: `Two-Step`
- Announce Interval: `1` - sent every 2 seconds
- Sync Interval: `0` - sent every 1 second
- Delay Request Interval: `0` - sent every 1 second

For more settings, review the `port_data_set` field returned from the sensor's *HTTP system/time/ptp* interface.

9.4 Linux PTP Grandmaster Clock

An alternative to an external grandmaster PTP clock is to run a local Linux PTP master clock if accuracy allows. This is often implemented on a vehicle computer that interfaces directly with the lidar sensors.

This section outlines how to configure a master clock.

- *Configuring `ptp4l` for Multiple Ports*
- *Configuring `ptp4l` as a Local Master Clock*
- *Configuring `phc2sys` to Synchronize the System Time to the PTP Clock*
- *Configuring Chrony to Set System Clock Using PTP*

Example Network Setup

This section assumes the following network setup as it has elements of a local master clock and the option for an upstream PTP time source.

```
+-------------------------------------+
| Ubuntu 18.04 System                 |
| * 2x Intel i210 Ethernet Interfaces |
| * Linux PTP service                 |
|                                     |
|      eno1                 eno2      |
+-------+---------------------+-------+
        |                     |
+-------+-------+    +--------+------+
| Trimble GM100 |    |               + +
|  GPS -> PTP   |    |  Ouster OS1   | |
|  grandmaster  |    |               | |
|  (optional)   |    |               | |
+---------------+    +---------------- |
                      +--------------- +
```

The focus is on configuring the Linux PTP service to serve a common clock to all the downstream Ouster OS1 sensors using the Linux system time from the Ubuntu host machine.

Optionally, a grandmaster clock can be added to discipline the system time of the Linux host.

Installing Necessary Packages

Several packages are needed for PTP functionality and verification:

- `linuxptp` - Linux PTP package with the following components:
  - `ptp4l` daemon to manage hardware and participate as a PTP node
  - `phc2sys` to synchronize the Ethernet controller's hardware clock to the Linux system clock or shared memory region
  - `pmc` to query the PTP nodes on the network.
- `chrony` - A NTP and PTP time synchronization daemon. It can be configured to listen to both NTP time sources via the Internet and a PTP master clock such as one provided a GPS with PTP support. This will validate the time configuration makes sense given multiple time sources.
- `ethtool` - A tool to query the hardware and driver capabilities of a given Ethernet interface.

```
$ sudo apt update
...
Reading package lists... Done
Building dependency tree
Reading state information... Done

$ sudo apt install linuxptp chrony ethtool
Reading package lists... Done
Building dependency tree
```

(continues on next page)

(continued from previous page)

```
Reading state information... Done
The following NEW packages will be installed:
  chrony ethtool linuxptp
0 upgraded, 3 newly installed, 0 to remove and 29 not upgraded.
Need to get 430 kB of archives.
After this operation, 1,319 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 ethtool amd64 1:4.15-0ubuntu1 [114 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 linuxptp amd64 1.8-1 [112 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 chrony amd64 3.2-4ubuntu4.2 [203 kB]
Fetched 430 kB in 1s (495 kB/s)
Selecting previously unselected package ethtool.
(Reading database ... 117835 files and directories currently installed.)
Preparing to unpack .../ethtool_1%3a4.15-0ubuntu1_amd64.deb ...
Unpacking ethtool (1:4.15-0ubuntu1) ...
Selecting previously unselected package linuxptp.
Preparing to unpack .../linuxptp_1.8-1_amd64.deb ...
Unpacking linuxptp (1.8-1) ...
Selecting previously unselected package chrony.
Preparing to unpack .../chrony_3.2-4ubuntu4.2_amd64.deb ...
Unpacking chrony (3.2-4ubuntu4.2) ...
Setting up linuxptp (1.8-1) ...
Processing triggers for ureadahead (0.100.0-20) ...
ureadahead will be reprofiled on next reboot
Setting up chrony (3.2-4ubuntu4.2) ...
Processing triggers for systemd (237-3ubuntu10.13) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up ethtool (1:4.15-0ubuntu1) ...
```

Ethernet Hardware Timestamp Verification

Identify the ethernet interface to be used on the client (Linux) machine, e.g., eno1. Run the ethtool utility and query this network interface for supported capabilities.

