EPL-Viz

Visualizing Ethernet POWERLINK Traffic

Ahmad Fatoum, Denis Megerle, Joshua Schauer,
Dennis Schunder, Daniel Mensinger, Andreas Bihlmaier
Karlsruhe Institute of Technology (KIT), Germany
Email: bihlmaier@robodev.eu

***Abstract*—EPL-Viz is a visualization solution for Ethernet POWERLINK networks. It enables users to monitor the network state and narrow down the cause of issues. Thus, helping to detect and fix bugs in MNs as well as CNs. EPL-Viz provides various tools to analyze live and recorded traffic in a flexible GUI. Beyond the integrated consistency checks, users are also able to perform custom data analysis by means of an integrated Python interpreter.**

I. Introduction

Ethernet POWERLINK (EPL) has established itself as one of the major field bus standards with hard real-time guarantees on top of standard Ethernet. Yet, debugging a stateful real-time bus can be a daunting task: To fully reconstruct the state, out-of-band configuration files as well as prior network packets need to be taken into account. A task for which, to the best of our knowledge, no automated solutions exist.

As part of the curriculum for the B.Sc. in Computer Science at the Karlsruhe Institute of Technology, teams, of five students each, are assigned the task of collaborating on a software project over the course of a semester. The tasks are handed out in the form of a customer requirement specification. During the project, progress is discussed with a stakeholder, who is not only the supervisor, but also acts as the customer that has a particular interest in the resulting product. Our stakeholder was Andreas Bihlmaier from the robodev GmbH and the Intelligent Process Control and Robotics Group at the Institute for Anthropomatics and Robotics, both of which use Ethernet POWERLINK and CANopen extensively.

Debugging efforts so far involved the developer consulting the device descriptions and trying to match them in the packet list captured by the network analyzer Wireshark, a very time-consuming endeavor as thousands of frames can accumulate over the matter of seconds. This is where EPL-Viz comes into play: Built on top of Wireshark's EPL dissector, the software passively monitors the network and uses the extracted information to create an accurate model of the state of each node on the bus, which the user can then explore. It includes different visualization options and a number of tools aimed at increasing the comprehensibility of changes occurring through each network packet as well as finding possible network errors. This allows the user to fully focus on error discovery, leaving the tracking, visualization and interpretation of the data to EPL-Viz.

II. Project structure

A. Wireshark Dissector

Instead of creating our own implementation of the EPL protocol, the existing dissector in Wireshark was leveraged as backend. Yet, the dissector had to be significantly extended to satisfy EPL-Viz requirements. In particular, three features were missing:

SDO read responses aren't matched with prior read requests. The read request on its own only contains the data, but no indication of its origin index and subindex.

OD entry descriptions can't be used to enrich displayed information, because Wireshark doesn't parse XDDs.

PDOs are not partitioned because Wireshark doesn't track the SDOs that setup the object mapping.

We extended Wireshark to address these points, CN state is now saved in Wireshark-specific "conversations", which record previous read and write requests. Conversations can be enriched with device descriptions read from either CANopen EDS or POWERLINK XDD files: Indices and Subindices

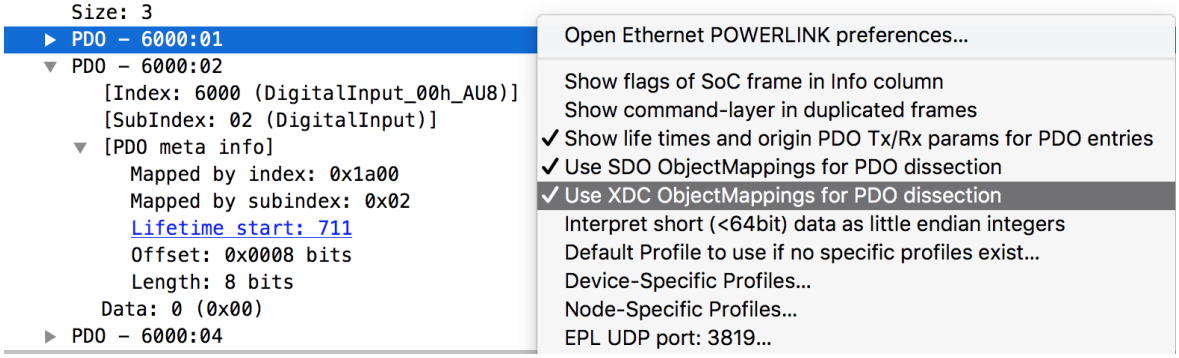

Fig. 1: The patchset for EPL-Viz added 7 new Wireshark configuration options.

are tagged with the textual description in the device profile and the data is displayed according to the type listed therein. Finally, object mappings configured over SDO and XDC are now taken into account: Poll request and response payloads are no longer shown as a single binary string. With SDO context available, the payload is partitioned into its constituent units and is annotated according to the appropriate XDD. As a side effect of the refactoring effort, a few bugs in the stock EPL dissector were fixed.

