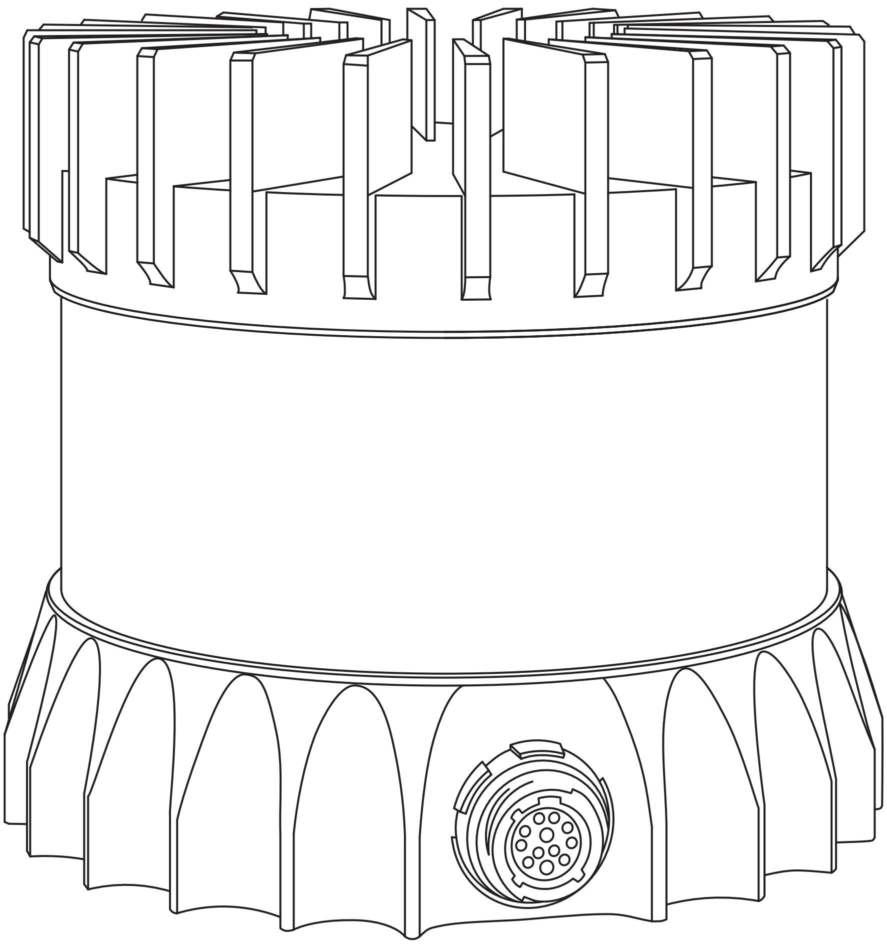


OS1 Gen1 User Guide

Release v1.14.0-beta.10

Ouster OS1 Gen1 High Resolution Imaging Lidar

Jun 08, 2020

Safety and Legal

1 Safety & Legal Notices

The OS1-64, OS1-32, and OS1-16 have been evaluated to be **Class 1 laser products** per **60825-1: 2014 (Ed. 3)** and operate in the 850nm band.

L'OS1-64, l'OS1-32, et l'OS1-16 répondent aux critères des **produits laser de classe 1**, selon la norme **IEC 60825-1: 2014 (3ème édition)** et émettent dans le domaine de l'infrarouge, à une longueur d'onde de 850nm environ.

FDA 21CFR1040 Notice: OS1-64, OS1-32, and OS1-16 comply with FDA performance standards for laser products except for deviations pursuant to Laser Notice No. 56, dated January 19,2018.

Notice FDA 21CFR1040: L'OS1-64, l'OS1-32, et l'OS1-16 sont conformes aux exigences de performances établies par la FDA pour les produits laser, à l'exception des écarts en application de l'avis nº56, daté du 19 janvier 2018.

CAUTIONS:

- The OS1 is a hermetically sealed unit, and is non user-serviceable.
- Use of controls, or adjustments, or performance of procedures other than those specified herein, may result in hazardous radiation exposure.
- 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
  - Permitting any third party to do any of the foregoing
- Operating the sensor without either the attached mount with which the sensor is shipped, or attaching the sensor to a surface of appropriate thermal capacity runs the risk of having the sensor overheat under certain circumstances.

PRECAUTIONS:

- L'OS1 est une unité hermétiquement scellée, qui ne peut être entretenue ou modifiée par l'utilisateur.
- L'utilisation de commandes, de réglages, ou l'exécution de procédures autres que celles spécifiées dans le présent document peuvent entraîner des rayonnements laser dangereux.
- L'utilisation de l'OS1 est soumise aux conditions de vente signées avec Ouster ou le distributeur/intégrateur, incluant l'interdiction de:
  - Retirer ou ouvrir de quelque façon le boîtier du capteur
  - Analyser les composants internes du capteur
  - Pratiquer la rétro-ingénierie de toute ou partie du capteur
  - Autoriser une tierce personne à mener les actions listées ci-dessus

Equipment Label: Note that the equipment label, which includes model and serial number and notice that the unit is a Class 1 Laser Product, is affixed to the underside of the Sensor Enclosure Base itself. It is only visible after the attached mount with which the Sensor is shipped, is removed. Please refer to location details in *Mechanical Interface*.

L'étiquette de l'équipement, comprenant le modèle, le numéro de série, et la classification du produit laser (ici, classe 1), est apposée au dessous de la base du boîtier du capteur. Il n'est visible qu'après avoir retiré le diffuseur de chaleur avec lequel le capteur est expédié. L'emplacement est décrit avec précision dans le Guide de l'Utilisateur, dans la section «*Mechanical Interface*».

Electromagnetic Compatibility: The OS1 is an FCC 47 CfR 15 Subpart B device. This device complies with part 15 of the FCC Rules. Operation is subject to the following conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation

"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.

2 Proper Assembly, Maintenance and Safe Use

The OS1 may be easily setup by mounting to the base to a mounting with the correct mounting hole pattern, and following the interconnection instructions delineated in *Mounting and Heatsinking Guidelines*. Any mounting orientation is acceptable. Each sensor is shipped attached to a mount for test or normal use specified operating temperature range, but the sensor may be mounted directly to any appropriate mount with Thermal Capacity appropriate for the application of the user. Please contact Ouster for assistance with approving the use of user specific mounting arrangements.

Any attempt to utilize the sensor outside the Environmental parameters delineated in the OS1 data sheet may result in voiding of the warranty.

When power is applied, the sensor powers up and commences boot-up with the laser disabled. The bootup sequence is approximately 60s in duration, after which the internal sensor optics subassembly commences spinning, and the laser is activated, and the unit operates in the default 1024 x 10 Hz mode. When the sensor is running, and the laser is operating, a faint red flickering light may be seen behind the optical window. Note that the OS1 utilizes an 850nm infrared laser that is only dimly discernable to the naked eye, while transmitting a laser eye- safe fundamental signal in the 850nm IR band. While the sensor is fully Class 1 eye safe, Ouster strongly recommends against peering into the optical window at close range while the sensor is operating. The OS1 is a hermetically sealed unit, and is not user-serviceable. Any attempt to unseal the enclosure has the potential to expose the operator to hazardous laser radiation.

Ouster sensors are equipped with a multi-layer series of internal safety interlocks to ensure compliance to Class 1 Laser Eye Safe limits. The Sensor Software User interface may be used configure the sensor to a number of combinations of scan rates and resolutions other than the default values of 1024 x 10 Hz resolution. In all available combinations, the unit has been evaluated by an NRTL to remain within the classification of a Class 1 Laser Device as per IEC 60825-1:2014 (Ed. 3).

2.1 Assemblage correct et utilisation sûre

L'OS1 s'installe facilement en fixant la base sur un support percé de trous concordants, et en suivant les instructions d'interconnexion décrites dans le Guide de l'Utilisateur «*Mounting and Heatsinking Guidelines*». Toute orientation de montage est acceptable. Chaque capteur est expédié équipé d'un dissipateur de chaleur, utilisable en phase de test et en conditions normales. Néanmoins tout autre support présentant une capacité thermique appropriée pour l'application de l'utilisateur peut être utilisé. Veuillez contacter Ouster dans le cas où un montage spécifique à votre application serait nécessaire.

Toute tentative d'utilisation du capteur en dehors des paramètres environnementaux définis dans la fiche technique de l'OS1 peut entraîner l'annulation de la garantie.

Lorsque l'on observe le capteur en cours d'utilisation, on peut apercevoir une faible lumière rouge vacillante derrière la vitre teintée. Cette lumière, à peine perceptible à l'œil nu est inoffensive pour l'oeil: elle accompagne les rayonnements laser de classe 1 de l'OS1 émis dans le domaine de l'infrarouge, eux-même sans danger pour l'oeil humain. Cependant, bien que les rayonnements laser de class 1 soient sans danger dans des conditions raisonnablement prévisibles, Ouster recommande fortement de ne pas regarder fixement la vitre teintée pendant que le capteur est en marche.

L'OS1 est une unité hermétiquement scellée, qui ne peut pas être entretenue, modifiée ou réparée. Toute tentative d'ouverture du boîtier a pour risque d'exposer l'opérateur à un rayonnement laser dangereux.

Les capteurs Ouster sont équipés d'une série de dispositifs de sécurité à plusieurs niveaux, de façon à assurer en toutes circonstances le respect des limites d'irradiance correspondant aux rayonnements lasers de classe 1, sans danger pour les yeux.

L'interface utilisateur du logiciel du capteur peut être utilisée pour configurer le capteur selon un certain nombre de combinaisons de vitesses de balayage et de résolutions autres que les valeurs utilisées par défaut, respectivement de 1024 x 10 Hz.

3 OS1 Gen1 Overview

The OS1 Gen1 offers an industry-leading combination of price, performance, reliability, size, weight, and power. It is 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 facias, robots, and drones.

The OS1 family of sensors consist of three models, the OS1-64, OS1-32, and OS1-16, 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.

3.1 What's in the box

The OS1 Gen1 is shipped with the following items:

- OS1 lidar sensor
- 24V AC/DC power supply
- Interface Box
- RJ45 gigabit Ethernet cable
- Mounting guidelines
- Baseplate heatsink

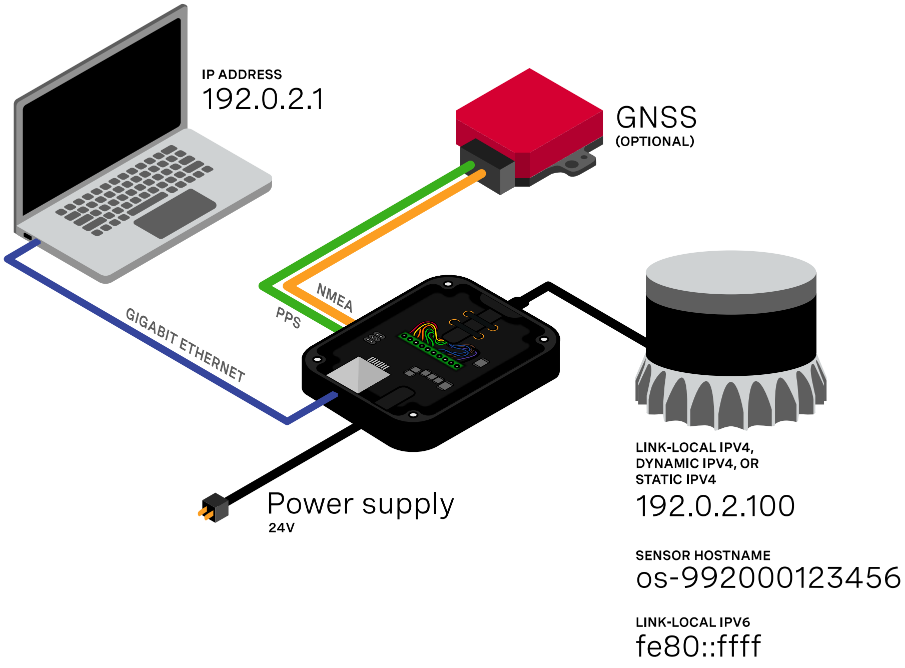
Please contact your sales representative or local distributor for custom accessories including optional mounts and interface boxes with different length cables.

4 OS1 Product Models

The Gen 1 OS1 is available with 64, 32, or 16 beams of vertical resolution and with uniform, gradient, or below horizon beam spacing options. Product specs and more information on these configurations can be found on the OS1 product page.

5 Connecting to Sensor Overview

The OS1 requires a computer with a gigabit Ethernet connection and a 24V supply. Optionally you may time synchronize the sensor through an external time source or through the computer via PTP.



6 Network Configuration

The sensor is designed to communicate with a host machine through a variety of different methods such a DHCP, IPv6/IPv4 link-local, and static IP.

On most systems you should be able to connect the sensor into your network or directly to a host machine and simple use the senor hostname to communicate with it.

The sensor hostname is, `os-991234567890.local`, where `991234567890` is the sensor serial number.

For more detailed guidance on communicating with the sensor on various operating systems and network setttings please reference the *Networking Guide*.

7 Sensor Visualization

After connecting to your sensors, you can quickly visualize the point cloud through either Ouster Studio or with our sample drivers. Both Ouster Studio and our sample drivers are available for Linux, Mac, and Windows. Please contact your sales rep if you do not already have links to either the latest Ouster Studio or the sample drivers.

Currently the latest public Ouster Studio drivers are found at https://www.paraview.org/ousterstudio/ and the latest sample drivers are released on www.github.com/ouster. Note that while firmware v1.14 is still in beta, the latest public visualization tools are still meant for Gen1 sensors.

8 Coordinate Frames and XYZ Calculation

8.1 Lidar Coordinate Frame

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° elevation beam angle of the lidar).

The Lidar Coordinate Frame axes are arranged with:

- positive x-axis pointed at encoder angle 0° and the red external connector
- positive y-axis pointed towards encoder angle 90°
- positive z-axis pointed towards the top of the sensor

The Lidar Coordinate Frame is marked in both diagrams below with X_L, Y_L, and Z_L.

