IN3032UG: Ethernet Debugger User Guide

Website: https://intona.eu

Contents

1 Introduction

The Intona Ethernet Debugger is a device to capture packets between two Gigabit Ethernet devices. It provides two ethernet ports, and each port forwards all traffic to the other port, as well as to a PC connected via USB. The intended purpose is low level debugging of anything above the ethernet physical layer, mainly using Wireshark and similar protocol analyzers. It helps when developing your own protocols layered on top of ethernet, developing your own MAC, or just for observing what is going on on your network.

1.1 Features

This device can log complete ethernet packets as received by the PHY. There is no processing of captured packets – preamble, SFD, and FCS are all left intact. Packets with incorrect CRC sums are not discarded. Ethernet packets which violate the specification are captured as far it is possible. Some normally invisible low level details are explicitly logged, such as interpacket gaps and CRC errors. Jumbo frames (ethernet packets longer than 1500 bytes) are supported and fully captured up to 16KB size.

Capture output is directly streamed to the PC. There is no kernel device driver. The device is accessed through a libusb userspace driver. You do not necessarily need elevated privileges. Installing the device will not destabilize your system. In particular, the device is not exposed as network device. This has the advantage that your OS will not mess with it. Neither will it attempt to drop or filter packets received through it, nor will it attempt to send random packets to it (ARP etc.). The latter would show up in Wireshark, and confuse your development efforts.

Capture can be directly started from Wireshark (if installed correctly). The userspace driver also provides a command line interface, which can be used to access advanced feature. An IPC interface is provided for use cases like scripting.

There are many other features. See Other features section.

1.2 Requirements

The software works on Windows, Linux, and macOS. We provide an installer for Windows. Windows 10 64 bit is required, but Windows 7 may work as well. For macOS, a homebrew tap is provided[1]. For Linux, source code and build instructions are provided[2], which should work on any Linux distro.

USB 3.0 or later host and cable are recommended. USB 2.0 may work in low bandwidth scenarios. Using an USB hub, and/or connecting multiple USB devices to an USB hub/host may reduce the maximum bandwidth at which capture is possible without capture overflows.

1.3 Restrictions

Ethernet is intercepted by putting two PHYs between the two ports. There is no direct connection between the ethernet TX/RX wires of the ports. Each PHY negotiates the ethernet connection separately. No link can be established without USB power. If the devices connected to the debugger's ports use different ethernet standards (for example 100 MBit vs. 1 Gb), the port speeds need to be adjusted manually. You can for example force the debugger's PHYs to use a specific mode (see MDIO access section).

The PC needs to be fast to capture at full speed. Capturing in real time with maximum ethernet bandwidth directly to Wireshark or a slow hard disk may not be possible. (This is due to host performance problems

[1] https://github.com/intona/homebrew-ethernet-debugger
[2] https://github.com/intona/ethernet-debugger#build-instructions

outside of our control.) Packet buffer overflows should be expected when operating near maximum bandwidth. There is no hardware packet filter.

2 Hardware Setup

The ethernet debugger needs to be powered via USB to forward any packets. Without power, communication between the two ports is blocked. Make sure USB cable and host port are at least USB 3.0 compliant. If you use USB hubs or USB isolators, make sure they all support USB 3.0. USB 2.0 will work, but will provide degraded functionality, as USB 2.0 bandwidth is lower than that of Gigabit Ethernet.

Attach the ethernet cables to the ports. Make sure you are breathing regularly. As soon as both port LEDs indicate a connection, and the negotiated ethernet speed matches between both ports, communication is possible.

2.1 LED Meaning

2.1.1 Port LEDs

The LED in use indicates the ethernet speed mode. If packets are sent or received, the corresponding LEDs will blink.

Speed	LED L	LED R
1000 MBit	off	on
100 MBit	on	on
10 MBit	on	off
no link	off	off

Note that it's possible for one port to have a link, while the other port does not. They also can have mismatched speed. In both cases, capture will not work. 10 MBit mode is not supported.

