Open-Source Ethernet Networking for Intel® RealSense™ Depth Cameras

Authors: Sergey Dorodnicov, Anders Grunnet-Jepsen, Alexey Puzhevich, Daniel Piro

Revision 0.94

April 2020

Contents

1 Introduction

1.1 About This Document

This document presents a step-by-step guide for how to enable Intel® RealSense™ depth cameras to be networked over an ethernet or WiFi connection, as depicted in Figure 1. It describes an open-source reference design that is meant to be easy to replicate with off-the-shelf components and free software. The presented architecture is intended for rapid prototyping and evaluation and does not currently offer the maturity and full feature set needed to serve as a production solution. There are several excellent alternative 3rd-party products available for those who are interested in mature industrialized Intel® RealSense™ cameras with ethernet interfaces [1] [2]. The goal of this project is to offer an alternative solution that allows users to network existing Intel® RealSense™ depth cameras D415, D435 and D435i (and in future more Intel® RealSense™ cameras), with nearly seamless integration with the Intel® RealSense™ SDK 2.0 and Intel® RealSense™ Viewer, versions 2.34 and above.

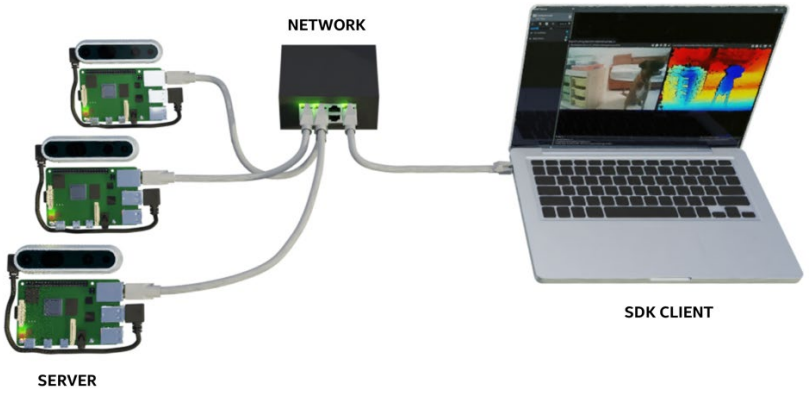

Figure 1. Components of the Intel® RealSense™ Camera over ethernet reference design

We will be covering the basic architecture, the reference design parts, and the source code, as well as examples of how to use it. We will then discuss the expected performance, as well as avenues for future improvements.

1.2 Motivation

USB and Ethernet standards offer different advantages, as highlighted in Table 1:

	USB	Ethernet
Speed	480 Mbps for USB2 5000 Mbps for USB3	100 Mbps or 1000 Mbps for GigE
Length	Up to 5 meters Shorter cable is strongly recommended	Up to 100 meters
Power	5V Up to 2.5W	Requires special PoE equipment and step-down from ~50V to 5V
Sensitivity	Sensitive to cable movement and electromagnetic interference from lightning or industrial equipment	Sufficiently robust for most industrial applications

Table 1: Advantages and disadvantages of USB and Ethernet interfaces

While the standard high-speed USB interface of the Intel® RealSense™ cameras serves most applications very well, many usages require an interface that allows for longer cable lengths, remote access via network, and robustness to repeated disconnection / reconnection to different PC clients. In particular, these requirements are prevalent in robotics and industrial applications.

Figure 2. Industrial and Robotic Applications of Intel® RealSense™ Technology. From left to right: HEBI Robotics[17], Kinova[14], PUDU Robotics[15], ANYmal C by Anybotics[16]

2 Quick Start

2.1 List of Components

	Component	Description	Link
	**9-inch 90-degree Type-A to USB Type-C Cable**	Right-angle short cable is recommended for this project to save space when using with provided case	example link
	**Standard CAT5 Ethernet Cable**		example link
	**microSD Card**	SD-Card is needed to operate the Raspberry Pi. 16GB and up is recommended	example link
	**Raspberry Pi 4 Model B 4GB**	We are using Raspberry Pi 4 in this project due to its versatility and popularity. This guide can be applied with minor modifications to other compute boards [3]	official link
	**Intel® RealSense™ Camera D415 / D435 / D435i**	Provided case and software were designed to work out-of-the-box with D415, D435 and D435i cameras	official link
	**Raspberry Pi PoE Hat** *(optional)*	PoE Hat is required in order to enable Power-over-Ethernet capability. Without it, the project will have the same functionality but will require external power source via Raspberry Pi USB-C connector	official link