8.2 Lidar Range to XYZ

Given the following information, range data may be transformed into 3D cartesian XYZ coordinates in the Lidar Coordinate Frame:

From an azimuth data block from the UDP packet:

- `encoder_count` of the azimuth block
- `range_mm` value of the data block of the *i*-th channel

From the `get_beam_intrinsics` TCP command:

- `lidar_origin_to_beam_origin_mm` value
- `beam_altitude_angles` array
- `beam_azimuth_angles` array

The corresponding 3D point can be computed by

$$\begin{aligned}
r &= range\_mm \\
n &= lidar\_origin\_to\_beam\_origin\_mm \\
\theta_{encoder} &= -2\pi \frac{encoder\_count}{90112} \\
\theta_{azimuth} &= -2\pi \frac{beam\_azimuth\_angles[i]}{360} \\
\phi &= 2\pi \frac{beam\_altitude\_angles[i]}{360}
\end{aligned}$$

$$\begin{aligned}
x &= (r-n)\cos(\theta_{encoder}+\theta_{azimuth})\cos(\phi) + n\cos(\theta_{encoder}) \\
y &= (r-n)\sin(\theta_{encoder}+\theta_{azimuth})\cos(\phi) + n\sin(\theta_{encoder}) \\
z &= (r-n)\sin(\phi)
\end{aligned}$$

Figures Fig. 8.1 and Fig. 8.2 show, respectively, a top-down and side view of the sensor.

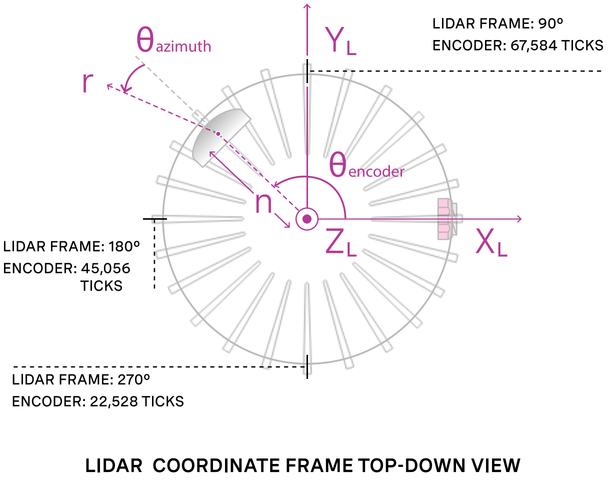
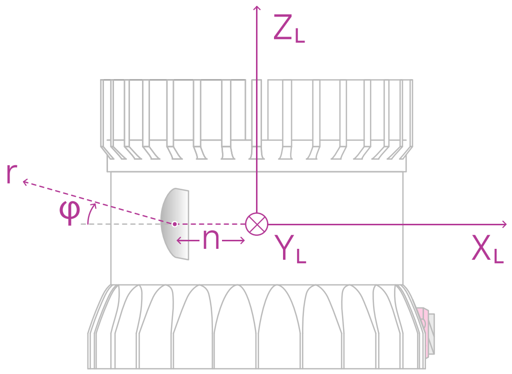


Figure8.1: Top-down view of Lidar Coordinate Frame

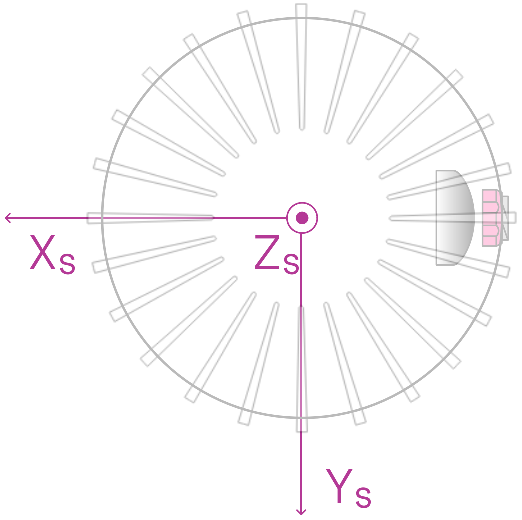
8.3 Sensor Coordinate Frame

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.



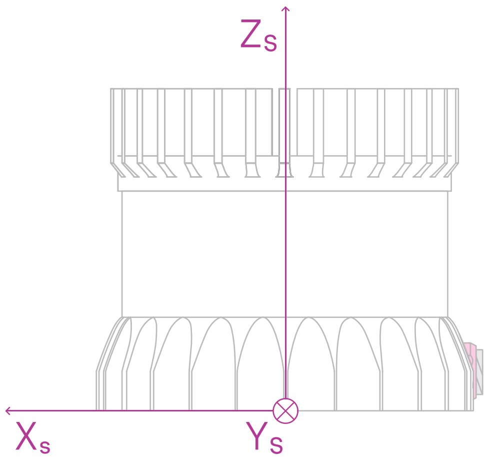
LIDAR COORDINATE FRAME SIDE VIEW

Figure8.2: Side view of Lidar Coordinate Frame



SENSOR COORDINATE FRAME TOP-DOWN VIEW

Figure8.3: Top-down view of Sensor Coordinate Frame



SENSOR COORDINATE FRAME SIDE VIEW

Figure8.4: Side view of Sensor Coordinate Frame

8.4 Combining Lidar and Sensor Coordinate Frame

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° position and ends at the 360° 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 90,111, the actual angle about the z-axis in the Lidar Coordinate Frame will decrease.

8.5 Lidar Intrinsic Beam Angles

The intrinsic beam angles for each beam may be queried with a TCP command `get_beam_intrinsics` to provide an azimuth and elevation adjustmen offset 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.

8.6 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 for the OS1, and the data must be rotated 180º around the z-axis. This information can be queried over TCP in the form of a homogeneous transformation matrix in row-major ordering.

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,
    74.296,
    0,
    0,
    0,
    1
    ]
 }
```

Which corresponds to the following matrix

$$M\_lidar\_to\_sensor = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 74.296 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The table below lists all product lines' distances of the aperture stop above the sensor origin for use in the z translation.

Product Line	Lidar aperture stop above sensor origin
OS0	36.180 mm
OS1	36.180 mm
OS2	74.296 mm

8.7 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 homogeneous transformation matrix in row-major ordering.

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