2.1.2 Main LED

Normally, the main LED should be blue. It will blink if capturing is in progress. The following states exist:

LED state	Meaning
Blue	USB super speed connection is up
Green	USB high speed connection is up (slow, but functional)
Cyan	USB power only (will not work)
Blue blinking	capturing at USB super speed is in progress
Green blinking	capturing at USB high speed is in progress (will drop packets)
Blue/green blinking	the blink_led command is being used (only for a short moment)
Red	initial bootloader failed (probably flash problem)
Red blinking	bootloader failed (probably flash problem)
Off	low level bootloader/firmware crash, or no USB power
Red/yellow blinking	fatal higher level firmware error
Red/blue or Red/green blinking	firmware update failed; running factory firmware version

When you experience problems, it is useful to describe the LED state exactly in support requests, even if the observed variant is not listed above.

3 Software Installation

The software consists of a host tool, called ”nose”. It performs the following roles:

- userspace driver for the device
- Wireshark integration via Wireshark extcap
- command line tool for explicit access

3.1 Linux

No binaries or packages are provided. You can build it from the public git repository. Binary builds or packages for popular Linux distributions may be provided if there is enough demand.

- build the host tool from the git repository (see https://github.com/intona/ethernet-debugger#build-instructions)
- create a symlink for Wireshark extcap (see Wireshark section for details)

You may need to install an udev rule to get access to the USB device as normal user:

```
sudo cp udev.rules /etc/udev/rules.d/50-intona-ethernet-debugger.rules
sudo udevadm trigger
```

3.2 macOS

No binaries are provided. Hence the software is provided as source code, you can automatically download and build it using Homebrew.

```
brew install --HEAD intona/ethernet-debugger/nose
```

Homebrew is a 3rd party project for installing freely available software on macOS. See https://brew.sh/ for details and how to install Homebrew itself.

You need to setup Wireshark extcap manually. See below.

We decided to not provide binaries for macOS because it is quite impossible to deliver unified, stable executables that will work on various releases of both software API and CPU types. Think on the platform change to ARM. This is no issue when compiling from sources using Homebrew.

3.3 Windows

- make sure that Wireshark is installed before you proceed
- double-click the installer
- press next a lot of times

It does not matter whether the device is connected during installation. In fact, the installer does not try to access the device at all.

You can reinstall any time. Updating Wireshark might remove the Ethernet Debugger Wireshark integration. Reinstalling the Ethernet Debugger will restore it.

3.4 Verifying device access

A simple way to verify whether the software works is by simply running the host tool. It is a command line program. On Windows, double-clicking **nose.exe** will open a terminal window, while on Unix, you need to open a terminal window manually, and then run **nose** in it.

If the installation succeeded, and the device is connected, you should see the following:

```
Device 2:3 opened.
PHY 1: link=0 speed=0
PHY 2: link=0 speed=0
```

The example above has the device on USB bus 2, port 3.

If the device could not be found or accessed, the following is shown:

```
No devices found.
```

You can stop the host tool by closing the terminal or by entering the ”exit” command.

3.5 Wireshark extcap setup

Wireshark is the recommended way to use the Ethernet Debugger. It is 3rd party software and not developed by Intona. Download and install it from Wireshark's website: https://www.wireshark.org/download.html

Normally, the Windows installation procedure installs our host tool as a Wireshark ”extcap”. In short, ”extcap” allows external programs (such as our host tool) to provide a capturing source. See the Wireshark extcap section for details. If the host tool is not correctly installed as extcap source, you will not be able to start capture from the Wireshark GUI (but other methods of capturing will still work.)

The host tool supports extcap directly via special command line parameters. It must be located in the ”extcap” sub directory within the Wireshark installation directory or the user's Wireshark configuration directory. The paths depend on the operating system and the Wireshark installation location.

You can confirm whether it's installed correctly by opening the Wireshark about dialog, and switching to the ”Plugins” tab. There should be an entry named ”nose”.

Old Wireshark versions
The non-global/user-specific extcap paths below require at least Wireshark 3.1.1. Older releases support global paths only.

3.5.1 Linux, macOS

It is recommended to create a symlink to the host tool. The global path is something like /usr/lib/wireshark/extcap/ or /usr/lib/x86_64-linux-gnu/wireshark/extcap/. The exact path depends on the Linux distro. The user-specific path is usually **~/.config/wireshark/extcap/**.

The following should install it locally, assuming ”nose” is already installed:

```
mkdir -p ~/.config/wireshark/extcap/
ln -s `which nose` ~/.config/wireshark/extcap/nose
```

3.5.2 macOS (app bundle)

The following is useful if you want to install the extcap globally, or on older Wireshark versions, assuming "nose" is already installed via Homebrew:

```
ln -s `which nose` /Applications/Wireshark.app/Contents/MacOS/extcap/nose
```

The **/Applications/Wireshark/** part of the path can be different, depending on where exactly Wireshark is installed. Check the Wireshark about dialog ("Folders" tab) if you are unsure.