6 x M5 5mm Screws

Custom 3D printed case

(optional)

We provide custom Raspberry Pi case schematics with mounting points for Intel® RealSense™ Camera. The case is ready for print and can be ordered online via one of commercial 3D-printing services

download link

2.2 List of Tools

Figure 4: Getting started with Networking Intel RealSense Cameras

- **Windows / Linux / Mac OS Laptop or Desktop PC**
- **Phillips Screwdriver**
- **microSD Card Reader** (example link)
- (*optional*) **PoE Switch** (example link)
  Power-over-Ethernet is not supported by most consumer routers, so in order to utilize PoE capability compatible network switch is required.

The size and cost of this project can be significantly reduced by using camera modules (example link) and replacing USB connector by soldering USB wires directly to the PCB. To simplify assembly this guide assumes off-the-shelf Intel® RealSense™ Cameras and requires no soldering skills

2.3 Preparing the SD-Card

Raspberry Pi 4 requires microSD-card with compatible software to work.
Latest official software is available at raspberrypi.org.

Intel® RealSense™ SDK can be installed using the following instructions.

As reference, a pre-built image is available (download link).

After downloading and extracting the image, use balenaEtcher to flash it to the SD-card:

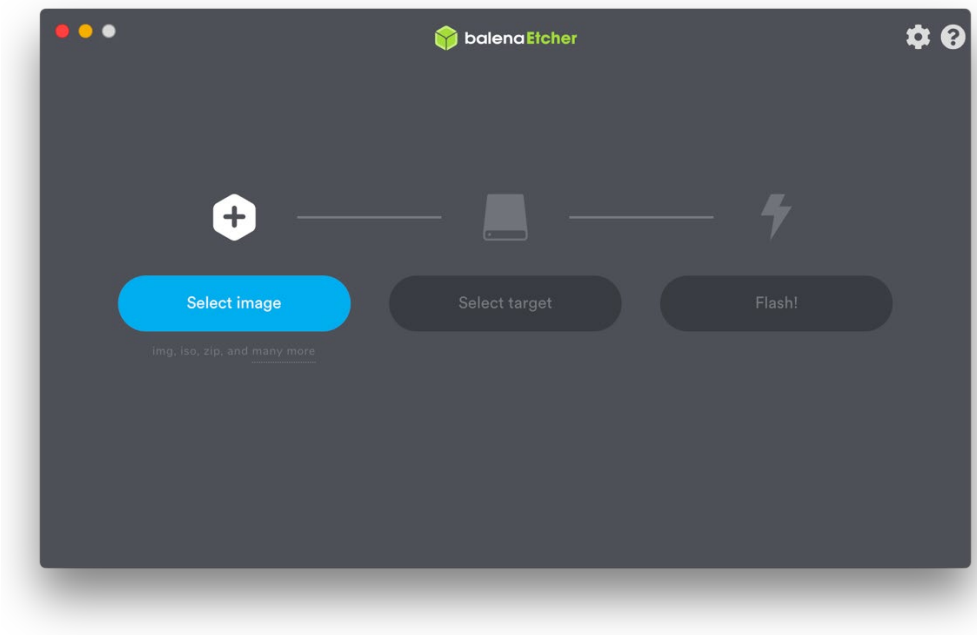

Figure 5: BalenaEtcher is free and open-source software for Windows, Linux, and Mac OS

2.4 Assembly Steps

Figure 6. Steps for assembling an ethernet-ready Intel RealSense Camera

2.5 First Boot

For initial configuration, Raspberry Pi can be connected to a monitor, keyboard and mouse. Alternatively, it is possible to connect the device directly to a laptop / desktop PC via Raspberry Pi USB Type-C port [4]:

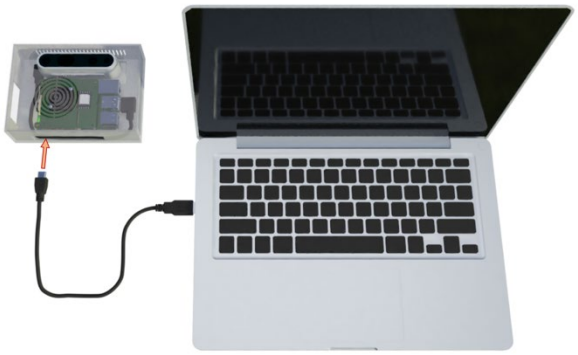

Figure 7. A PC can be directly connected to the Raspberry Pi via the USB3 type-C port

For this method to work the cable must be USB Type-A to USB Type-C.
Type-C to Type-A to Type-C will also work is some cases, but Type-C to Type-C will not work due to Pi's USB hardware limitation

After about 30 seconds, host computer should detect new network named "PI4 USB Device".
It is then possible to establish remote terminal session to the device:

`ssh realsense@10.55.0.1` (On Windows use Putty)

Initial password and username are `realsense`.

First things first:
Start by running `passwd` *command to change the default password from* realsense *to a new strong password*

Raspberry Pi 4 offers number of networking solutions – first and foremost there is the gigabit ethernet port, but there is also Wi-Fi and limited network over USB that we used for initial configuration. In this guide we'll be setting-up ethernet connection with a static IP address, but it is worth noting there are other ways to utilize the project.

2.6 Setting Static IP

While logged-in to the Pi via secure shell, enter:
`sudo nano /etc/dhcpcd.conf`

After entering your password, navigate to "Example static IP configuration" section:

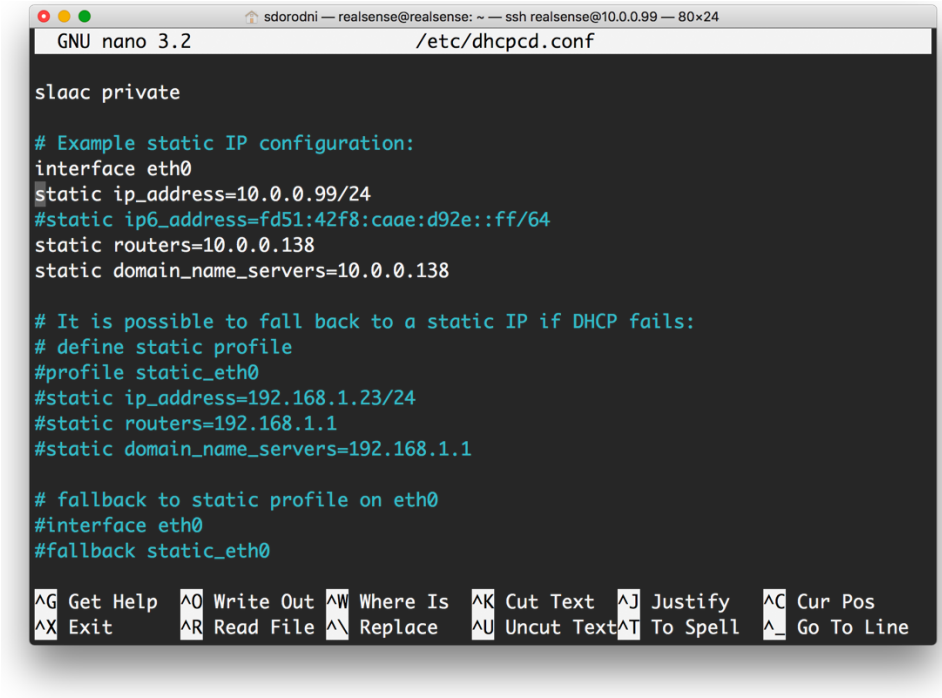

Figure 8. Screen showing how to set a static IP address for the Raspberry Pi

Uncomment `ip_address`, `routers` and `domain_name_servers` lines.
Enter new intended static IP address in ip_address line. The line must end with `/24`, do not remove it. Enter the IP address of your network router under `routers` and `domain_name_servers`.

To find out your router IP, on Linux type-in `ip route | grep default` *in a new terminal window. On Windows, run* `ipconfig` *and look for Default Gateway, and on a Mac – run* `netstat -nr | grep default`

Press `Control + X, Y` to save changes.
Restart the system (using `sudo reboot`) to apply the new configuration.