```
{
  "imu_to_sensor_transform": [
    1,
    0,
    0,
    6.253,
    0,
    1,
    0,
    -11.775,
    0,
    0,
    1,
    7.645,
    0,
    0,
    0,
    1
    ]
}
```

Which corresponds to the following matrix

$$M\_imu\_to\_sensor = \begin{bmatrix} 1 & 0 & 0 & 6.253 \\ 0 & 1 & 0 & -11.775 \\ 0 & 0 & 1 & 7.645 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

9 Lidar Data

9.1 Lidar Data Format

Note: The calculations and illustrations of the lidar data in this section presupposes the use of the latest firmware v1.14.0 or v2.0.0. Previous firmware did not scale lidar data packet size with number of channels in a sensor. Please contact support@ouster.io if you need an older version of the guide to lidar data format.

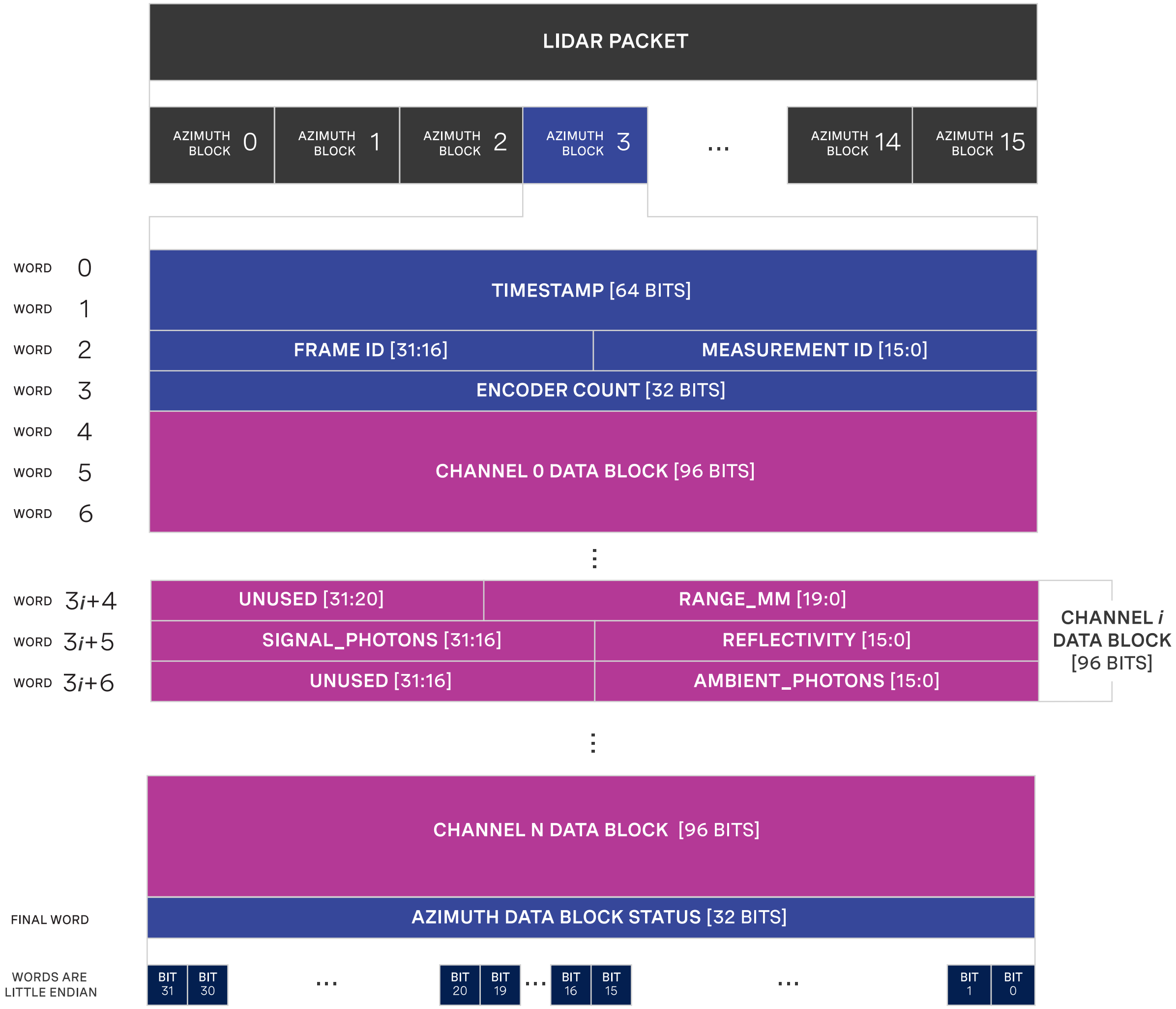
Lidar data packets consist of 16 azimuth blocks and varies in length relative to the number of channels in the sensor. The packet rate is dependent on the output mode. Words are 32 bits in length and little endian. By default, lidar UDP data is forwarded to Port `7502`.

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 90,111 - 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 pixel, each containing:
  - **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 Photons** [16 bit unsigned int] - ambient photons in the ambient 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.

9.2 Lidar Data Packet Size Calculation

The table below shows the lidar data packet size breakdown for all products. Since the size of the azimuth block varies proportional the number of channels in a sensors, all sensors with the same number of channels have the same lidar packet data structure and size.



N+1 = NUMBER OF CHANNELS IN SENSOR, E.G. 16, 32, 64, 128

Product	Number of words in azimuth block	Size of single azimuth block (Bytes)	Size of lidar packet (Bytes)
OS1-16	53	212	3,392
OS0-32, OS1-32, OS2-32	101	404	6,464
OS0-64, OS1-64, OS2-64	197	788	12,608
OS0-128, OS1-128, OS2-128	389	1,556	24,896

10 IMU Data

10.1 IMU Data Format

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

Each IMU data block contains:

- **IMU Diagnostic Time** [64 bit unsigned int] - timestamp of monotonic system time since boot in nanoseconds.
- **Accelerometer Read Time** [64 bit unsigned int] - timestamp for accelerometer time relative to *timestamp_mode* in nanoseconds.
- **Gyroscope Read Time** [64 bit unsigned int] - timestamp for gyroscope time relative to *timestamp_mode* in nanoseconds.
- **Acceleration in X-axis** [32 bit float] - acceleration in g.
- **Acceleration in Y-axis** [32 bit float] - acceleration in g.
- **Acceleration in Z-axis** [32 bit float] - acceleration in g.
- **Angular Velocity about X-axis** [32 bit float] - Angular velocity in deg per sec.
- **Angular Velocity about Y-axis** [32 bit float] - Angular velocity in deg per sec.
- **Angular Velocity about Z-axis** [32 bit float] - Angular velocity in deg per sec.

	IMU PACKET
WORD 0	IMU DIAGNOSTIC TIME [64 BITS]
WORD 1	
WORD 2	ACCELEROMETER READ TIME [64 BITS]
WORD 3	
WORD 4	GYROSCOPE READ TIME [64 BITS]
WORD 5	
WORD 6	ACCELERATION IN X-AXIS [32 BITS]
WORD 7	ACCELERATION IN Y-AXIS [32 BITS]
WORD 8	ACCELERATION IN Z-AXIS [32 BITS]
WORD 9	ANGULAR VELOCITY ABOUT X-AXIS [32 BITS]
WORD 10	ANGULAR VELOCITY ABOUT Y-AXIS [32 BITS]
WORD 11	ANGULAR VELOCITY ABOUT Z-AXIS [32 BITS]

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

The second two timestamps, (Words 2,3) and (Words 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.

11 Data Rates

The table below calculates the data of all products operating at the highest lidar modes, `2048x10` or `1024x20` (*).

Product	Lidar packet size (Bytes)	Lidar packets rate * (Hz)	IMU packet size (Bytes)	IMU packets per second	Data rate (Mbps)
OS1-16	3,392	1,280	48	100	34.77
OS0-32, OS1-32, OS2-32	6,464	1,280	48	100	66.23
OS0-64, OS1-64, OS2-64	12,608	1,280	48	100	129.14
OS0-128, OS1-128, OS2-128	24,896	1,280	48	100	254.97

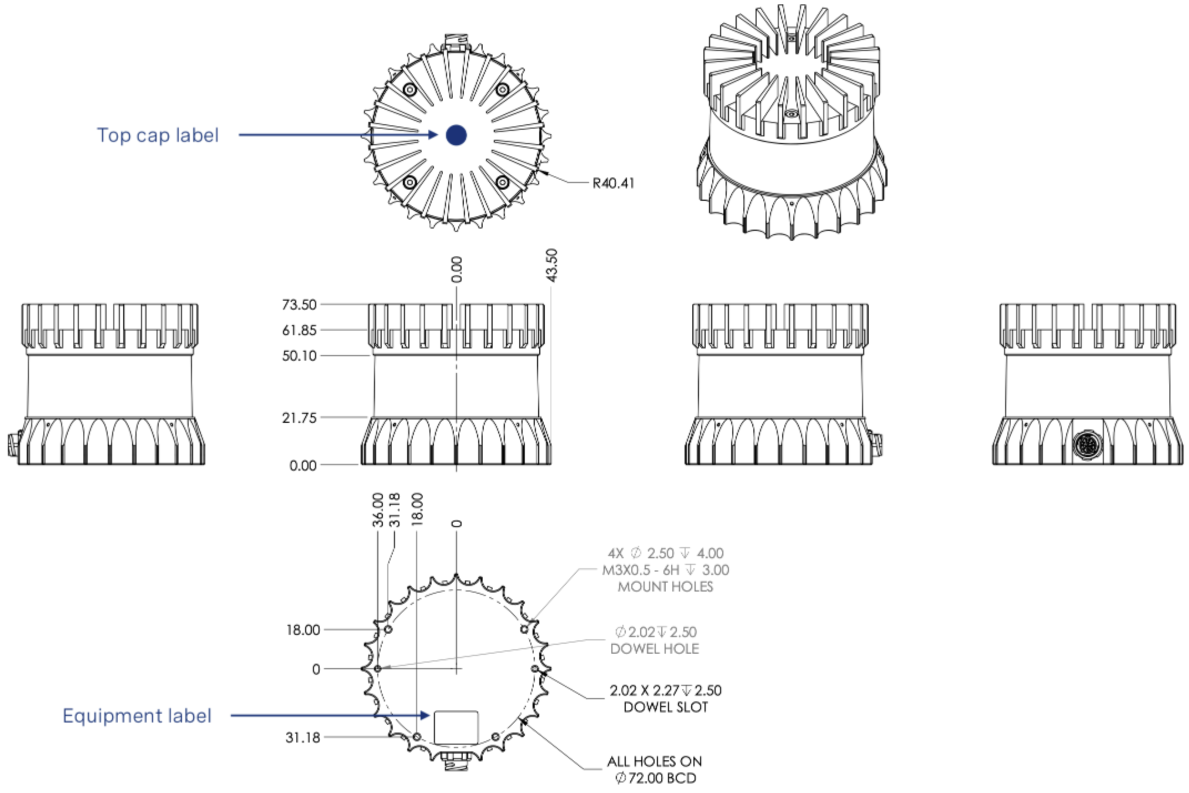
Lidar packets account for >99% of data coming from the sensor. For most applications, a gigabit Ethernet network connection is required for reliable performance.

12 Mechanical Interface

12.1 Included Components

The OS1 is shipped with the following items

- OS1-64, OS1-32, or OS1-16
- Sensor to interface box cable/connector
- Interface box
- Interface box AC/DC power supply (2 meters)
- RJ45 cable (1 meter)
- Optional: Heat sink



Downloadable CAD files for the OS1 can be found online at www.ouster.com/lidar-product-details

Warning: Water ingress protection: The sensor ingress protection rating is only valid if the I/O cable is plugged into the panel mount connector on the base of the sensor, and the locking collet rotated past the detent click to the properly locked condition i.e past the détente position. The cable and plug are an element of the sensor ingress protection system. Without this the ingress protection rating may be compromised. Bending the cable at a sharp angle directly after egress from the plug over mold should also be avoided. Sharp bends and high axial stresses on the cable immediately adjacent to the plug over mold may create a moisture ingress path into the the connector. Please note the cable minimum bend radius requirements below:

I/O Cable type	**O.D.**	**Cable Minimum Bend Radius**	
		Fixed Bend	**Flexing Bend**
Ouster Thick Cable	10.5mm	79mm (7.5*OD)	158mm (15*OD)
Ouster Thin Cable (Standard)	8mm	50mm (5*OD)	80mm (10*OD)

12.2 Mounting and Heatsinking Guidelines

Our sensors ship with modular mounting options. Proper mounting will ensure optimal sensor performance, reducing noise from vibration and providing efficient heat dissipation.

- Mount to a material with high thermal conductivity. The following are recommended aluminum alloys and their thermal conductivity:

  1) 6061: 167 W/m-K
  2) 7075: 130 W/m-K
  3) 2024: 121 W/m-K

- Ensure interfaces are clean and free from debris
- Torque bolts appropriately for the mount material and bolts
- Use TIM (Thermal Interface Material) for any irregular or unmachined surfaces
- Do not overconstrain the sensor if mounting to both the top and the bottom
- Use a thermally conductive pad to ensure good conductivity while not overconstraining.
- Ensure your implementation maintains the base and top of the sensor at no greater than 25ºC above ambient with an ambient less than 50ºC
- The shape of any heatsink should maximize the surface area for free and forced convection while being thick enough to allow the heat to conduct through the material

If you have questions about your specific mounting situation please contact the Ouster at support@ouster.io.

12.3 Thermal Requirements

Thermal requirements specific to Gen 1 OS1 are listed below. Note that the modular cap are sensors with removable and modular caps with radial fins whereas legacy caps cannot be removed.

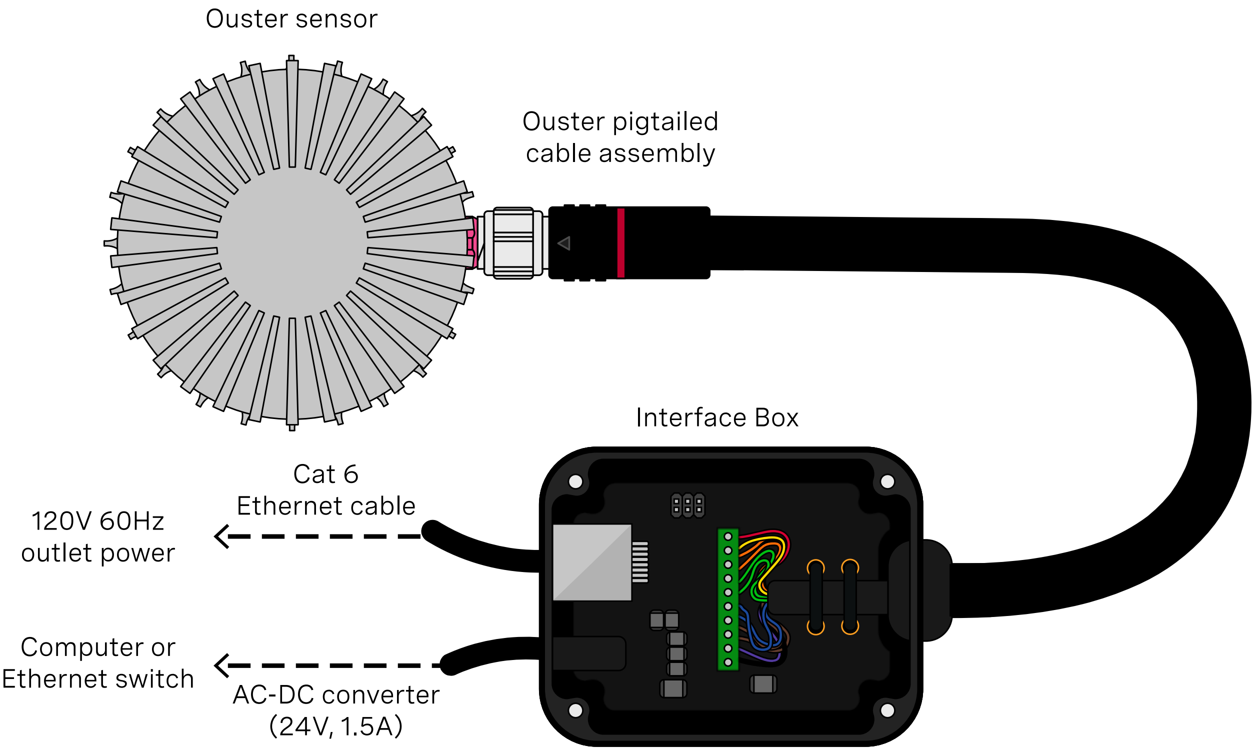
Table 12.1: Thermal requirements for Gen1 OS1 with modular caps

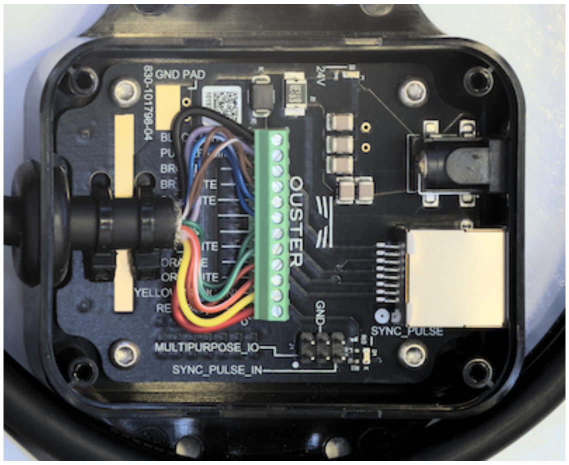
	Requirements		**Example Test Case**
	Base Enclosure Temp (ºC)	**Top Modular Cap Temp (ºC)**	**Convective Air Temp with Radial Heatsink and Standard Base (ºC)**
Legacy Cap - Max Temp before Shot Limiting	55	55	50
Legacy Cap - Max Temp before Sensor Shuts Off	65	65	60