You need to start Wireshark at least once before you run the above command. Otherwise, macOS may show a security warning, and it won't work.

3.5.3 Windows

Since Windows does not support symlinks properly, it is recommended to create a .bat file in the Wireshark extcap sub-directory. The installer creates a file named **intona-ethernet-debugger.bat** with the following contents (assuming default paths and English locale):

```
"C:\Program Files\Intona\Ethernet Debugger\nose.exe" "%*"
```

This "redirects" all invocations to the actual installation location.

The user-specific path is **C:\Users\USERNAME\AppData\Roaming\Wireshark\extcap**. Replace **USERNAME** with the actual username. You may need to create the last component of the path. The installer does not try to use this.

Updating Wireshark tends to remove and recreate the Wireshark installation directory. This will also remove the **intona-ethernet-debugger.bat** file created by the Ethernet Debugger installer. You could run the installer again to fix this. Manually moving the .bat to the Wireshark user-specific configuration path mentioned above avoids this.

3.6 Firmware Update

Firmware updates may be required to get new features and to apply bug fixes. Normally they are not necessary. Applying such an update must be done explicitly. The update process rewrites the device's flash memory, and should not be interrupted. Make sure the device is connected via USB 3.0, as the update will take a long time with USB 2.x.

3.6.1 Unix-like

Plug in the device, and ensure it's using USB 3.0. Then run the following command on the terminal:

```
nose --firmware-update Downloads/firmware.dat
```

Where **Downloads/firmware.dat** is the path to the firmware binary you downloaded. If the firmware file is accepted, and the device is accessible, the update will start automatically. The tool will exit when the update is finished or aborted. On success, the device restarts on its own, and runs the new firmware immediately.

You can confirm successful update by comparing the device version number (**bcdDevice**) with the version indicated by the update. If the device comes up and blinks red, retry the update, or if the tool fails again,

contact support.

3.6.2 Windows

The same instructions as with Unix apply, but you need to provide the full path to the host tool, which is inconvenient. ”Building” the command line can be made less tiresome by using drag & drop, instead of entering filenames manually:

- start the terminal with cmd.exe (click the Windows start icon, type cmd.exe, hit the enter key)
- open Explorer, and look for the Ethernet Debugger installation directory
- drag nose.exe on the terminal window (inserts the full path)
- hit space, enter ”--firmware-update”, hit space again
- locate the firmware update file in Explorer, drag it on the terminal (inserts the full path)
- hit enter

4 Capturing

The main purpose of the ethernet debugger is to capture packets. The following methods are available.

4.1 Wireshark extcap

If you open Wireshark, it should display any plugged in Intona Ethernet Debuggers as **`Ethernet Debugger USB (1:2)`** in the list of capture interfaces. The **`1:2`** in the brackets is the device name (as used by the host tool), which consist of the USB bus and port numbers. (Some versions of Wireshark display this twice.) Double click this entry, and Wireshark should start capturing. The Ethernet Debugger's main LED will start blinking.

There is no hotplug mechanism for Wireshark extcaps. If you connect or disconnect devices while Wireshark is running, you may need to press F5 or restart Wireshark to update the device list of Ethernet Debugger capture devices.

Note that if the bandwidth utilization is high, the internal FIFO may overrun, leading to lost packets. The host tool adds a packet comment to the first packet after a run of dropped packets.

It may also happen that Wireshark freezes if the amount of data is too large, because the GUI requires a large amount of resources to deal with packet input. (Capturing to disk via the "nose" tool may help reducing packet drop. You can open the capture file with Wireshark afterwards.)

Various error conditions may deadlock Wireshark and the host tool on capture start.

4.1.1 Capturing options

Wireshark lets you set some Ethernet Debugger specific options before starting capture. Click on the gear-like symbol left of the Ethernet Debugger interface in Wireshark's Capture interface list.

4.1.2 Wireshark extcap toolbar

The host tool provides a toolbar in Wireshark. This is implemented through the extcap mechanism. It is slightly clunky due to Wireshark restrictions (all GUI code is provided by Wireshark, and not everything can be realized). The toolbar can be shown by enabling it in the Wireshark "View" menu, "Interface Toolbars" sub-menu.