Output of `ethtool -T` for a functioning Intel i210 Ethernet interface:

```
$ sudo ethtool -T eno1
Time stamping parameters for eno1:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
```

Configuring ptp4l for Multiple Ports

On a Linux system with multiple Ethernet ports (i.e. Intel i210) ptp4l needs to be configured to support all of them.

Modify /etc/linuxptp/ptp4l.conf and append the following, replacing eno1 and eno2 with the appropriate interface names:

```
boundary_clock_jbod 1
[eno1]
[eno2]
```

The default systemd service file for Ubuntu 18.04 attempts to use the eth0 address on the command line. Override systemd service file so that the configuration file is used instead of hard coded in the service file.

Create a systemd drop-in directory to override the system service file:

```
$ sudo mkdir -p /etc/systemd/system/ptp4l.service.d
```

Create a file at /etc/systemd/system/ptp4l.service.d/override.conf with the following contents:

```
[Service]
ExecStart=
ExecStart=/usr/sbin/ptp4l -f /etc/linuxptp/ptp4l.conf
```

Restart the ptp4l service so the change takes effect:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart ptp4l
$ sudo systemctl status ptp4l
* ptp4l.service - Precision Time Protocol (PTP) service
   Loaded: loaded (/lib/systemd/system/ptp4l.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/ptp4l.service.d
           └─override.conf
   Active: active (running) since Wed 2019-03-13 14:38:57 PDT; 3s ago
     Docs: man:ptp4l
 Main PID: 25783 (ptp4l)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/ptp4l.service
           └─25783 /usr/sbin/ptp4l -f /etc/linuxptp/ptp4l.conf

Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.756] port 1: INITIALIZING to LISTENING on INITIALIZE
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.756] driver changed our HWTSTAMP options
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.756] tx_type   1 not 1
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.756] rx_filter 1 not 12
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.756] port 2: INITIALIZING to LISTENING on INITIALIZE
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.757] port 0: INITIALIZING to LISTENING on INITIALIZE
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.757] port 1: link up
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.757] port 2: link down
Mar 13 14:38:57 leadlizard ptp4l[25783]: [590188.757] port 2: LISTENING to FAULTY on FAULT_DETECTED (FT_
↪UNSPECIFIED)
Mar 13 14:38:58 leadlizard ptp4l[25783]: [590189.360] port 1: new foreign master 001747.fffe.700038-1
```

The above systemctl status ptp4l console output shows systemd correctly read the override file created earlier before starting several seconds after the restart command.

The log output shows that a grandmaster clock has been discovered on port 1 (eno1) and port 2 (eno2) is currently disconnected and in the faulty state as expected. In the test network a Trimble Thunderbolt PTP GM100 Grandmaster Clock is attached on eno1.

Logs can be monitored (i.e. followed) like so:

```
$ journalctl -f -u ptp4l
-- Logs begin at Fri 2018-11-30 06:40:50 PST. --
Mar 13 14:51:37 leadlizard ptp4l[25783]: [590948.224] master offset        -17 s2 freq  -25963 path delay  ⊠
↪   14183
Mar 13 14:51:38 leadlizard ptp4l[25783]: [590949.224] master offset        -13 s2 freq  -25964 path delay  ⊠
↪   14183
Mar 13 14:51:39 leadlizard ptp4l[25783]: [590950.225] master offset         35 s2 freq  -25920 path delay  ⊠
↪   14192
Mar 13 14:51:40 leadlizard ptp4l[25783]: [590951.225] master offset        -59 s2 freq  -26003 path delay  ⊠
↪   14201
Mar 13 14:51:41 leadlizard ptp4l[25783]: [590952.225] master offset        -24 s2 freq  -25986 path delay  ⊠
↪   14201
Mar 13 14:51:42 leadlizard ptp4l[25783]: [590953.225] master offset        -39 s2 freq  -26008 path delay  ⊠
↪   14201
Mar 13 14:51:43 leadlizard ptp4l[25783]: [590954.225] master offset         53 s2 freq  -25928 path delay  ⊠
↪   14201
Mar 13 14:51:44 leadlizard ptp4l[25783]: [590955.226] master offset        -85 s2 freq  -26050 path delay  ⊠
↪   14207
Mar 13 14:51:45 leadlizard ptp4l[25783]: [590956.226] master offset        127 s2 freq  -25863 path delay  ⊠
↪   14207
Mar 13 14:51:46 leadlizard ptp4l[25783]: [590957.226] master offset          9 s2 freq  -25943 path delay  ⊠
↪   14208
Mar 13 14:51:47 leadlizard ptp4l[25783]: [590958.226] master offset        -23 s2 freq  -25973 path delay  ⊠
↪   14208
Mar 13 14:51:48 leadlizard ptp4l[25783]: [590959.226] master offset        -61 s2 freq  -26018 path delay  ⊠
↪   14190
Mar 13 14:51:49 leadlizard ptp4l[25783]: [590960.226] master offset         69 s2 freq  -25906 path delay  ⊠
↪   14190
Mar 13 14:51:50 leadlizard ptp4l[25783]: [590961.226] master offset        -73 s2 freq  -26027 path delay  ⊠
↪   14202
Mar 13 14:51:51 leadlizard ptp4l[25783]: [590962.226] master offset         19 s2 freq  -25957 path delay  ⊠
↪   14202
Mar 13 14:51:52 leadlizard ptp4l[25783]: [590963.226] master offset        147 s2 freq  -25823 path delay  ⊠
↪   14202
...
```

Configuring ptp4l as a Local Master Clock

The IEEE-1588 Best Master Clock Algorithm (*BMCA*) will select a grandmaster clock based on a number of masters. In most networks there should be only a single master. In the example network the Ubuntu machine will be configured with a non-default *clockClass* so its operation qualifies it to win the BMCA.

Replace the default value with a lower clock class (higher priority) and restart linuxptp. Edit /etc/linuxptp/ptp4l.conf and comment out the default clockClass value and insert a line setting it 128.

```
#clockClass     248
clockClass      128
```

Restart ptp4l so the configuration change takes effect.

```
$ sudo systemctl restart ptp4l
```

This will configure `ptp4l` to advertise a master clock on eno2 as a clock that will win the BMCA for an Ouster OS1 sensor.

However, the `ptp4l` service is only advertising the Ethernet controller's PTP hardware clock, not the Linux system time as is often expected.

Configuring `phc2sys` to Synchronize the System Time to the PTP Clock

To synchronize the Linux system time to the the PTP hardware clock the `phc2sys` utility needs to be run. The following configuration will tell `phc2sys` to take the Linux `CLOCK_REALTIME` and write that time to the PTP hardware clock in the Ethernet controller for `eno2`. These interfaces are then connected to PTP slaves such as Ouster OS1 sensors.

Create a systemd drop-in directory to override the system service file:

```
$ sudo mkdir -p /etc/systemd/system/phc2sys.service.d
```

Create a file at `/etc/systemd/system/phc2sys.service.d/override.conf` with the following contents:

```
[Service]
ExecStart=
ExecStart=/usr/sbin/phc2sys -w -s CLOCK_REALTIME -c eno2
```

Note: If multiple interfaces need to be synchronized from `CLOCK_REALTIME` then multipe instances of the `phc2sys` service need to be run as it only accepts a single slave (i.e. `-c`) argument.

Restart the `phc2sys` service so the change takes effect:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart phc2sys
$ sudo systemctl status phc2sys
```

Configuring Chrony to Set System Clock Using PTP

An upstream PTP grandmaster clock (e.g., a GPS disciplined PTP clock) can be used to set the system time if precise absolute time is needed for sensor data. Chrony is a Linux time service that can read from NTP and PTP and set the Linux system time using the most accurate source available. With a proper functioning PTP grandmaster the PTP time source will be selected and the error from the public time servers can be reviewed.