Modular Cap - Max Temp before Shot Limiting	52	52	47
Modular Cap - Max Temp before Sensor Shuts Off	65	65	60

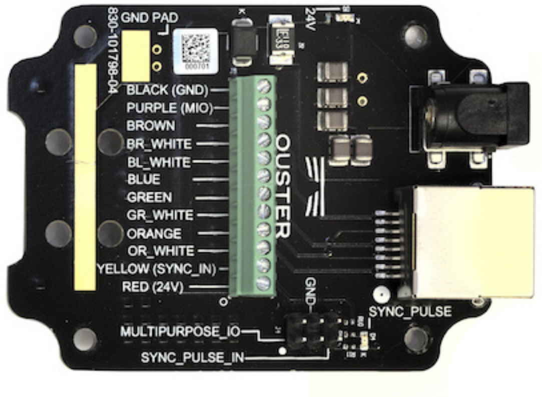
13 Electrical Interface

13.1 Interface Box

The Interface Box that accompanies the OS1 is designed to allow the sensor to be operated for test and evaluation purposes. It terminates the interface cable from the sensor, allows it to be powered up and provides access to the sensor gigabit Ethernet Interface via a standard RJ45 connector. DC Power to the sensor is provided to the Interface Box by the accompanying 24V DC supply.


Ouster sensor
Ouster pigtailed cable assembly
Interface Box
Cat 6 Ethernet cable
120V 60Hz outlet power
Computer or Ethernet switch
AC-DC converter (24V, 1.5A)





13.2 Direct Cable Connection and Pinout

The OS1 can be operated without the use of an Interface Box.

Warning: Ouster is not responsible for any errors in wiring as a result of bypassing the Interface Box and this activity may result in a voiding of your warranty if it results in damage to the sensor. The following guidelines for direct cable connection assume use of the Ouster-provided 24V 1.5A power supply. Ouster cannot be held responsible for damage to the device if alternate is used

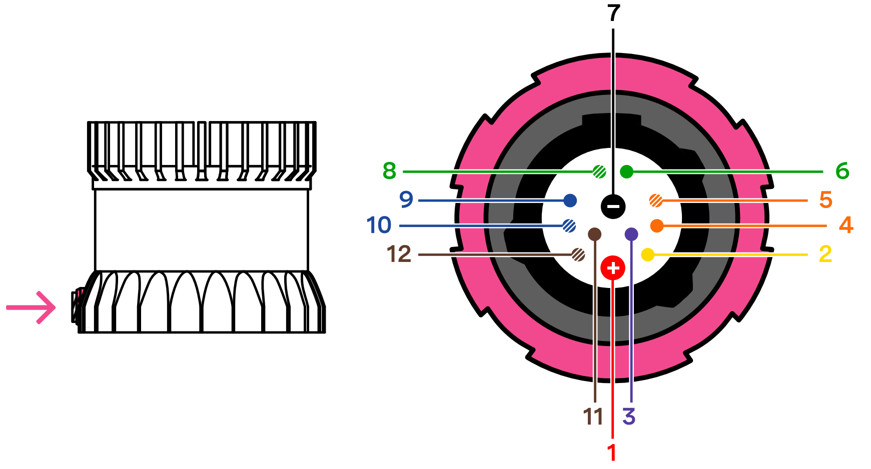


Figure13.1: cable pinout of on-sensor receptacle

Table13.1: Ouster Cable Pinout: Sinbon-Type Connector, Sensor P/Ns: 840_101855, 840_102144, 840_102145, 840_102146

Net Name	Pin Number	Wire	Twisted With
MULTIPURPOSE_IO	3	Purple, 26 AWG	N/A
SYNC_PULSE_IN	2	Yellow, 26 AWG	N/A
VCC_24	1	Red, 22 AWG	N/A
GROUND	7	Black, 22 AWG	N/A
TRP_1_P (Ethernet)	5	White/Orange, 26 AWG	Orange
TRP_1_N (Ethernet)	4	Orange, 26 AWG	White/Orange
TRP_2_P (Ethernet)	8	White/Green, 26 AWG	Green
TRP_2_N (Ethernet)	6	Green, 26 AWG	White/Green
TRP_3_P (Ethernet)	9	Blue, 26 AWG	White/Blue
TRP_3_N (Ethernet)	10	White/Blue, 26 AWG	Blue
TRP_4_P (Ethernet)	12	White/Brown, 26 AWG	Brown
TRP_4_N (Ethernet)	11	Brown, 26 AWG	White/Brown

14 Digital IO

14.1 SYNC_PULSE_IN

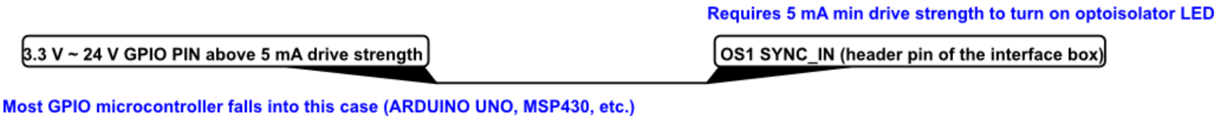
SYNC_PULSE_IN is a dedicated input channel that is accessible within the Interface Box Jumper J4. This channel expects an input pulse sequence which can be used for time synchronization. See the *Setting Configuration Parameters* for more information on configuring this input. Any references to pulse polarity in this document references the signal polarity on the SYNC_PULSE_IN pin of the sensor. This input channel is protected by an optoisolator which will draw 10 mA at full turn on.

Table14.1: SYNC_PULSE_IN Interface Requirements

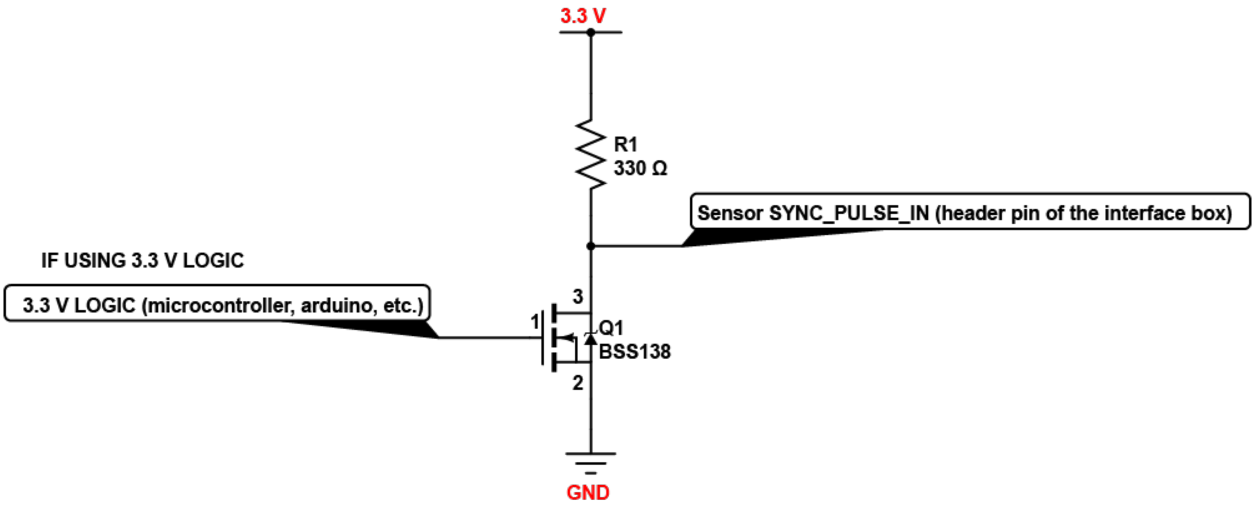
Parameter	Min Voltage	Max Voltage	Min Driver Current
LOGIC LOW	0 V	1 V	N/A
LOGIC HIGH	3.3 V	15 V	5 mA

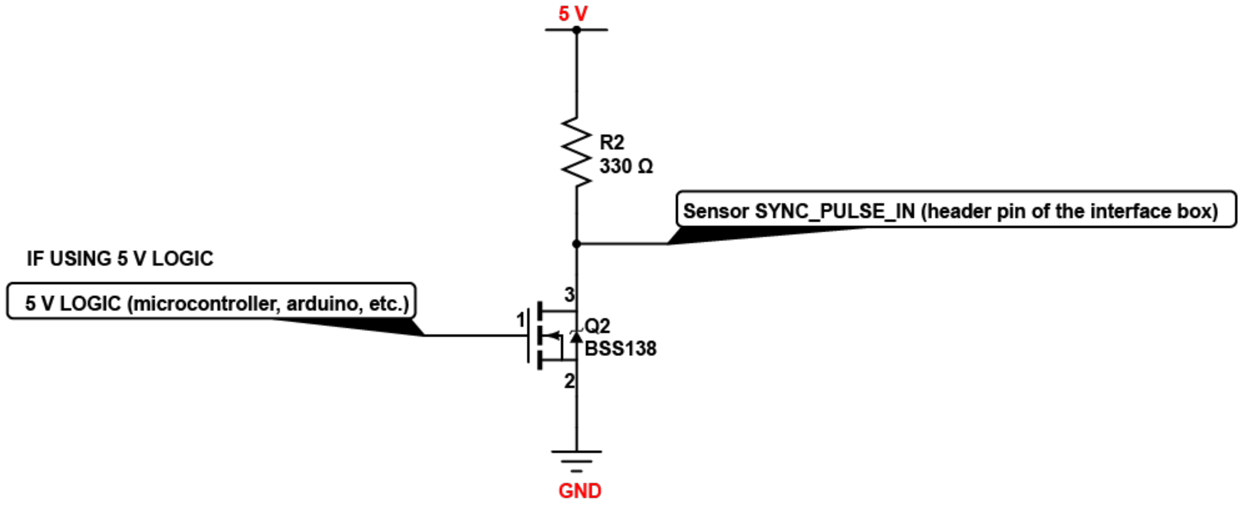
SYNC_PULSE_IN Interface Requirements were tested with 2 m cable interface box connection at 2 MHz.

- When GPIO has 5 mA drive strength minimum, GPIO can be directly connected to the SYNC_PULSE_IN pin of the interface box header. This is the most common case and has been tested to work on common Arduino microcontroller series. Typical common logic levels of 3.3 V, 5 V GPIO of microcontrollers can produce drive strength of 5 mA min (Arduino, MSP430, etc.).


Requires 5 mA min drive strength to turn on optoisolator LED
3.3 V ~ 24 V GPIO PIN above 5 mA drive strength
OS1 SYNC_IN (header pin of the interface box)
Most GPIO microcontroller falls into this case (ARDUINO UNO, MSP430, etc.)

- If the 5 mA drive strength minimum cannot be met, a buffer circuit is required to drive SYNC_PULSE_IN. Example circuits are provided for common 3.3 V and 5 V logic.


3.3 V
R1
330 Ω
Sensor SYNC_PULSE_IN (header pin of the interface box)
IF USING 3.3 V LOGIC
3.3 V LOGIC (microcontroller, arduino, etc.)
3
1
Q1
BSS138
2
GND


5 V
R2
330 Ω
Sensor SYNC_PULSE_IN (header pin of the interface box)
IF USING 5 V LOGIC
5 V LOGIC (microcontroller, arduino, etc.)
3
1
Q2
BSS138
2
GND

14.2 MULTIPURPOSE_IO (M_IO)

`MULTIPURPOSE_IO` (`M_IO`) is a configurable input or output channel accessible within the Interface Box Jumper J4 connected to the MULTIPURPOSE_IO pin of the Interface Box. Detailed information on how to configure this channel using the sensor TCP interface can be found in the *TCP API* section. By default this channel is disabled.

When this channel is configured as an **OUTPUT**, the `M_IO` sends a pulse sequence that can be used for timesynchronization or event triggering outside the sensor. For a full description of output pulse triggering options, see the *Sensor Modes* section. This output is an optoisolated open collector circuit, relying on an externally provided pull-up resistor. This resistor is not provided as part on of the interface box.

Table14.2: MULTIPURPOSE_IO - OUTPUT Interface Requirements

Parameter	Min	Max
Pull Up Voltage	N/A	15 V
Sinking Current	N/A	25 mA

Output is an optoisolated open collector. However current OS1 units have an earlier version of this port which requires an external circuit to utilize this port in output mode. See the below MULTIPURPOSE_IO ERRATA section for more details.

When this channel is configured as an **INPUT**, the `M_IO` can accept a standard NMEA $GPRMC UART message. These messages are a common way for GPS systems to share timestamp information in UTC time format. More information on this packet structure and supported baud rates can be found in the *Time Synchronization* section.

Table14.3: MULTIPURPOSE_IO - INPUT Interface Requirements

Parameter	Min Voltage	Max Voltage	Min Driver Current
LOGIC LOW	0 V	1 V	N/A
LOGIC HIGH	1.7 V	15 V	10 mA

Above are tested with 2 m cable interface box connection at 2 MHz.

Note: Please note that these input channels are being renamed. Interface Boxes may be labeled with the deprecated name.`PPS_MASTER` has been renamed to `MULTIPURPOSE_IO`, and `PPS_SLAVE` has been renamed to `SYNC_PULSE_IN`

15 Web Interface

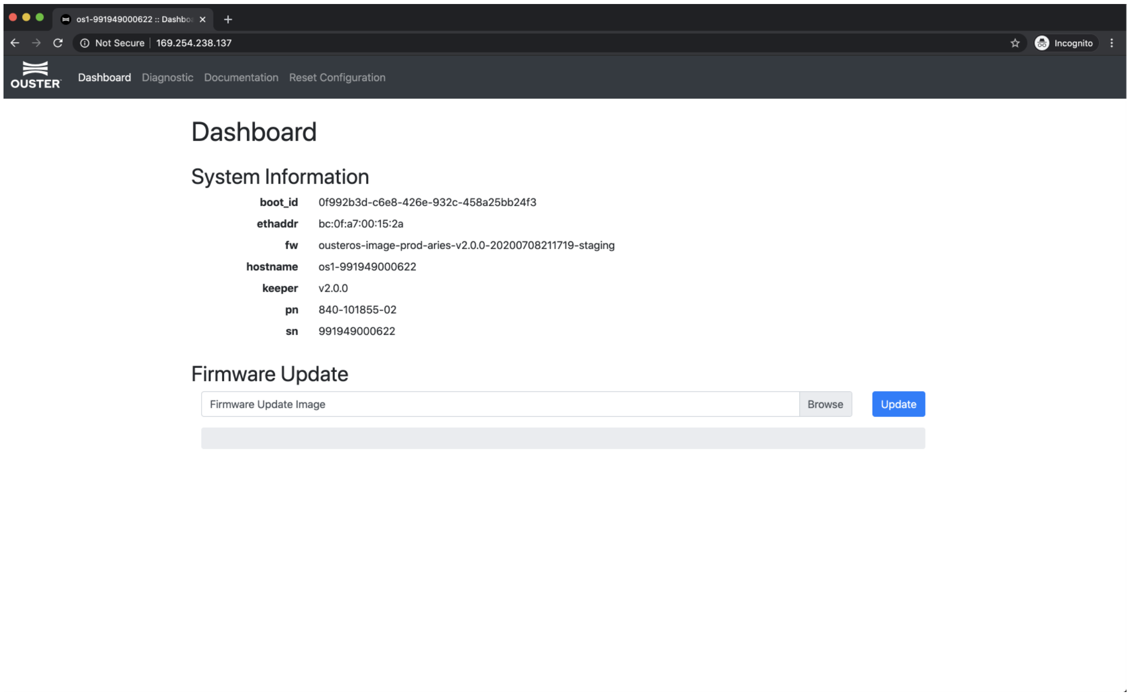
The sensor homepage can be access by typing in the sensor's address (IPv4, IPv6, or hostname) in a web browser. From here you can see information about the sensor, accesss documentation, and reset sensor settings.

Dashboard: Contains basic information about the sensor. You can update firmware on this page. See *Updating Firmware* for more details.

Diagnostic: Contains diagnostic alert and error information about the sensor for troubleshooting purposes. For a list of possible alerts and errors, see *Alerts and Errors*.

Documentation: Contains the HTTP and TCP API guides that are compatible the version of the firmware on the sensor. See www.ouster.com for latest user manuals and data sheets.

Reset Configuration: Resets sensor to factory configurations and settings. Note that this resets any static IP address given to the sensor.



Dashboard

System Information

boot_id	0f992b3d-c6e8-426e-932c-458a25bb24f3
ethaddr	bc:0f:a7:00:15:2a
fw	ousteros-image-prod-aries-v2.0.0-20200708211719-staging
hostname	os1-991949000622
keeper	v2.0.0
pn	840-101855-02
sn	991949000622

Firmware Update

Firmware Update Image

Figure15.1: The sensor homepage, accessed through its IPv4 link-local address

16 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`

16.1 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-v2.0.0"
}
```