2.7 Testing the Camera

Download and run Intel® RealSense™ Viewer on the host PC (instructions for Windows, Linux, Mac)

Click `Add Source` > `Add Network Device`. Enter camera IP address:

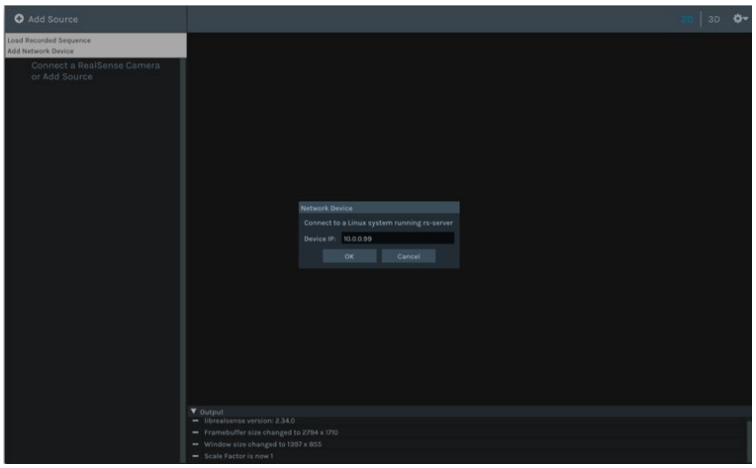

Figure 9. Configuring the Intel® RealSense™ Viewer to access the ethernet-enabled depth camera

Click Ok, and toggle Depth and RGB sensors on:

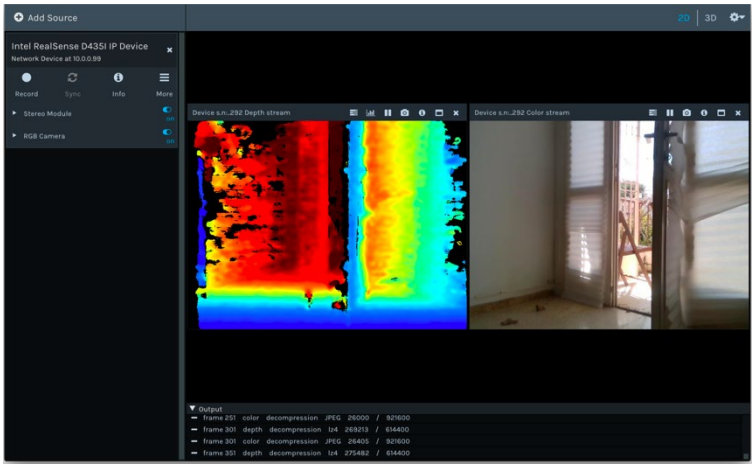

Figure 10. Example of streaming depth and color from an ethernet-enabled device

The device should work similar to standard Intel® RealSense™ camera, with depth, infrared and color streams available, standard sensor controls, point-cloud and texture mapping in the 3D view. For list of software limitations refer to section 3.5.

3 SDK Integration

3.1 Software Overview

Intel® RealSense™ SDK 2.34 has introduced two new software components –

1. `realsense2-net` module
   The module exports single function `rs2_create_net_device` and encapsulates all dependencies required for working with network cameras
2. `rs-server` tool
   This application will connect to first available Intel RealSense camera and listen for incoming connection requests

No network code is compiled into core `realsense2` *module, limiting code size and security risks for customers who don't need this functionality*

3.2 Building from Source

Intel® RealSense™ SDK 2.0 is open-source and cross-platform software. The Network module follows the same guidelines and can be built from source as well.

In order to build `realsense2-net` module and the `rs-server` tool,
add `-DBUILD_NETWORK_DEVICE=ON` to the list of CMake flags when configuring the project.
When building for the Raspberry Pi
`-DFORCE_RSUSB_BACKEND=ON` flag is also strongly recommended.

At the moment, `rs-server` can be only compiled on Linux but `realsense2-net` works on Windows, Linux and Mac OS. Integration with python, Android and other wrappers in librealsense ecosystem will be provided in future releases based on feedback from the community.

3.3 Software Architecture

The project uses industry standard RTP protocol for video streaming.

In addition, RTSP protocol is used for camera control. Networking is handled by Live555 – open-source cross-platform library for low-power multimedia streaming[5]. Depth compression is done using LZ4 [6], color and infrared streams are compressed using libjpeg-turbo open-source library [7].