The following phc2shm service will synchronize the time from `eno1` (where the external grandmaster is attached) to the system clock.

Create a file named `/etc/systemd/system/phc2shm.service` with the following contents:

```
# /etc/systemd/system/phc2shm.service
[Unit]
Description=Synchronize PTP hardware clock (PHC) to NTP SHM
Documentation=man:phc2sys
After=ntpdate.service
Requires=ptp4l.service
After=ptp4l.service

[Service]
Type=simple
ExecStart=/usr/sbin/phc2sys -s eno1 -E ntpshm -w

[Install]
WantedBy=multi-user.target
```

Then start the newly created service and check that it started.

```
$ sudo systemctl start phc2shm
$ sudo systemctl status phc2shm
```

Add the PTP time source to the chrony configuration which will read the shared memory region managed by the `phc2shm` service created above.

Append the following to the `/etc/chrony/chrony.conf` file:

```
refclock SHM 0 poll 1 refid ptp
```

Restart chrony so the updated configuration file takes effect:

```
$ sudo systemctl restart chrony
```

After waiting a minute for the clock to synchronize, review the chrony client timing accuracy:

```
$ chronyc tracking
Reference ID    : 70747000 (ptp)
Stratum         : 1
Ref time (UTC)  : Thu Mar 14 02:22:58 2019
System time     : 0.000000298 seconds slow of NTP time
Last offset     : -0.000000579 seconds
RMS offset      : 0.001319735 seconds
Frequency       : 0.502 ppm slow
Residual freq   : -0.028 ppm
Skew            : 0.577 ppm
Root delay      : 0.000000001 seconds
Root dispersion : 0.000003448 seconds
Update interval : 2.0 seconds
Leap status     : Normal

$ chronyc sources -v
```

(continues on next page)

(continued from previous page)

```
210 Number of sources = 9

  .-- Source mode  '^' = server, '=' = peer, '#' = local clock.
 / .- Source state '*' = current synced, '+' = combined , '-' = not combined,
| /   '?' = unreachable, 'x' = time may be in error, '~' = time too variable.
||                                                 .- xxxx [ yyyy ] +/- zzzz
||      Reachability register (octal) -.           |  xxxx = adjusted offset,
||      Log2(Polling interval) --.      |          |  yyyy = measured offset,
||                                \     |          |  zzzz = estimated error.
||                                 |    |           \
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
#* ptp                           0   1   377     1    +27ns[  +34ns] +/-  932ns
^- chilipepper.canonical.com     2   6   377    61   -482us[ -482us] +/-   99ms
^- pugot.canonical.com           2   6   377    62   -498us[ -498us] +/-  112ms
^- golem.canonical.com           2   6   337    59   -467us[ -468us] +/-   95ms
^- alphyn.canonical.com          2   6   377    58   +957us[ +957us] +/-   95ms
^- legacy13.chi1.ntfo.org        3   6   377    62    -10ms[  -10ms] +/-  178ms
^- tesla.selinc.com              2   6   377   128   +429us[ +514us] +/-   42ms
^- io.crash-override.org         2   6   377    59   +441us[ +441us] +/-   58ms
^- hadb2.smatwebdesign.com       3   6   377    58  +1364us[+1364us] +/-   99ms
```

Note that the `Reference ID` matches the `ptp` refid from the chrony.conf file and that the sources output shows the `ptp` reference id as selected (signified by the `*` state in the second column). Additionally, the NTP time sources show a small relative error to the high accuracy PTP time source.

In this case the PTP grandmaster is properly functioning.

If this error is large, chrony will select the NTP time sources and mark the PTP time source as invalid. This typically signifies that something is mis-configured with the PTP grandmaster upstream of this device or the linuxptp configuration.