→

Response JSON Object

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

Status Codes

- 200 OK - No error

16.2 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

16.3 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,
      "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,
```

(continues on next page)

(continued from previous page)

```
      "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
      },
      "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,
```

(continues on next page)

(continued from previous page)

```
      "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,
```

(continues on next page)

(continued from previous page)

```
    "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,
```

(continues on next page)

(continued from previous page)

```
    "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,
    "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",
    "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
      }
    },
```

(continues on next page)

(continued from previous page)

```
    "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 the `get_time_info` TCP command.

17 TCP API

17.1 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. Note that "os-xxx" refers to the sensor serial number and can look like "os-xxx" or "os1-xxx".

```
$ nc os-991900123456 7501
get_sensor_info
{"status": "RUNNING", "image_rev": "ousteros-image-prod-aries-v2.0.0-20200429230129",
 "base_pn": "000-101323-03", "prod_sn": "991900123456", "proto_rev": "v1.1.1",
 "base_sn": "101837000752", "prod_line": "OS-1-128", "build_rev": "v2.0.0",
 "prod_pn": "840-101855-02", "build_date": "2020-01-28T22:58:18Z"}
```

A sensor may have one of the following statuses:

Status	Description
INITIALIZING	When the sensor is booting and not yet outputting data.
WARMUP	Sensor has gone into thermal warmup state.
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 manual power cycle or reboot.

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

The following commands will return sensor configuration and calibration information:

Table17.1: 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]`
		`}`

continues on next page

Table 17.1 - continued from previous page

Command	Description	Response Example

`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-v2.0.0-20200429230129",`
		`"base_pn": "000-101323-03",`
		`"prod_sn": "991900123456",`
		`"proto_rev": "v1.1.1",`
		`"base_sn": "101837000752",`
		`"prod_line": "OS-1-128",`
		`"build_rev": "v2.0.0",`
		`"prod_pn": "840-101855-02",`
		`"build_date": "2020-01-28T22:58:18Z"`
		`}`

continues on next page

Table 17.1 - continued from previous page

Command	Description	Response Example

get_time_info	Returns JSON-formatted sensor timing configuration and status of udp timestamp, sync_pulse_in, and multipurpose_io.	{
		    "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"
		        }
		    }
		}

continues on next page

Table 17.1 - continued from previous page

<table>
<tr><th>Command</th><th>Description</th><th>Response Example</th></tr>
<tr><td>get_beam_intrinsics</td><td>Returns JSON-formatted beam altitude and azimuth offsets, in degrees. Length of arrays is equal to the number of channels in sensor. Also returns distance between lidar origin and beam origin in mm, to be used for point cloud calculations.</td><td><pre>
{
  "lidar_origin_to_beam_origin_mm": 13.762,
  "beam_altitude_angles": [
    16.926,
    16.313,
    "...",
    -16.078,
    -16.689
  ],
  "beam_azimuth_angles": [
    3.052,
    0.857,
    "...",
    -0.868,
    -3.051
  ]
}
</pre></td></tr>
<tr><td>get_imu_intrinsics</td><td>Returns JSON-formatted IMU transformation matrix needed to adjust to the Sensor Coordinate Frame.</td><td><pre>
{
  "imu_to_sensor_transform": [
    1,
    0,
    0,
    6.253,
    0,
    1,
    0,
    -11.775,
    0,
    0,
    1,
    7.645,
    0,
    0,
    0,
    1
  ]
}
</pre></td></tr>
</table>

continues on next page

Table 17.1 - 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`
		`  ]`
		`}`

continues on next page

Table 17.1 - continued from previous page

<table>
<tr><th>Command</th><th>Description</th><th>Response Example</th></tr>
<tr>
<td>get_alerts <START_CURSOR></td>
<td>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.</td>
<td><pre>
{
   "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"
      }
   ]
}
</pre></td>
</tr>
</table>

continues on next page

Table 17.1 - continued from previous page

Command	Description	Response Example

get_lidar_data_format	Returns JSON-formatted response that describes the structure of a lidar packet.
	columns_per_frame: Number of azimuth columns per frame. This can be 512, 1024, or 2048, depending upon the set lidar mode.
	columns_per_packet: Number of azimuth blocks contained in a single lidar packet. Currently in v1.14 and earlier, this is 16.
	pixel_shift_by_row: Offset in terms of pixel count. Can be used to destagger image. Varies by lidar mode. Length of this array is equal to the number of channels of the sensor.
	pixels_per_column: Number of channels of the sensor.
		{
		"columns_per_frame": 1024,
		"columns_per_packet": 16,
		"pixel_shift_by_row": [
		18,
		12,
		6,
		0,
		"...",
		18,
		12,
		6,
		0
		],
		"pixels_per_column": 128
		}

17.2 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:

Table17.2: 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.	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 time synchronization section in sensor user guides for 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)

continues on next page

Table 17.2 - continued from previous page

get_config_param	Command Description	Response
auto_start_flag	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)

Table17.3: Sensor Modes

Command	Command Description	Response
multipurpose_io_mode	Returns the configured mode of the MULTIPURPOSE_IO pin. See Time Syncrhonization section in sensor user guide 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)

17.3 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:

Table17.4: 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

continues on next page

Table 17.4 - 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 Time Sycrhonization section in sensor user guide for 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

continues on next page

Table 17.4 - continued from previous page

set_config_param	Command Description	Response
auto_start_flag <1/0>	Set 1 to ensure that the *next* time the sensor starts up, it will stay in the dormant state until it is manually commanded to start spinning and firing lasers again. It will **not** put the sensor into the dormant state while it is already running. To use the auto_start_state configuration, sensor must be power cycled after setting this flag. When it comes up again, it will be in the UNCONFIGURED state and will not lase or spin. Running the following commands will bring it back into the RUNNING state: set_config_param auto_start_flag 1, reinitialize, write_config_txt. This mode is useful for power-sensitive applications, e.g. drone flights.	0 (default)

Table17.5: 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 the Time Syncrhonization section of the sensor 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 the Time Syncrhonization section of sensor user guide.	set_config_param on success, error: otherwise

continues on next page

Table 17.5 - continued from previous page

set_config_param	Command Description	Response
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 the Time Syncrhonization section of sensor user guide.	set_config_param on success, error: otherwise

Table17.6: 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

18 Troubleshooting

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 problems 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 5, 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 The TCP API Guide.

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": ""
      }
    ]
}
```

19 Common Issues

20 Alerts and Errors

The sensor will provide a list of active alerts and errors via the TCP command *get_alerts*. These alerts also viewable on the sensor homepage under the Diagnostics tab.

All possible alerts and errors that the sensor can provide are listed below. Where appropriate, the message from the sensor aims to help the user diagnose and fix the issue themselves.

Note: The alerts and errors in the table below assume latest firmware v1.14.0 or v2.0.0. Previous firmware versions do not have the same set of alerts of errors.

Table20.1: Alerts and Errors in v1.14.0 / v2.0.0

id	category	level	msg
0x00000000	UNKNOWN	ERROR	An unknown error has occurred; please contact Ouster at https://ouster.com/tech-support.
0x01000000	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x01000001	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x01000002	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x01000003	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x01000004	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.

continues on next page

Table 20.1 - continued from previous page

id	category	level	msg
0x01000005	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x01000006	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x01000007	UNDERTEMP	ERROR	Unit internal temperature too low; please see user guide for heat sinking requirements.
0x01000008	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x01000009	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x0100000A	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x0100000B	OVERTEMP	ERROR	Unit internal temperature too high; please see user guide for heat sinking requirements.
0x0100000C	INTERNAL_COMM	ERROR	Unit has experienced an internal COMM error; please contact Ouster at https://ouster.com/tech-support.
0x0100000D	INTERNAL_COMM	ERROR	Unit has experienced an internal COMM error; please contact Ouster at https://ouster.com/tech-support.
0x0100000E	SHOT_LIMITING	NOTICE	Temperature is high enough where shot limiting may be engaged; please see user guide for heat sinking requirements.
0x0100000F	SHOT_LIMITING	WARNING	Shot limiting mode is active. Laser power is partially attenuated; please see user guide for heat sinking requirements.
0x01000010	INTERNAL_FW	ERROR	Unit has experienced an internal error; please contact Ouster at https://ouster.com/tech-support.
0x01000011	ETHERNET_LINK_BAD	WARNING	Ethernet link bad, please check network switch and harnessing can support 1 Gbps Ethernet.
0x01000012	INTERNAL_COMM	WARNING	Unit has experienced an internal COMM warning: some measurements may have been skipped.
0x01000013	INTERNAL_COMM	WARNING	Unit has experienced an internal COMM warning: some measurements may have been skipped.

continues on next page

Table 20.1 - continued from previous page

id	category	level	msg
0x01000014	INTERNAL_COMM	WARNING	Unit has experienced an internal COMM warning: some measurements may have been skipped.
0x01000015	UDP_TRANSMISSION	WARNING	Client not reachable on lidar data port; check udp_ip, udp_port_lidar, and client machine.
0x01000016	UDP_TRANSMISSION	WARNING	Could not send lidar data UDP packet to host; check that network is up.
0x01000017	UDP_TRANSMISSION	WARNING	Received an unknown error when trying to send lidar data UDP packet; closing socket.
0x01000018	UDP_TRANSMISSION	WARNING	Client not reachable on IMU port; check udp_ip, udp_port_imu, and client machine.
0x01000019	UDP_TRANSMISSION	WARNING	Could not send IMU UDP packet to host; check that network is up.
0x0100001A	UDP_TRANSMISSION	WARNING	Received an unknown error when trying to send IMU UDP packet; closing socket.
0x0100001B	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100001C	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100001D	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100001E	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100001F	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000020	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000021	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000022	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.

continues on next page

Table 20.1 - continued from previous page

id	category	level	msg
0x01000023	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000024	STARTUP	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000025	INTERNAL_COMM	ERROR	Unit has experienced an internal COMM error; please contact Ouster at https://ouster.com/tech-support.
0x01000026	INTERNAL_COMM	ERROR	Unit has experienced an internal COMM error; please contact Ouster at https://ouster.com/tech-support.
0x01000027	INTERNAL_COMM	ERROR	Unit has experienced an internal COMM error; please contact Ouster at https://ouster.com/tech-support.
0x01000028	STARTUP	WARNING	Unit has experienced an internal warning during startup and is restarting.
0x01000029	STARTUP	WARNING	Unit has experienced an internal warning during startup and is restarting.
0x0100002A	STARTUP	WARNING	Unit has experienced an internal warning during startup and is restarting.
0x0100002B	STARTUP	WARNING	Unit has experienced an internal warning during startup and is restarting.
0x0100002C	STARTUP	WARNING	Unit has experienced an internal warning during startup and is restarting.
0x0100002D	STARTUP	WARNING	Unit has experienced an internal warning during startup and is restarting.
0x0100002E	INPUT_VOLTAGE	WARNING	Input voltage is close to being too low. Raise voltage immediately.
0x0100002F	INPUT_VOLTAGE	ERROR	Input voltage is too low. Unit shutting down.
0x01000030	INPUT_VOLTAGE	WARNING	Input voltage is close to being too high. Lower voltage immediately.
0x01000031	INPUT_VOLTAGE	ERROR	Input voltage is too high. Unit shutting down.
0x01000032	UDP_CONNECT	WARNING	Couldn't open lidar UDP socket; please contact Ouster at https://ouster.com/tech-support.
0x01000033	UDP_CONNECT	WARNING	Couldn't resolve IP address; check network and udp_ip.

continues on next page

Table 20.1 - continued from previous page

id	category	level	msg
0x01000034	UDP_CONNECT	WARNING	Invalid UDP port number; check network and udp_port_lidar.
0x01000035	UDP_CONNECT	WARNING	Couldn't reach lidar network.
0x01000036	UDP_CONNECT	WARNING	Couldn't open imu UDP socket; please contact Ouster at https://ouster.com/tech-support.
0x01000037	UDP_CONNECT	WARNING	Couldn't resolve IP address; check network and udp_ip.
0x01000038	UDP_CONNECT	WARNING	Invalid UDP port number; check network and udp_port_imu.
0x01000039	UDP_CONNECT	WARNING	Couldn't reach IMU network.
0x0100003A	SHOT_LIMITING	WARNING	Shot limiting mode at maximum and no longer has thermal control authority.
0x0100003B	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100003C	INTERNAL_FAULT	ERROR	Internal fault detected; unit will restart to attempt recovery.
0x0100003D	INTERNAL_FAULT	ERROR	Internal fault detected; unit will restart to attempt recovery.
0x0100003E	INTERNAL_FAULT	ERROR	Internal fault detected; unit will restart to attempt recovery.
0x0100003F	INTERNAL_COMM	ERROR	Unit has experienced an internal COMM error; please contact Ouster at https://ouster.com/tech-support.
0x01000040	INTERNAL_FAULT	ERROR	After restart attempts, unit did not recover. Going to error state.
0x01000041	INTERNAL_COMM	WARNING	Unit has experienced an internal COMM warning: some measurements may have been skipped.
0x01000042	INTERNAL_COMM	ERROR	Unit has experienced an internal COMM error; please contact Ouster at https://ouster.com/tech-support.
0x01000043	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000044	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.

continues on next page

Table 20.1 - continued from previous page

id	category	level	msg
0x01000045	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000046	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000047	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000048	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x01000049	INTERNAL_FW	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100004A	STARTUP	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100004B	STARTUP	ERROR	Unit has experienced a startup error; please contact Ouster at https://ouster.com/tech-support.
0x0100004C	INTERNAL_FAULT	ERROR	Internal fault detected; unit going to error stop state.
0x0100004D	INTERNAL_FAULT	ERROR	Internal fault detected; unit going to error stop state.
0x0100004E	WARMUP_ISSUE	WARNING	Sensor warmup process is taking longer than expected; please ensure sensor is thermally constrained per requirements.
0x0100004F	WARMUP_ISSUE	WARNING	Sensor warmup process is taking longer than expected; please ensure sensor is thermally constrained per requirements.
0x01000050	MOTOR_CONTROL	WARNING	The phase lock offset error has exceeded the threshold.
0x01000051	MOTOR_CONTROL	ERROR	The phase lock control failed to achieve a lock multiple times; please contact Ouster at https://ouster.com/tech-support.
0x01000052	CONFIG_INVALID	ERROR	Configuration value is invalid or out of bounds.
0x01000053	WARMUP_ISSUE	ERROR	Sensor warmup process has failed.

21 Hardware Errata

21.1 MULTIPURPOSE_IO Errata

Current OS1 sensors require an external compensation circuit when the MULTIPURPOSE_IO port is configured as an OUTPUT. Without this circuit, the resulting waveform is a low amplitude pulse train regardless of the pull up voltage. The following example circuit can correct the output waveform. Current Gen2 sensors including the OS1 Gen2 have this fix implemented and thus do no require a buffer circuit.

Buffer circuit fix for MULTIPURPOSE_IO in OUTPUT mode:

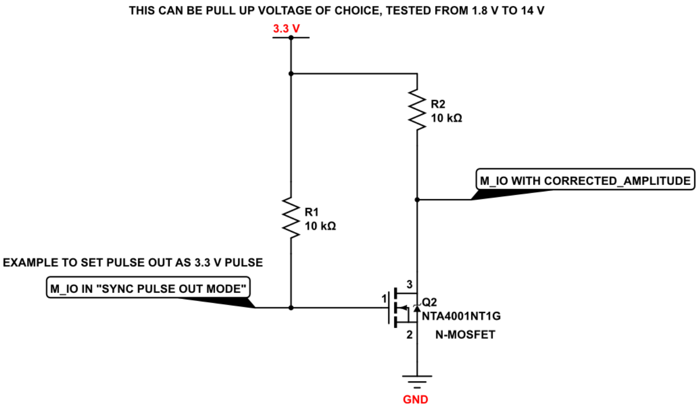

THIS CAN BE PULL UP VOLTAGE OF CHOICE, TESTED FROM 1.8 V TO 14 V
3.3 V
R2
10 kΩ
M_IO WITH CORRECTED_AMPLITUDE
R1
10 kΩ
EXAMPLE TO SET PULSE OUT AS 3.3 V PULSE
M_IO IN "SYNC PULSE OUT MODE"
1
3
2
Q2
NTA4001NT1G
N-MOSFET
GND

Table21.1: MULTIPURPOSE_IO - OUTPUT Buffer Circuit Example

Parameter	M_IO IN SYNC_PULSE_OUT_MODE	SYNC_PULSE_OUT_MODE with corrected amplitude
Example Waveform LOGIC LOW	0 V	0 V
Example Waveform LOGIC HIGH	1.2 V	3.3 V

22 Networking Guide

22.1 Networking 101

Welcome to this guide that will help you understand how to quickly get connected to your sensor to start doing great things with it. When trying to connect to the sensor for the first time there are some basics that need to to be achieved for successful communication between the host machine and the sensor.

We need to ensure that the sensor receives an IP address from the host machine so that we can talk to it. This can be achieved with a few different methods such as DHCP, link-local, static IP. We also need to ensure that the sensor and the host machine are talking on the same subnet.

Once the sensor receives an IP address and is on the correct subnet we can talk to it using its hostname, `os-991234567890.local`, where `991234567890` is the sensor serial number.

If some of this terminology is new to you don't fret, we have defined some of it for you. Here is some basic terminology that will help you digest the steps and be more familiar with networking in general.

IPv4 Address This is the address that can be used to communicate with devices on a network. The format of an IPv4 address is a set of four octets, `xxx.xxx.xxx.xxx` with `xxx` being in the range `0-255`. For example, your host machine Ethernet port may have an address of `192.0.2.1` and your sensor may have an address of `192.0.2.130`.

DHCP (Dynamic Host Configuration Protocol) Server This is a server that may run on your host machine, switch, or router which will serve an IPv4 address to a device that is connected to it. It will ensure that each device connected will have a unique IPv4 address on the network.

Link-local IPv4 Address These are the addresses that are self-assigned between the host machine and a device connected to it in the absence of a DHCP server. They are only valid within the network segment that the host is connected to. The addresses lie within the block `169.254.0.0/16 (169.254.0.0 - 169.254.255.255)`.

Subnet Mask This defines which bits of the IPv4 address are the network prefix and which are the host identifiers. See the table below for an example.

	Binary Form	Decimal-dot notation
IP address	`11000000.00000000.00000010.10000010`	`192.0.2.130`
Subnet mask	`11111111.11111111.11111111.00000000`	`255.255.255.0`
Network prefix	`11000000.00000000.00000010.00000000`	`192.0.2.0`
Host identifier	`00000000.00000000.00000000.10000010`	`0.0.0.130`

Note: Subnet mask can be abbreviate with the number of bits that apply to the network prefix. E.g. /24 for 255.255.255.0 or /16 for 255.255.0.0.

Static IPv4 Address This is when you specify the addresses for the host machine and/or connected device rather than letting the host machine self-assign or using a DHCP server. For example, you may want to specify the host machine IPv4 address to be `192.0.2.100/24` and the sensor to be `192.0.2.200`.

Hostname This is the more human readable name that comes with your sensor. The sensor's hostname is `os-991234567890.local`, where `991234567890` is the sensor serial number.

Note: The `.local` portion of the hostname denotes the local domain used in combination with multicast DNS (mDNS). It is employed when using the sensor in a local network environment with supporting operating system services. This means when the sensor is directly connected to the host machine or if the host machine and sensor are on the same network connected through a router or switch. If you are trying to connect to the sensor on another domain with a supporting DHCP and DNS server configuration you should replace the .local with the domain the sensor is on. For example, if the sensor is connected to a network with domain `ouster-domain.com` the sensor will be reachable on `os-991234567890.ouster-domain.com`.

22.2 Windows

The following steps have been tested on Windows 10. The sensor's hostname is `os-991234567890.local`, where `991234567890` is the sensor serial number.

Connecting the Sensor

1. Connect the sensor to an available Ethernet port on your host machine or router.
2. The sensor will automatically obtain an IP address either through link-local or DHCP (if preconfigured) depending on your network configuration.

Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

The Sensor Homepage

1. Type `os-991234567890.local`/ in the address bar of your browser to view the sensor homepage

Note: If you are unable to load the sensor homepage, follow the steps in *Determining the IPv4 Address of the Sensor* to verify your sensor is on the network and has a valid IPv4 address.

Determining the IPv4 Address of the Sensor

1. Open a command prompt on the host machine by pressing **Win+X** and then **A**
2. Use the ping command to determine the IPv4 address of the sensor

Command