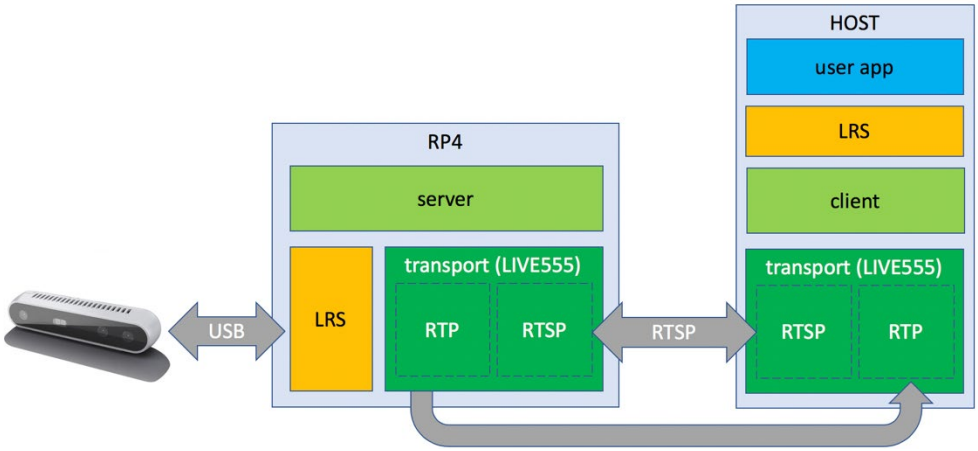

Figure 11. Ethernet-enabled camera software architecture

3.4 SDK Examples

Existing SDK examples and Apps will not work with a network device out of the box since network connection needs to be explicitly established.

That said, modifying existing code to work with a network camera is easy:

1. Add `#include <librealsense2-net/rs_net.hpp>` to the code
2. Inside `CMakeLists.txt` add `realsense2-net` to `target_link_libraries`
3. Create network device from IP and add it to a context:
   ```
   rs2::net_device dev("10.0.0.99");
   rs2::context ctx; dev.add_to(ctx);
   ```
4. Pass context to pipeline constructor:
   ```
   rs2::pipeline pipe(ctx);
   ```

No additional code changes should be necessary. From that point the SDK should treat network camera like any other Intel® RealSense™ camera.

3.5 Design Considerations

This section outlines the considerations for key design decisions in this project.

3.5.1 RTP/RTSP

RTP/RTSP protocols are used for streaming and control [26].

ROS ecosystem also offers good networking capabilities and may be a better choice if the client is also using ROS [10]. ROS is, however, less portable, requires larger download and does not integrate easily into existing SDK applications.

Another significant alternative is GigE Vision – a well-known industry format for network cameras [11]. RTP/RTSP was chosen because of availability of lightweight, permissively open-sourced and portable implementation.

3.5.2 USB2 vs USB3

Different types of USB interface offer different advantages with USB3 bandwidth being significantly higher allowing for higher resolutions, and USB2 cables being more widespread and less sensitive to electromagnetic interference [9]

3.5.3 Compression

Depth data is compressed using LZ4 algorithm. Depth does not follow some of the assumptions of common lossy compression algorithm and using these algorithms on depth data can have destructive effects on 3D applications. This is why lossless algorithm was chosen [21].

Color and Infrared data are compressed using JPEG algorithm. While video compression algorithms such as MPEG can offer better compression rates, they are sensitive to frame drops [12]. In addition, ffmpeg is industry standard for MPEG compression, but it is not as permissively licensed as other SDK components [13].