9.5 Verifying Operation

If the PTP grandmaster was just setup and configured, it's recommended to power cycle the sensor. The sensor will then jump to the correct time instead of slowly easing in the time adjustment which will take time if the grandmaster initially set the sensor to the wrong time.

HTTP API

The sensor can be queried for the state of its local PTP service through the *HTTP system/time/ptp*.

JSON response fields to check:

- `parent_data_set.grandmaster_identity` should list the identity of the local grandmaster
- `port_data_set.port_state` should be `SLAVE`

LinuxPTP PMC Tool

The sensor will respond to PTP management messages. The linuxptp `pmc` (see `man pmc`) utility can be used to query all PTP devices on the local network.

On the Linux host for the `pmc` utility to communicate with then run the following command:

```
$ sudo pmc 'get PARENT_DATA_SET' 'get CURRENT_DATA_SET' 'get PORT_DATA_SET' 'get TIME_STATUS_NP' -i eno2
sending: GET PARENT_DATA_SET
sending: GET CURRENT_DATA_SET
sending: GET PORT_DATA_SET
sending: GET TIME_STATUS_NP
        bc0fa7.fffe.c48254-1 seq 0 RESPONSE MANAGEMENT PARENT_DATA_SET
                parentPortIdentity                    ac1f6b.fffe.1db84e-2
                parentStats                           0
                observedParentOffsetScaledLogVariance 0xffff
                observedParentClockPhaseChangeRate    0x7fffffff
                grandmasterPriority1                  128
                gm.ClockClass                         6
                gm.ClockAccuracy                      0x21
                gm.OffsetScaledLogVariance            0x4e5d
                grandmasterPriority2                  128
                grandmasterIdentity                   001747.fffe.700038
        bc0fa7.fffe.c48254-1 seq 1 RESPONSE MANAGEMENT CURRENT_DATA_SET
                stepsRemoved     2
                offsetFromMaster 613554162.0
                meanPathDelay    117977.0
        bc0fa7.fffe.c48254-1 seq 2 RESPONSE MANAGEMENT PORT_DATA_SET
                portIdentity            bc0fa7.fffe.c48254-1
                portState               LISTENING
                logMinDelayReqInterval  0
                peerMeanPathDelay       0
                logAnnounceInterval     1
                announceReceiptTimeout  3
                logSyncInterval         0
                delayMechanism          1
                logMinPdelayReqInterval 0
                versionNumber           2
        bc0fa7.fffe.c48254-1 seq 3 RESPONSE MANAGEMENT TIME_STATUS_NP
                master_offset              613554162
                ingress_time               0
                cumulativeScaledRateOffset +0.000000000
                scaledLastGmPhaseChange    0
                gmTimeBaseIndicator        0
                lastGmPhaseChange          0x0000'0000000000000000.0000
                gmPresent                  true
                gmIdentity                 001747.fffe.700038
```

9.6 Tested Grandmaster Clocks

- **Trimble Thunderbolt PTP GM100 Grandmaster Clock**
  - Firmware version: `20161111-0.1.4.0, November 11 2016 15:58:25`

        - PTP configuration:

          →

```
> get ptp eth0
             Enabled : Yes
            Clock ID : 001747.fffe.700038-1
             Profile : 1588
       Domain number : 0
  Transport protocol : IPV4
             IP Mode : Multicast
     Delay Mechanism : E2E
           Sync Mode : Two-Step
         Clock Class : 6
          Priority 1 : 128
          Priority 2 : 128
       Multicast TTL : 0
       Sync interval : 0
    Del Req interval : 0
       Ann. interval : 1
Ann. receipt timeout : 3
```

- **Ubuntu 18.04 + Linux PTP as a master clock**
        - Intel i210 Ethernet interface
            - PCI hardware identifiers: `8086:1533 (rev 03)`
    - Ubuntu 18.04 kernel package: `linux-image-4.18.0-16-generic`
    - Ubuntu 18.04 linuxptp package: `linuxptp-1.8-1`

HTTP Routing Table