```
ping -4 [sensor_hostname]
```

Example

```
C:WINDOWSsystem32>ping -4 os-991234567890.local
```

Note: If this command hangs you may need to go back and configure you interface to link-local in the section *Connecting the Sensor*

Response

```
Pinging os-991234567890.local [169.254.0.123] with 32 bytes of data:
Reply from 169.254.0.123: bytes=32 time<1ms TTL=64
Reply from 169.254.0.123: bytes=32 time<1ms TTL=64
Reply from 169.254.0.123: bytes=32 time<1ms TTL=64
Reply from 169.254.0.123: bytes=32 time<1ms TTL=64

Ping statistics for 169.254.0.123:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
 Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

3. You can also browse for the sensor IPv4 address using dns-sd and the sensor hostname. Learn more about this in *Finding a Sensor with mDNS Service Discovery*

Command

```
dns-sd -G v4 [sensor_hostname]
```

Example

```
C:WINDOWSsystem32>dns-sd -G v4 os-991234567890.local
```

Response

```
Timestamp     A/R  Flags   if   Hostname                Address        TTL
14:22:46.897  Add  2       6    os-991234567890.local  169.254.0.123  120
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

Determining the IPv4 Address of the Interface

1. Open a command prompt by pressing **Win+X** and then **A**
2. View the IPv4 address of your interfaces

Command

```
netsh interface ip show config
```

Example

```
C:WINDOWSsystem32>netsh interface ip show config
```

Response

```
Configuration for interface "Local Area Connection"
    DHCP enabled:                         Yes
    IP Address:                           169.254.0.1
    Subnet Prefix:                        169.254.0.0/16 (mask 255.255.0.0)
    InterfaceMetric:                      25
    DNS servers configured through DHCP:  None
    Register with which suffix:           Primary only
    WINS servers configured through DHCP: None

Configuration for interface "Loopback Pseudo-Interface 1"
    DHCP enabled:                         No
    IP Address:                           127.0.0.1
    Subnet Prefix:                        127.0.0.0/8 (mask 255.0.0.0)
    InterfaceMetric:                      75
    Statically Configured DNS Servers:    None
    Register with which suffix:           Primary only
    Statically Configured WINS Servers:   None
```

- In this example, your sensor is plugged into interface "Local Area Connection"
- Your host IPv4 address will be on the line that starts with IP Address: In this case it is `169.254.0.1`

Note: If your interface IPv4 address is of the form `169.254.x.x`, it is connected via link-local to the sensor. This means that Windows self-assigned an IP address in the absence of a DHCP server.

Setting the Host Interface to DHCP

Use this to set your interface to automatically obtain an IP address via DHCP. This is useful for architectures that need to be more plug and play.

Set your interface to DHCP

Command

```
netsh interface ip set address ["Network Interface Name"] dhcp
```

Example with interface name `"Local Area Connection"`

```
C:WINDOWSsystem32>netsh interface ip set address "Local Area Connection" dhcp
```

Response blank

Setting the Host Interface to Static IP

Use this to set your interface to be assigned a static IPv4 address. This is useful for controlling the IP address that the sensor will be sending data to.

Set your interface to static

Command

```
netsh interface ip set address name="Network Interface Name" static [IP address] [Subnet Mask]
↪[Gateway]
```

Example with interface name “Local Area Connection” and IPv4 address 192.0.2.1/24.

```
C:WINDOWSsystem32>netsh interface ip set address name="Local Area Connection" static
↪192.0.2.1/24
```

Note: The `/24` is shorthand for Subnet Mask = `255.255.255.0`

Response blank

Finding a Sensor with mDNS Service Discovery

The sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named `_roger._tcp`. You can use service discovery tools such as dns-sd (Windows/macOS) to find all sensors connected to the network.

Note: If your version of Windows does not have dns-sd on the command line you can install it by downloading the Bonjour SDK for Windows (available through Apple or Softpedia)

1. Find all sensors and their associated service text on a network.

   Command

   ```
   dns-sd -Z [service type]
   ```

   Example

   ```
   C:WINDOWSsystem32> dns-sd -Z _roger._tcp
   ```

   Response

   ```
   Browsing for _roger._tcp

   ; To direct clients to browse a different domain, substitute that domain in place of
   ↪'@'
   lb._dns-sd._udp                                 PTR     @

   ; In the list of services below, the SRV records will typically reference dot-local
   ↪Multicast DNS names.
   ```

```
; When transferring this zone file data to your unicast DNS server, you'll need to
↪replace those dot-local
; names with the correct fully-qualified (unicast) domain name of the target host
↪offering the service.

_roger._tcp                                      PTR     Ouster032Sensor032
↪991234567890._roger._tcp
Ouster032Sensor032 991234567890._roger._tcp    SRV     0 0 7501 os-991234567890.local.
↪; Replace with unicast FQDN of target host
Ouster032Sensor032 991234567890._roger._tcp    TXT     "pn=840-102145-B" "sn=
↪991234567890" "fw=ousteros-image-prod-aries-v2.0.0-20200417193957"
```

2. Browse for the sensor IPv4 address using dns-sd and the sensor hostname.

   Command

   ```
   dns-sd -G v4 [sensor_hostname]
   ```

   Example

   ```
   C:WINDOWSsystem32>dns-sd -G v4 os-991234567890.local
   ```

   Response

   ```
   Timestamp     A/R  Flags   if   Hostname                Address        TTL
   14:22:46.897  Add  2       6    os-991234567890.local.  169.254.0.123  120
   ```

   Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`

22.3 macOS

The following steps have been tested on macOS 10.15.4. The sensor's hostname is `os-991234567890.local`, where `991234567890` is the sensor serial number.

Connecting the Sensor

1. Connect the sensor to an available Ethernet port on your host machine or router.
2. The sensor will automatically obtain an IP address either through link-local or DHCP (if preconfigured) depending on your network configuration.

Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

The Sensor Homepage

1. Type `os-991234567890.local`/ in the address bar of your browser to view the sensor homepage

Note: If you are unable to load the sensor homepage, follow the steps in *Determining the IPv4 Address of the Sensor* to verify your sensor is on the network and has a valid IPv4 address.

Determining the IPv4 Address of the Sensor

1. Open a Terminal window on the host machine by pressing **CMD+SPACE** and typing **Terminal** in the search bar, then press enter.
2. Use the ping command to determine the IPv4 address of the sensor

   Command

   ```
   ping -c3 [sensor_hostname]
   ```

   Example

   ```
   Mac-Computer:~ username$ ping -c3 os-991234567890.local
   ```

   Note: If this command hangs you may need to go back and configure you interface to link-local in the section *Connecting the Sensor*

   Response

   ```
   PING os-991234567890.local (169.254.0.123): 56 data bytes
   64 bytes from 169.254.0.123: icmp_seq=0 ttl=64 time=0.644 ms
   64 bytes from 169.254.0.123: icmp_seq=1 ttl=64 time=0.617 ms
   64 bytes from 169.254.0.123: icmp_seq=2 ttl=64 time=0.299 ms

   --- os-991234567890.local ping statistics ---
   3 packets transmitted, 3 packets received, 0.0% packet loss
   round-trip min/avg/max/stddev = 0.299/0.520/0.644/0.157 ms
   ```

   Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

3. You can also browse for the sensor IPv4 address using dns-sd and the sensor hostname. Learn more about this in *Finding a Sensor with mDNS Service Discovery*

   Command

   ```
   dns-sd -G v4 [sensor_hostname]
   ```

   Example

   ```
   Mac-Computer:~ username$ dns-sd -G v4 os-991234567890.local
   ```

Response

```
DATE: ---Tue 28 Apr 2020---
11:40:43.228  ...STARTING...
Timestamp     A/R  Flags  if  Hostname                 Address          TTL
11:40:43.414  Add  2      18  os-991234567890.local.  169.254.0.123     120
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

Determining the IPv4 Address of the Interface

This will help you find the IPv4 address of the interface that you have plugged the sensor into. It is helpful to know which interface you have plugged into, e.g. `en1` in the example below.