3.6 Software Limitations

At the moment, the following features are not yet available for network device:

1. D400 Advanced Mode API
2. Dynamic and Self Calibration
3. Firmware Update API
4. IMU streaming support
5. Integration with SDK Wrappers
6. Post-processing

In addition, to better match ethernet bandwidth limitation `rs-server` is only exposing the following camera modes:

Format	Resolution	Framerate	Comments
Z16 (16 bits per pixel)	1280x720	6	
	640x480	6, 15, 30	
	480x270	6, 15, 30, 60	
Y8 (8 bits per pixel)	1280x720	6	
	640x480	6, 15, 30	
	480x270	6, 15, 30, 60	
UYVY (16 bits per pixel)	1280x720	6	
	640x480	6, 15, 30	
	480x270	6, 15, 30, 60	
YUY2 (16 bits per pixel)	1280x720	6	
	640x480	6, 15, 30	
	424x240	6, 15, 30, 60	

Table 2: Table of supported camera modes

Finally, current design is not addressing camera discovery, authentication and data encryption. For list of known issues please refer to SDK release notes [27]

4 Operating the Device

This section assumes that the camera is configured with static IP `10.0.0.99`

4.1 Common Tasks

Restart the device:
```
ssh realsense@10.0.0.99
sudo reboot
```

Restart camera service (`rs-server`):
```
ssh realsense@10.0.0.99
sudo systemctl restart rs-server.service
```

4.2 Firmware and Software Update

These are the steps required to update firmware of a remote camera:

1. Download latest firmware from dev.intelrealsense.com/docs/firmware-releases
2. Extract firmware image file
3. Open secure shell session to the device:
   `ssh realsense@10.0.0.99`
4. Copy firmware binary from host to remote machine:
   `scp Signed_Image_UVC_5_12_3_0.bin realsense@10.0.0.99:/home/realsense/Signed_Image_UVC_5_12_3_0.bin`
5. Use rs-fw-update tool to update camera firmware:
   `rs-fw-update -f Signed_Image_UVC_5_11_6_250.bin`

Updating server software requires building new version of the SDK from source and installing it on the device. See compilation instructions

4.3 Tuning Linux Network Stack

Running the following commands can significantly improve network camera performance:

```
echo 8388608 > /proc/sys/net/core/wmem_default
echo 8388608 > /proc/sys/net/core/rmem_default
```

Raspberry Pi image comes pre-configured with these settings; however, we recommend also running this on the client side as well.

5 Power and Performance

All measurements were captured using the following setup:

- Intel® RealSense™ Depth Camera D435
- Raspberry Pi 4 Model B 4GB running `rs-server`
- Intel® Core™ i7-8550U @ 1.8GHz running Ubuntu 16.04
- 3000x2000 60Hz Monitor
- Gigabit ethernet (Raspberry Pi, host PC and network switch)

5.1 Latency

Intel® RealSense SDK 2.0 includes a simple tool for estimating camera latency – `rs-latency-tool`. It can be built from source using the OpenCV wrapper instructions.

`rs-latency-tool` *is designed to estimate camera latency without special equipment. It can offer only an upper-bound on the actual camera latency. In practice results also depend on host performance and monitor refresh rate*

Results vary from configuration to configuration, but overall show that the ethernet stack is adding on average between 1 to 1½ frames of extra latency:

	USB	Ethernet
RGB 640x480 30 FPS	112ms / 3.3 frames	160ms / 4.8 frames
RGB 1280x720 6 FPS	320ms / 1.9 frames	330ms / 2 frames
IR 640x480 30 FPS	93ms / 2.7 frames	130ms / 3.9 frames
IR 1280x720 6 FPS	166ms / 1 frame	330ms / 1.9 frames

Table 3: Table of latency measurements over Ethernet and USB

RGB sensor is rolling-shutter and by default actual FPS is controlled by auto-exposure algorithm. To reduce these effects, measurements were taken with manual exposure of 44ms and gain 93.

5.2 Power Consumption

The following results show data from example runs of the system at various configurations with both Depth and RGB streams enabled. Device was restarted between runs.

- Power – power consumption in Watts
- CPU% - percent of overall system CPU utilization, over all cores, counting both kernel and userspace load
- Temperature – temperature of the CPU in degrees Celsius
- FPS – average FPS across 5 seconds intervals
- Jitter – number of instances per second where frame did not arrive within 5ms range of expected time