4.2 Directly starting Wireshark from host tool

You may use the **`--wireshark`** option of the host tool to start Wireshark and capturing in one go. It attempts to find the installation path of Wireshark, sets up a named FIFO, and starts a new Wireshark process.

Example

```
nose --wireshark
```

4.3 Statistics

The host tool **`--capture-stats`** option can be used to enable regular statistic updates on the terminal. The "set capture-stats true" command can be used to do this at runtime. (You can enter this command on the Wireshark extcap toolbar, for example.)

4.4 Capturing to a file

The host tool **`--fifo`** option can be used to capture either to a real file on disk, or a named FIFO. The **`capture_start`** command is similar, and can be used to start capturing via the host tool command line or IPC interface. The format of the output is PcapNG (see https://pcapng.github.io/pcapng/). You may use the third party open source libpcap library to parse such files. If you use an actual FIFO, you can stream in real time.

Note that if you capture to disk, overruns can happen due to waiting on disk I/O. The host tool tries to avoid this by using decoupled memory buffers, but these may be slowly filled up, until a software overrun happens.

Example

```
# Capturing to a file until Ctrl+C is hit, and log capture statistics to stdout.
nose --capture-stats --fifo target_file.pcapng

# Manually starting Wireshark.
# On terminal 1:
mkfifo /tmp/fifo
nose --fifo /tmp/fifo
# On terminal 2:
wireshark -k -i /tmp/fifo
```

4.5 Selecting the device

If you have multiple Ethernet Debuggers, the **`--device`** option can be used to pick a specific device. Passing the special value **`help`** to this option lists all devices that were found.

Multi-Capture
Selecting multiple devices at once is not possible. However, if extcap is correctly installed, you can select multiple capture devices in Wireshark. This will provide a merged view of data coming from multiple devices and host tool instances.

4.6 Configuring the buffer size

The **`--capture-soft-buffer`** and **`--capture-usb-buffer`** can be used to fine-tune the sizes of the fixed size buffers allocated on the host. Raising them may reduce buffer overruns on the host PC.

5 Other Features

5.1 PoE passthrough

Both ethernet ports are capable of handling Power over Ethernet (PoE) as specified in IEEE 802.3af, 802.3at and 802.3bt. Power is passed through in both directions without interception.

5.2 Supported command line options

You can list supported options as follows:

```
nose --help
```

Current list of options:

Name	Description
--verbosity 0|1|2	Set verbosity log level: 0 silent, 1 normal/default, 2 verbose messages
--version	Print host software version and exit
--selftest	Run internal self-test. (Requires a loop between the two ports.)
--selftest-serial	Internal.
--wireshark	Start wireshark and dump packets to it. Terminate once done.
--device name	Open this device. (Pass ”help” to get a list.)
--firmware-update file	Perform a firmware update using this file.
--fifo file	Start capture and write to the given file or fifo. (Overwrites the target if it’s a file.)
--ipc-connect path	Connect IPC to this socket/named pipe. Terminate on disconnect.
--ipc-server name	Host IPC on this socket/named pipe.
--capture-soft-buffer num	Capture soft buffer (in bytes, accepts kib/mib/gib suffix)
--capture-usb-buffer num	Capture libusb buffer (in bytes, accepts kib/mib/gib suffix)
--capture-stats	Show capture statistics every 1 seconds.
--extcap-*	Various options for use by the Wireshark extcap mechanism.
--capture	Used by Wireshark; ignored by host tool.

5.3 Interactive command line

The host tool has an interactive command line interface. When starting the host tool without commands, it will wait for commands from the terminal. You can use the ”help” command to list available commands and their parameters. Many advanced features (such as listed below) are accessible only through this interface.

By default, the host tool will read from stdin. It will terminate if stdin is closed or returns EOF. You can use the rlwrap[3] 3rd party tool to get comfortable line editing and history:

```
rlwrap nose <arguments for nose>
```

[3] https://github.com/hanslub42/rlwrap

5.4 IPC interface

The command line interface is available via IPC (unix domain sockets on UNIX, named pipes on Windows). This may be helpful for scripting. For example, you could inject or drop packets under specific situations, or start/stop capturing at specific times

The **`--ipc-server`** command can be used to bring up the server at a specific path while the host command runs. **`--ipc-connect`** can be used to make the host tool initiate the IPC connection, which may be more convenient with certain frameworks.