1. Open a Terminal window on the host machine by pressing **CMD+SPACE** and typing **Terminal** in the search bar, then press enter.
2. View the IPv4 address of your interfaces

Command

```
ifconfig
```

Example

```
Mac-Computer:~ username$ ifconfig
```

Response

```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
   options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
   inet 127.0.0.1 netmask 0xff000000
   inet6 ::1 prefixlen 128
   inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
   nd6 options=201<PERFORMNUD,DAD>
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
   options=400<CHANNEL_IO>
   ether 38:f9:d3:d6:33:8a
   inet6 fe80::1c30:1246:93a2:9f68%en0 prefixlen 64 secured scopeid 0x7
   inet 192.0.2.7 netmask 0xffffff00 broadcast 192.0.2.255
   nd6 options=201<PERFORMNUD,DAD>
   media: autoselect
   status: active
en1: flags=8963<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
   options=400<CHANNEL_IO>
   ether 48:65:ee:1d:22:35
   inet6 fe80::c27:1917:47ed:bcfe%en1 prefixlen 64 secured scopeid 0x12
   inet 169.254.0.1 netmask 0xffff0000 broadcast 169.254.255.255
   nd6 options=201<PERFORMNUD,DAD>
   media: autoselect (1000baseT <full-duplex>)
```

```
status: active
```

- In this example, your sensor is plugged into interface `en1`
- Your host IPv4 address will be on the line that starts with `inet`: In this case it is `169.254.0.1`

Note: If your interface IPv4 address is of the form `169.254.x.x`, it is connected via link-local to the sensor. This means that Windows self-assigned an IP address in the absence of a DHCP server.

Setting the Host Interface to DHCP

Use this to set your interface to automatically obtain an IP address via DHCP. This is useful for architectures that need to be more plug and play.

Set your interface to DHCP

Command

```
sudo ipconfig set [interface_name] DHCP
```

Example with interface name `en1`

```
Mac-Computer:~ username$ sudo ipconfig set en1 DHCP
```

Response blank, however you can verify the change has been made with the `ifconfig` command. The `inet` line will be blank if nothing is plugged in or shows the DHCP or link-local self-assigned IPv4 address. E.g. `169.254.0.1`

```
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500

options=6407<RXCSUM,TXCSUM,VLAN_MTU,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
   ether 48:65:ee:1d:22:35
   inet6 fe80::1c24:5e0a:2ea8:12e9%en1 prefixlen 64 secured scopeid 0x7
   inet 169.254.0.1 netmask 0xffff0000 broadcast 169.254.255.255
   nd6 options=201<PERFORMNUD,DAD>
   media: autoselect (1000baseT <full-duplex>)
   status: active
```

Setting the Host Interface to Static IP

Use this to set your interface to be assigned a static IPv4 address. This is useful for controlling the IP address that the sensor will be sending data to.

Set your interface to static

Command

```
sudo ipconfig set [interface_name] MANUAL [ip_address] [subnet_mask]
```

Example with interface name en1 and IPv4 address 192.0.2.1 and subnet mask 255.255.255.0.

```
Mac-Computer:~ username$ sudo ipconfig set en1 MANUAL 192.0.2.1 255.255.255.0
```

Note: The /24 is shorthand for Subnet Mask = 255.255.255.0

Response blank, however you can verify the change has been made with the ifconfig command. The inet line will show the static IPv4 address. e.g. 192.0.2.1.

```
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500

options=6407<RXCSUM,TXCSUM,VLAN_MTU,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
   ether 48:65:ee:1d:22:35
   inet6 fe80::1c24:5e0a:2ea8:12e9%en1 prefixlen 64 secured scopeid 0x7
   inet 192.0.2.1 netmask 0xffffff00 broadcast 192.0.2.255
   nd6 options=201<PERFORMNUD,DAD>
   media: autoselect (1000baseT <full-duplex>)
   status: active
```

Finding a Sensor with mDNS Service Discovery

The sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named _roger._tcp. You can use service discovery tools such as dns-sd (Windows/macOS) to find all sensors connected to the network.

1. Find all sensors and their associated service text on a network.

   Command

   ```
   dns-sd -Z [service type]
   ```

   Example

   ```
   Mac-Computer:~ username$ dns-sd -Z _roger._tcp
   ```

   Response

   ```
   Browsing for _roger._tcp
   DATE: ---Thu 30 Apr 2020---
   17:27:52.242  ...STARTING...

   ; To direct clients to browse a different domain, substitute that domain in place of
   ↪'@'
   lb._dns-sd._udp                                  PTR     @

   ; In the list of services below, the SRV records will typically reference dot-local
   ↪Multicast DNS names.
   ; When transferring this zone file data to your unicast DNS server, you'll need to
   ↪replace those dot-local
   ; names with the correct fully-qualified (unicast) domain name of the target host
   ↪offering the service.
   ```

```
_roger._tcp                                    PTR     Ouster Sensor 991234567890._
↪roger._tcp
Ouster Sensor 991234567890._roger._tcp    SRV     0 0 7501 os-991234567890.local. ;⊠
↪Replace with unicast FQDN of target host
Ouster Sensor 991234567890._roger._tcp    TXT     "pn=840-102145-B" "sn= 991234567890"⊠
↪"fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn= 991234567890"
^C
```

2. Browse for the sensor IPv4 address using dns-sd and the sensor hostname.

   Command

   ```
   dns-sd -G v4 [sensor_hostname]
   ```

   Example

   ```
   Mac-Computer:~ username$ dns-sd -G v4 os-991234567890.local
   ```

   Response

   ```
   DATE: ---Thu 30 Apr 2020---
   17:37:33.155  ...STARTING...
   Timestamp     A/R  Flags  if  Hostname                Address         TTL
   17:37:33.379  Add  2      7   os-991234567890.local.  169.254.0.123   120
   ```

   Note: In this example, your sensor IPv4 address is determined to be 169.254.0.123

22.4 Linux

The following steps have been tested on Ubuntu 18.04. The sensor's hostname is os-991234567890.local, where 991234567890 is the sensor serial number.

Connecting the Sensor

1. Connect the sensor to an available Ethernet port on your host machine or router.
2. The sensor will automatically obtain an IP address either through link-local or DHCP (if preconfigured) depending on your network configuration.
3. If directly connecting to the host machine you may need to set your Ethernet interface to Link-Local Only mode. This can be done via the command line or GUI. See instructions in *Setting the Interface to Link-Local Only*

   Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

Setting the Interface to Link-Local Only

Via Command Line

Command

```
nmcli con modify [interface_name] ipv4.method link-local ipv4.addresses ""
```

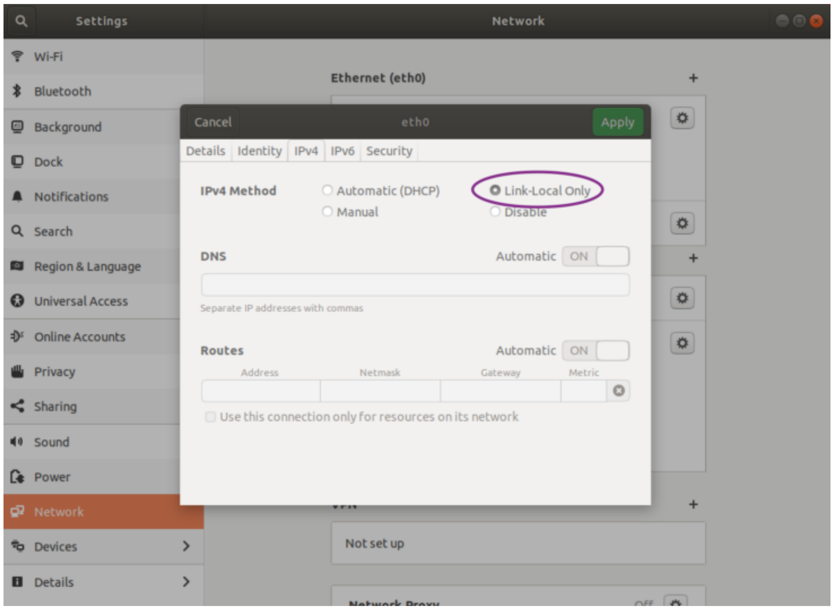
Example with interface name `eth0` and IPv4 address `""`.

```
username@ubuntu:~$ nmcli con modify eth0 ipv4.method link-local ipv4.addresses ""
```

Response blank, however you can verify the change has been made with the `ip addr` command. The `inet` line for the interface `eth0` will show the link-local IPv4 address automatically negotiated once the sensor is reconnected to the interface. e.g. `169.254.0.1`.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen⏎
↪1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group⏎
↪default qlen 1000
    link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.1/16 brd 169.254.255.255 scope link noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group⏎
↪default qlen 1000
    link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
    inet 172.16.79.232/24 brd 172.16.79.255 scope global wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe28:7a8a/64 scope link
       valid_lft forever preferred_lft forever
```

Via GUI The image below illustrates how to se the interface to `Link-Local Only` mode using the graphical user interface.



IPv4 Method ○ Automatic (DHCP) ○ Manual ● Link-Local Only ○ Disable

DNS Automatic ON

Separate IP addresses with commas

Routes Automatic ON

Address	Netmask	Gateway	Metric

□ Use this connection only for resources on its network

Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

The Sensor Homepage

1. Type `os-991234567890.local`/ in the address bar of your browser to view the sensor homepage

   Note: If you are unable to load the sensor homepage, follow the steps in *Determining the IPv4 Address of the Sensor* to verify your sensor is on the network and has a valid IPv4 address.

Determining the IPv4 Address of the Sensor

1. Open a Terminal window on the host machine by pressing **Ctrl+Alt+T**.
2. Use the `ping` command to determine the IPv4 address of the sensor

   Command

   ```
   ping -4 -c3 [sensor_hostname]
   ```

   Example

   ```
   username@ubuntu:~$ ping -4 -c3 os-991234567890.local
   ```

   Note: If this command hangs you may need to go back and configure you interface to link-local in the section *Setting the Interface to Link-Local Only*

Response

```
PING os-991234567890.local (169.254.0.123) 56(84) bytes of data.
64 bytes from os-991234567890.local (169.254.0.123): icmp_seq=1 ttl=64 time=1.56 ms
64 bytes from os-991234567890.local (169.254.0.123): icmp_seq=2 ttl=64 time=0.893 ms
64 bytes from os-991234567890.local (169.254.0.123): icmp_seq=3 ttl=64
time=0.568 ms

--- os-991234567890.local ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2025ms
rtt min/avg/max/mdev = 0.568/1.008/1.565/0.416 ms
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

3. You can also browse for the sensor IPv4 address using `avahi-browse` and the sensor service type, which is `_roger._tcp`. Learn more about this in *Finding a Sensor with mDNS Service Discovery*

Command

```
avahi-browse -lrt [service type]
```

Example

```
username@ubuntu:~$ avahi-browse -lrt _roger._tcp
```

Response

```
+   eth0 IPv6 Ouster Sensor 991234567890                    _roger._tcp          local
+   eth0 IPv4 Ouster Sensor 991234567890                    _roger._tcp          local
=   eth0 IPv6 Ouster Sensor 991234567890                    _roger._tcp          local
   hostname = [os-991234567890.local]
   address = [fe80::be0f:a7ff:fe00:1852]
   port = [7501]
   txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn=99201000067
8" "pn=840-102145-B"]
=   eth0 IPv4 Ouster Sensor 991234567890                    _roger._tcp          local
   hostname = [os-991234567890.local]
   address = [169.254.0.123]
   port = [7501]
   txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn= 991234567890"⊠
↪"pn=840-102145-B"]
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

Determining the IPv4 Address of the Interface

This will help you find the IPv4 address of the interface that you have plugged the sensor into. It is helpful to know which interface you have plugged into, e.g. `eth0` in the example below.

1. Open a Terminal window on the host machine by pressing **Ctrl+Alt+T**.
2. View the IPv4 address of your interfaces

   Command

   ```
   ip addr
   ```

   Example

   ```
   username@ubuntu:~$ ip addr
   ```

   Response

   ```
   1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
   ↪1000
       link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
       inet 127.0.0.1/8 scope host lo
          valid_lft forever preferred_lft forever
       inet6 ::1/128 scope host
          valid_lft forever preferred_lft forever
   2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
   ↪default qlen 1000
       link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
       inet 169.254.0.1/16 brd 169.254.255.255 scope link noprefixroute eth0
          valid_lft forever preferred_lft forever
       inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
          valid_lft forever preferred_lft forever
   3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
   ↪default qlen 1000
       link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
       inet 192.0.2.232/24 brd 192.0.2.255 scope global wlan0
          valid_lft forever preferred_lft forever
       inet6 fe80::250:56ff:fe28:7a8a/64 scope link
          valid_lft forever preferred_lft forever
   4: gpd0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default
   ↪qlen 500
       link/none
   ```

   - In this example, your sensor is plugged into interface `eth0`
   - Your host IPv4 address will be on the line that starts with `inet`: In this case it is `169.254.0.1`

   Note: If your interface IPv4 address is of the form `169.254.x.x`, it is connected via link-local to the sensor. This means that Windows self-assigned an IP address in the absence of a DHCP server.

Setting the Host Interface to DHCP

Use this to set your interface to automatically obtain an IP address via DHCP. This is useful for architectures that need to be more plug and play.

Note: It is recommended that you unplug the cable from the interface prior to making changes to the interface.

Via Command Line

Command

```
nmcli con modify [interface_name] ipv4.method auto ipv4.addresses ""
```

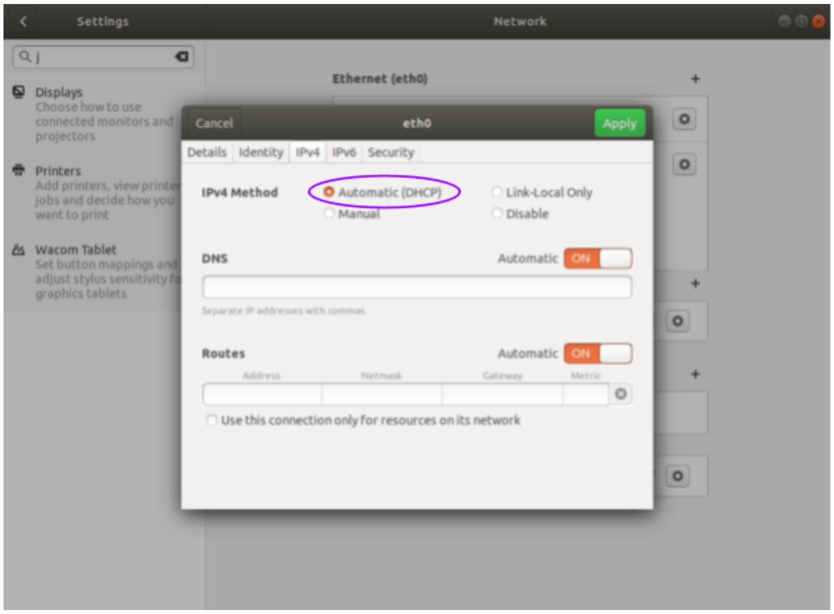
Example with interface name `eth0`

```
username@ubuntu:~$ nmcli con modify eth0 ipv4.method auto ipv4.addresses ""
```

Response blank, however you can verify the change has been made with the `ip addr` command. There will be no `inet` line for the interface `eth0` until you plug in a cable to a device that has a DHCP server to provide an IPv4 address the interface

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
↪qlen 1000
    link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen
↪1000
    link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
    inet 172.16.79.232/24 brd 172.16.79.255 scope global wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe28:7a8a/64 scope link
       valid_lft forever preferred_lft forever
```

Via GUI The image below illustrates how to se the interface to `Automatic (DHCP)` mode using the graphical user interface.