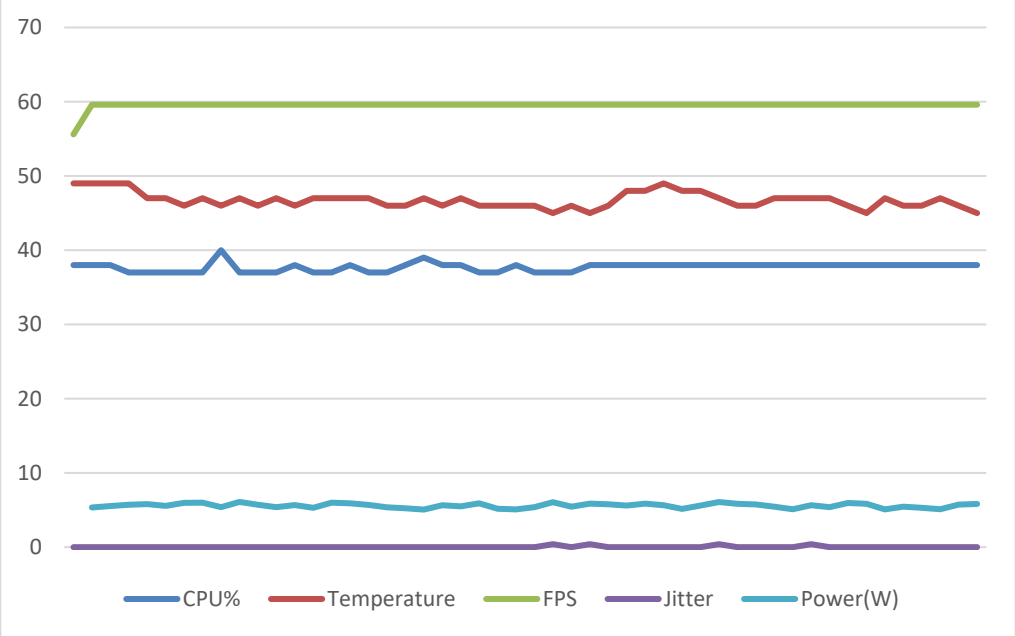

Figure 12. Low (480x270 for depth and 424x240 for RGB) resolution at 60 FPS

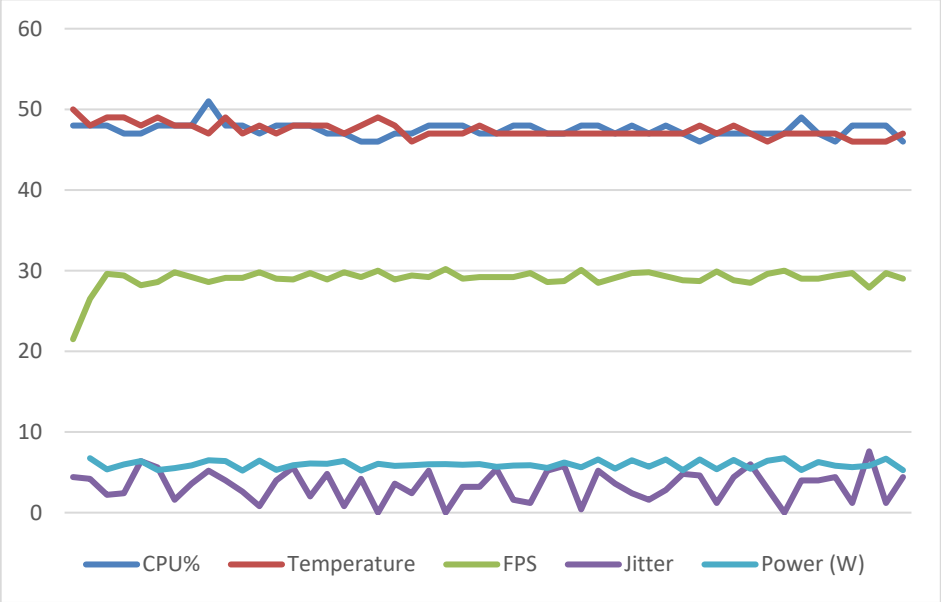

Figure 13. VGA resolution at 30 FPS

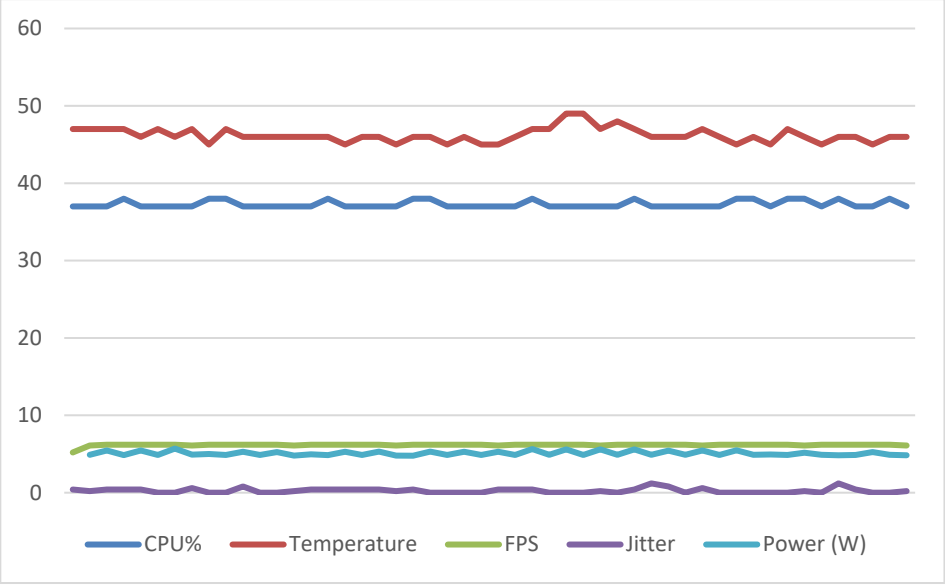

Figure 14. HD resolution at 6 FPS

6 Conclusion

6.1 Summary

This guide presents steps needed to assemble and operate network depth-camera based on Intel® RealSense™ technology. We've reviewed design considerations, software architecture and basic performance results.

6.2 Further Research

Streaming 3D content over ethernet is still active area of research. The following steps can further benefit this project:

- Leveraging better compression algorithms will allow more USB3 camera modes to be reliably supported over ethernet. Novel software and hardware 3D compression schemes are being continuously published and call for closer evaluation [22] [23] [24].
- Designing and manufacturing IP65 enclosure with an M12 connector could greatly improve applicability for industrial use-cases [18].
- Computational resources of the Raspberry Pi can be utilized to perform additional post-processing tasks. Depth decimation can further reduce network bandwidth, while spatial and temporal filtering can improve depth quality[19]. Furthermore, Intel® OpenVINO™ toolkit can be combined with Intel® RealSense™ SDK to run AI inference on the edge [20].
- RTP protocol allows for multi-client broadcasting. This opens new possibilities for sharing 3D data with wider audience over the Internet, for education and research purposes.