You can send commands in text or JSON form (the **`help`** command lists available commands and shows the basic syntax). The protocol uses 1 JSON object per line (in both directions of communication). A line is terminated with **`\n`** (ASCII line feed, byte 10). It is fairly self-describing:

IPC example

```
{"command":"mdio_read", "phy":1,"address":1,"id":1}   // client to host tool
{"id":1,"result":31049,"success":true}                // host tool to client
```

The **`id`** field is an arbitrary integer chosen by the client, and can be used to associate requests with replies. The C-style comments are for illustration, and not part of the protocol.

Log messages (such as output when PHY links change their state) are wrapped into special JSON messages:

Log message example

```
{"type":"log","msg":"PHY 1: link=up speed=1000MBit\n"}
```

5.5 PTP timestamps

The device provides high resolution timestamps for packets. This can help to debug PTP related issues, or any other timing issues.

Each PHY has its own FIFO, which may affect accuracy. In addition, device-internal CDC may affect the accuracy. Internally, the timestamps are relative to device start. The host tool capture output adds a start offset to the timestamps to make them roughly line up with wall clock times. However, this offset is not precise, and there is no time correction. The absolute time of a packet event might be slightly different from real time. The longer the capture is running, the higher the deviation will be.

5.6 Packet injection

It is possible to manually inject new packets into the ethernet connection. As no actual network interface is provided, OS mechanisms (such as sockets) cannot be used. Using the host tool is required.

The **`inject`** command provides this functionality. It is possible to send arbitrary ethernet packets, including packets that are not spec-compliant. This is not a high speed send path – full ethernet bandwidth cannot be reached. (Although you can instruct the command to repeat packets, in which case it will produce a high bandwidth stream of the same packet being repeated.)

CRC errors can be injected. Degenerate packets (too short, IPG too low) can be created. Low level physical layer coding errors cannot.

Using the command may drop packets coming from the opposite source port. The injection logic will wait until transmission is turned off, then it will inject the packet, and fully drop any other packets from the

opposite port.

Example

```
# Inject a packet that starts with a AB:CD:12:34:56 dest. MAC,
# the rest of the packet filled with 0s:
#  1         on port A (into the stream of traffic flowing from B to A)
#  (hex)     packet payload
inject 1 ABCD123456
```

See **help** output for advanced parameters that control repeating, raw output (your own preamble), and so on.

5.7 Packet disruption

The **disrupt** command can be used to drop or to destroy some or all packets in a certain direction. This is a form of error injection suited for testing reliability of low level protocols. It can either flip a single bit in a packet at a user-chosen byte offset, or discard entire packets.

(Restriction: cannot drop specific packet according to filter-like matching criteria etc.)

Example

```
# Disrupt a number of packets for a short time:
#  2         on port B (traffic flowing from A to B)
#  true      just drop the packets (instead of creating CRC errors)
#  20        drop 20 packets in total
#  10        drop only every 10th packet (this makes it active for ~200 packets)
disrupt 2 true 20 10
```

5.8 MDIO access

The **mdio_read** and **mdio_write** commands provide direct access to the PHYs.

5.8.1 Changing PHY ethernet speed

Ethernet speed is controlled by the PHYs. By default, they are set to auto negotiation. You can use the **speed** command to force specific speed modes on both PHYs:

Speed	Command
1000 MBit	speed 1000
100 MBit (full duplex)	speed 100
10 MBit (full duplex)	speed 10
auto	speed auto

The **auto** mode lets each PHY negotiate the speed, which will lead to problems if the device's port A and B PHYs do not negotiate the same speed.

6 Known Problems

- If no ethernet is connected, or there are no packets on the wire at all, the host tool may not terminate properly, even if Wireshark is terminated. This happens because the host tool will never write to the pipe to Wireshark. The host tool will continue to run in the background, and will block access to the device. In this case, the LED will continue blinking, and you may have to kill the host tool manually.
- If extcap is used, and start of capture fails (such as inaccessible device), Wireshark does not terminate gracefully. This seems to be a Wireshark issue.
- The USB host part of some AMD chipsets seem not to play well with this device at least on Linux.
- 10 MBit mode is not working.

Document version: 60 / Okt 29, 2020 13:22