Settings
Network
Displays
Choose how to use connected monitors and projectors
Printers
Add printers, view printer jobs and decide how you want to print
Wacom Tablet
Set button mappings and adjust stylus sensitivity fo graphics tablets
Ethernet (eth0)
Cancel
eth0
Apply
Details
Identity
IPv4
IPv6
Security
Separate IP addresses with commas
Address
Netmask
Gateway
Metric

IPv4 Method ● Automatic (DHCP) ○ Link-Local Only ○ Manual ○ Disable

DNS Automatic ON

Routes Automatic ON

□ Use this connection only for resources on its network

Setting the Host Interface to Static IP

Use this to set your interface to be assigned a static IPv4 address. This is useful for controlling the IP address that the sensor will be sending data to.

Note: It is recommended that you unplug the cable from the interface prior to making changes to the interface.

Via Command Line

Command

```
nmcli con modify [interface_name] ipv4.method manual ipv4.addresses [ip_address]
```

Example with interface name `eth0` and IPv4 address `192.0.2.1/24`.

```
username@ubuntu:~$ nmcli con modify eth0 ipv4.method manual ipv4.addresses 192.0.2.1/24
```

Note: The `/24` is shorthand for Subnet Mask = `255.255.255.0`

Response blank, however you can verify the change has been made with the `ip addr` command. The `inet` line for the interface `eth0` will show the static IPv4 address. e.g. `192.0.2.1`

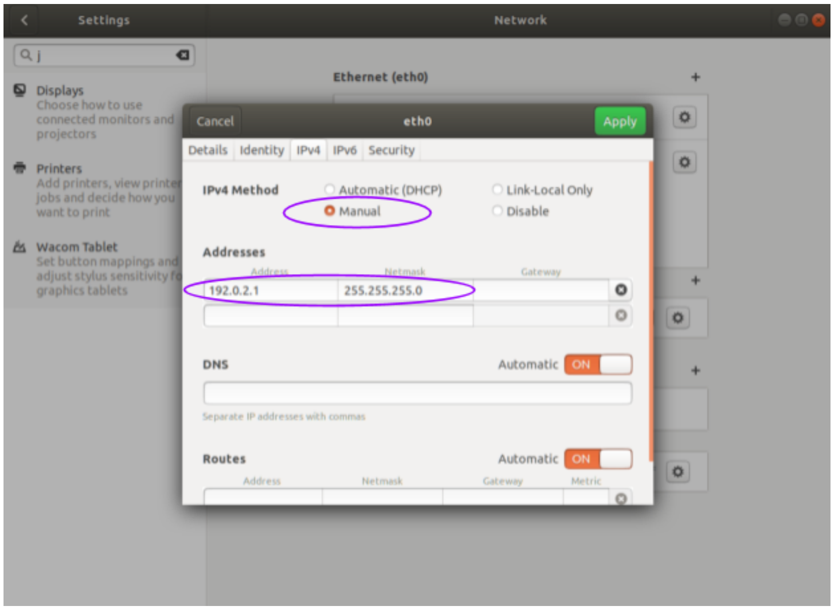
```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
```

(continues on next page)

(continued from previous page)

```
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen
↪1000
    link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen
↪1000
    link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
    inet 172.16.79.232/24 brd 172.16.79.255 scope global wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe28:7a8a/64 scope link
       valid_lft forever preferred_lft forever
```

Via GUI The image below illustrates how to se the interface to `Manual (static)` mode using the graphical user interface.


Settings
Network
Displays
Choose how to use connected monitors and projectors
Printers
Add printers, view printer jobs and decide how you want to print
Wacom Tablet
Set button mappings and adjust stylus sensitivity for graphics tablets
Ethernet (eth0)
Cancel
eth0
Apply
Details
Identity
IPv4
IPv6
Security

IPv4 Method ○ Automatic (DHCP) ● Manual ○ Link-Local Only ○ Disable

Addresses

Address	Netmask	Gateway
192.0.2.1	255.255.255.0	

DNS Automatic ON

Separate IP addresses with commas

Routes Automatic ON

Finding a Sensor with mDNS Service Discovery

The sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named `_roger._tcp`. You can use service discovery tools such as `avahi-browse` (Linux) to find all sensors connected to the network.

1. Find all sensors and their associated service text which includes the sensor IPv4 address using `avahi-browse` and the sensor service type `_roger._tcp`.

   Command

```
avahi-browse -lrt [service type]
```

Example

```
username@ubuntu:~$ avahi-browse -lrt _roger._tcp
```

Response

```
+   eth0 IPv6 Ouster Sensor 991234567890                    _roger._tcp          local
+   eth0 IPv4 Ouster Sensor 991234567890                    _roger._tcp         local
=   eth0 IPv6 Ouster Sensor 991234567890                    _roger._tcp          local
   hostname = [os-991234567890.local]
   address = [fe80::be0f:a7ff:fe00:1852]
   port = [7501]
   txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn=99201000067
8" "pn=840-102145-B"]
=   eth0 IPv4 Ouster Sensor 991234567890                    _roger._tcp          local
   hostname = [os-991234567890.local]
   address = []
   port = [7501]
   txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn= 991234567890"⊠
↪"pn=840-102145-B"]
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`.

23 Ouster Studio

24 Open Source Drivers

Our latest open source drivers are found on www.github.com/ouster. Note that while firmware v1.14 is still in beta, the latest public visualization tools are still meant for Gen1 sensors.

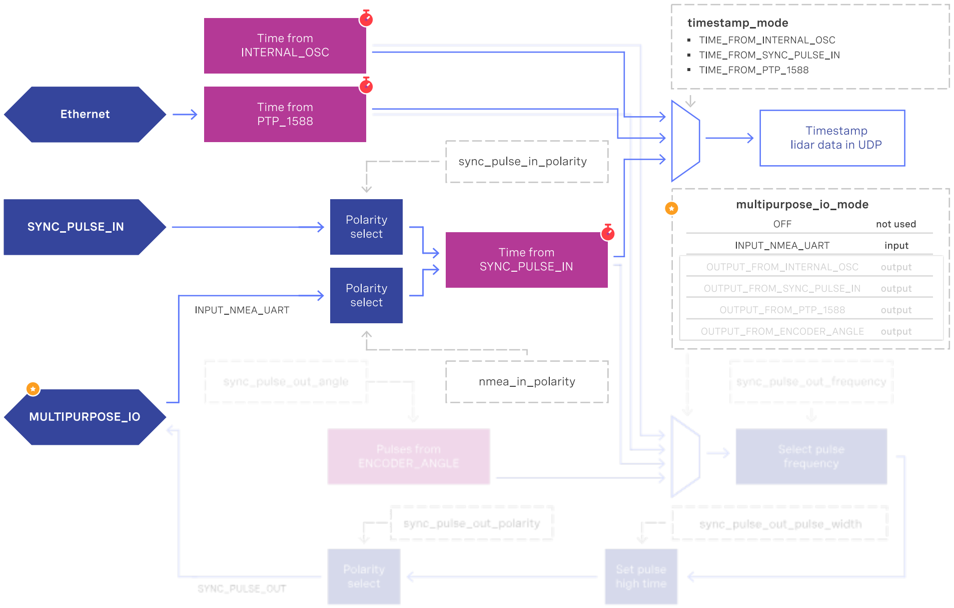
25 OS1 CAD files

All the most up-to-date CAD files of our products can be found on our Lidar Product Details page.

26 Time Synchronization

26.1 Timing Overview Diagram

Signal path with MULTIPURPOSE_IO set as input



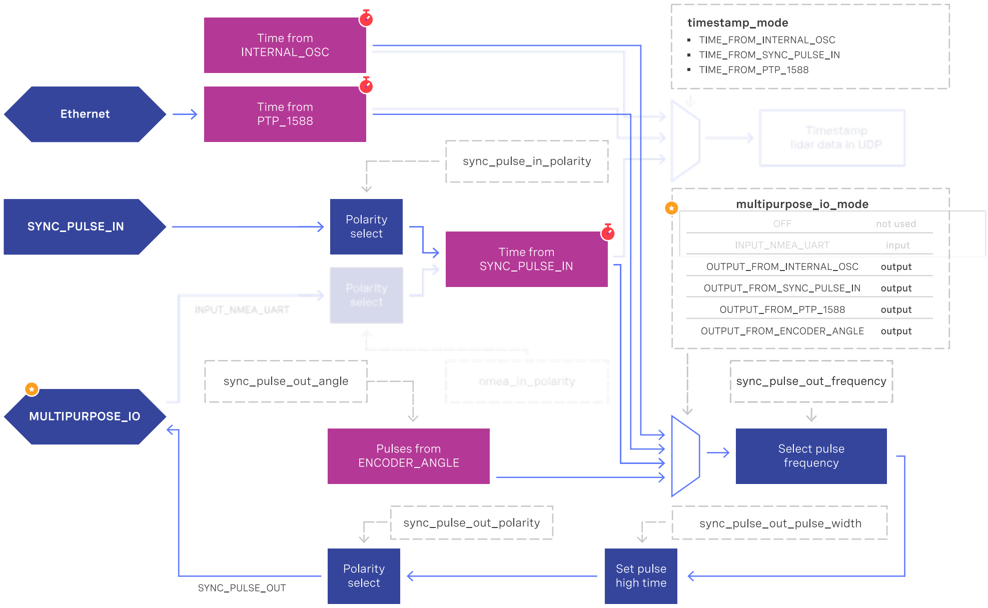
timestamp_mode

- TIME_FROM_INTERNAL_OSC
- TIME_FROM_SYNC_PULSE_IN
- TIME_FROM_PTP_1588

multipurpose_io_mode

OFF	not used
INPUT_NMEA_UART	input
OUTPUT_FROM_INTERNAL_OSC	output
OUTPUT_FROM_SYNC_PULSE_IN	output
OUTPUT_FROM_PTP_1588	output
OUTPUT_FROM_ENCODER_ANGLE	output

Signal path with MULTIPURPOSE_IO set as output



26.2 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.

26.3 Setting Ouster Sensor Time Source

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

Table26.1: 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 the TCP API Guide. 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 26.4. 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 TCP API Guide. 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`

26.4 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 the *TCP API* section. 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 26.3 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 optionalbecause 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 / sec) and the angle is set to 360°, 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 / sec) and the angle is set to 45°. Each full rotation will have 8 pulses. Each pulse is 10 ms wide.

26.5 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 *

27 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

27.1 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.

27.2 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.

27.3 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.

27.4 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.

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
```

(continues on next page)

(continued from previous page)

```
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
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.

27.5 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
```

(continues on next page)

(continued from previous page)

```
                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
```

27.6 Tested Grandmaster Clocks

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

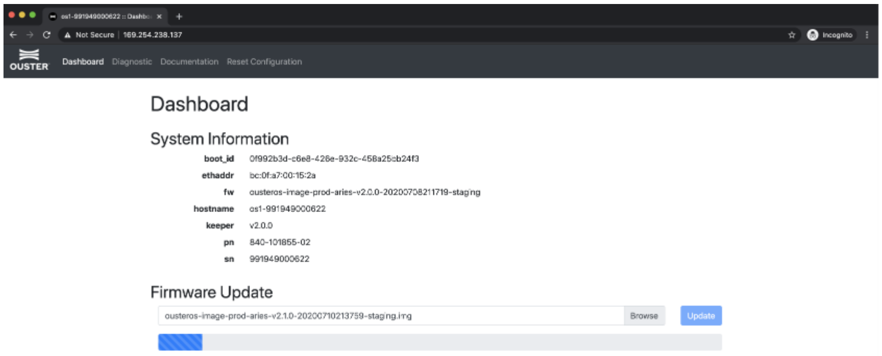
    →

    ```
    > 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`

28 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.



Dashboard

System Information

boot_id	0f992b3d-c6e8-426e-932c-458a25cb24f3
ethaddr	bc:0f:a7:00:15:2a
fw	ousteros-image-prod-aries-v2.0.0-20200708211719-staging
hostname	os1-991949000622
keeper	v2.0.0
pn	840-101855-02
sn	991949000622

Firmware Update

ousteros-image-prod-aries-v2.1.0-20200710213759-staging.img

Figure28.1: Uploading a new firmware image onto the sensor

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.

29 Best Practices

Please check back later for best practices and use cases for using your Ouster sensor.

30 Changelog

HTTP Routing Table