- Additional areas of research – pairing, camera discovery [25], authentication, data encryption and PTP time synchronization.

7 References

1. LIPSedge™ AE400 Industrial 3D Camera
2. FRAMOS Industrial Depth Camera D435e
3. Aeon Up-Board and Power-over-Ethernet Hat
4. Pi4 USB-C Gadget by Benjamin Hardill
5. live555 Media Server
6. LZ4 - Extremely fast compression algorithm
7. libjpeg-turbo – optimized JPEG compression implementation
8. Intel® RealSense™ Depth Camera over Ethernet
9. CompTIA A+ Certification Guide – on differences between USB2 and USB3
10. ROS Intel RealSense WIKI – using multiple cameras on two machines
11. Wikipedia - GigE Vision
12. JPEG and MPEG compression
13. FFmpeg License and Legal Considerations
14. KINOVA Gen3 Robots
15. Pudu Robotics and Intel RealSense Technology
16. Anybotics ANYmal C autonomous legged robot
17. HEBI Robotics
18. Mouser Electronics - M12 versus RJ45 Ethernet connection systems
19. Depth Post-Processing for Intel® RealSense™ D400 Depth Cameras
20. Intel® Distribution of OpenVINO™ toolkit with the Intel® RealSense™ Viewer
21. Fast Lossless Depth Image Compression by Andrew D. Wilson (video)
22. Intel RealSense D400 depth stream encoding to HEVC Main10 by Bartosz Meglicki
23. Real-time decoding and AR playback of the emerging MPEG video-based point cloud compression standard
24. Point Cloud Compression in MPEG - IEEE International Conference on Image Processing
25. PTP/IP Adapter Design and Connectivity Techniques for Legacy Imaging Appliances
26. Internet Protocols for Real-Time Multimedia Communication
27. Intel® RealSense™ SDK 2.0 Release Notes