Our patchset has been successfully merged upstream. Wireshark 2.4.0, scheduled for end of July 2017, will be the first stable version of Wireshark to ship with these changes.

The new conversation mechanism also enables future enhancements to the way Wireshark may detect possible EPL errors, which in turn benefits EPL-Viz.

Apart from the obvious benefit of using a well-tested protocol implementation, using Wireshark as backend also meant that we can rely on its live capture and file format support capabilities. This makes EPL-Viz portable to theoretically any of the plethora of systems supported by Wireshark itself; given that appropriate C++ support is available.

B. EPL_DataCollect

EPL_DataCollect is the C++ backbone of the EPL-Viz project. Its main task is to accumulate the data provided by the Wireshark dissector and organize it in line with the EPL protocol structure.

The main structure in EPL_DataCollect is the cycle, which spans multiple packets and is, after

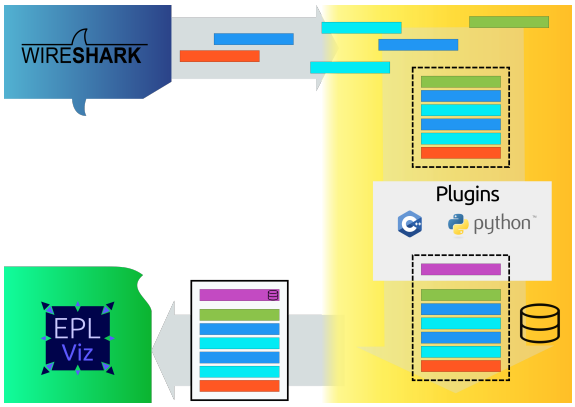

Fig. 2: An overview of EPL_DataCollects role.

aggregation, passed to all loaded plugins.

Two types of plugins are supported: C++ plugins, which are linked into the program at compile-time and dynamically loadable Python plugins. The latter can be loaded just before starting a new live capture or analyzing an offline capture file. Plugins have full access to the network state by means of the cycles data. Furthermore, plugins have access to a CycleStorage, which allows them to attach data to cycles. Thereby, they can retain historical information and provide additional data, which GUI views can then display to the user.

After a cycle has been processed by all plugins, the resulting data is passed to our default graphical frontend, EPL-Viz.

EPL_DataCollect further provides an event system, allowing communication between itself, the plugins and the frontend. Using these, users can write plugins that interact with the view in a way beyond just providing data. An example usage of this is

using a Python plugin to highlight specific devices once certain conditions are met, which can be highly helpful to detect errors with many CNs.

In order to allow users to see a networks state over time, EPL_DataCollect takes snapshots of the network state at fixed intervals. As this only provides a look back at certain points in time, any non-preserved time points are reconstructed by taking the closest snapshot before it and applying any packets that were sent in between that time frame. This in turn allows for a continuous time line of the network state, while significantly reducing the amount of memory required to retain all states.

To ensure that consumers do not lose events and cycles, e.g. because the frontend is not able to process and visualize data as fast as the backend can provide it, the consumer's important data structures are safeguarded by buffers that have to be queried by consumers once they are ready to process the data. A query will then always provide any data that has arrived since the last query. This ensures that every consumer gets a chance to view any and all data provided by the network.

Frontends can make use of EPL_DataCollect by starting a capture on an interface, similar to Wireshark, or by loading a previously saved capture for playback.

In case the current capture of the network (be it a live or file capture) did not yet send a Start of Cycle (SoC) packet and thus did not yet start the first cycle, EPL_DataCollect can optionally be configured to use so called Pre SoC cycles. These allow the same cycle bound interactivity in the GUI by creating cycles out of Start of Async packets. In turn, this can then be used to analyze network startup errors. As soon as a Start of Cycle packet arrives, this feature is automatically disabled to ensure the program remains functional.

C. Plugin-API

As previously mentioned, plugins can read all cycle data, network state and information accumulated by EPL_DataCollect. Python plugins can be created right inside the GUI with the integrated KDE based editor. Alternatively, the user can directly load an external script. For performance-critical tasks, plugins can also be written in C++, which additionally have full access to the backend on top of the cycle data structure that the Python plugins receive.

Fig. 3: Users can extend EPL_DataCollect functionality with both C++ and Python plugins.

Both kinds of plugins can then interact with any given view through the event system provided by EPL_DataCollect. These events are logged to an event log and can also be displayed on the time line, in order to alert the user.

Furthermore, plugins can act as filters for Node ODs. Once the filter plugins are loaded, the user can freely switch between the set of filters during playback and recording to narrow down the search for essential information.

```
def run(self):
  cy = self.getCycle()
  i = int(cy.getODEntry_Sub(1,
                            0x6200,
                            1))
  if i >= 20:
    self.addEvent(
      Events.EV_TEXT.value,
      "Threshold reached!",
      ""
    )
```

Fig. 4: A plugin displaying an event whenever a specific OD entry reaches a threshold.

Sample plugins and extensive documentation are distributed together with EPL-Viz.

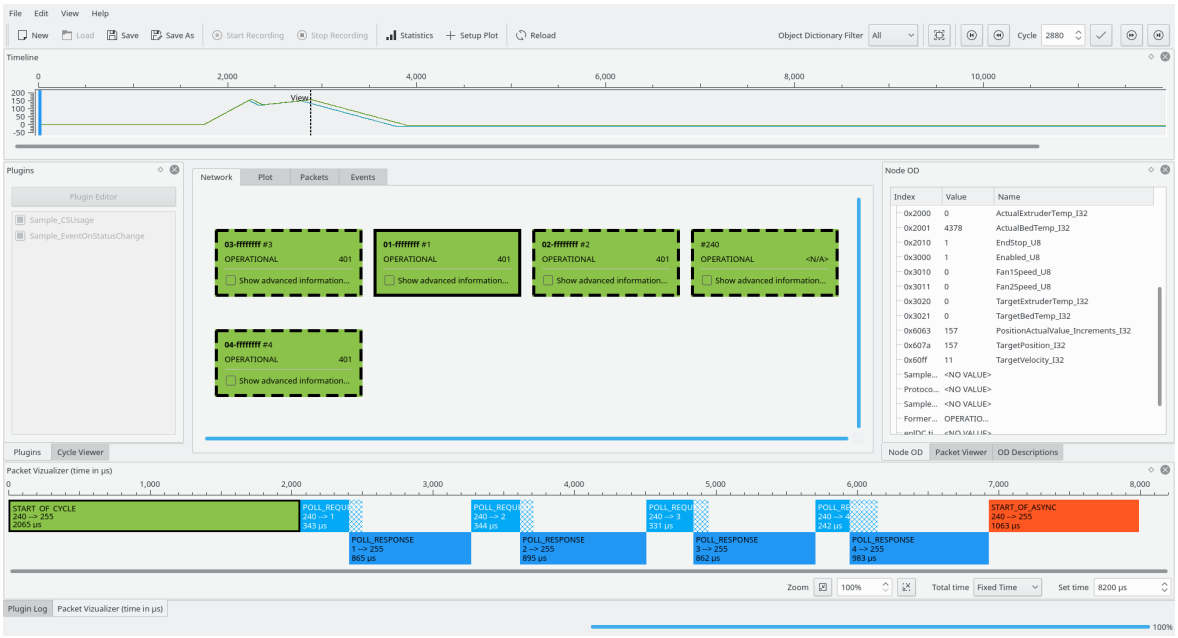

Fig. 5: A screenshot of EPL-Viz.

D. EPL-Viz

EPL-Viz is the default graphical frontend to EPL_DataCollect. It is built with Qt 5.5 to ensure a modern look and feel.

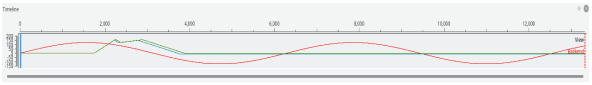

Fig. 6: The "timeline" with several plots.

The timeline (see figure 6) is the main part of the program, presenting a historical overview of the network. It displays the events that have occurred, the cycle selected by the user and can also be used to visually plot the values of data fields. The most important feature of the timeline, however, is synchronized interaction with all other parts of EPL-Viz. Clicking on any particular point of time sets the current cycle to the one selected, causing cycle-dependent widgets to display the information for that cycle. This makes navigation between different points in time simple. The same also holds during capturing live network traffic or while parsing a file. In this case, the view will stop updating while the backend continues working. These timepoints are then represented by a "View" and "Backend" marker. The main usage of the timeline is to allow the user to easily see the network state change over time, as well as making the discovery of time points at which the network had experienced errors easier.

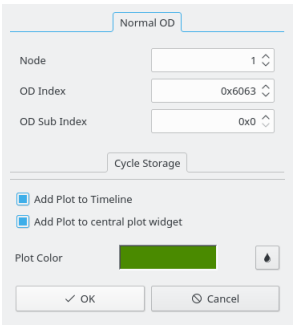

Fig. 7: The "Setup Plot" Dialog.

Users can create plots for the OD values of specified nodes, which includes the custom CycleStorage data written by plugins. Plots can be easily set up using the "Setup Plot" dialog (see figure 7). Plots are displayed in the timeline, as well as in the dedicated "Plot" Widget. An important note regarding plots: A plot does not receive information about the time before it was created. In case the user wants to visualize data from before setup of the plot, EPL-Viz provides a "Reload" functionality that reparses the current file and thereby showing all data. Plots can be especially useful in cases where particular values, such as the speed of a faulty motor, are suspected

to cause problems and must be analyzed in detail. With the addition of custom values, this allows users to keep track of essential data. The latter is further enhanced, given the timeline, which provides an easy way to jump to certain parts of the plot.

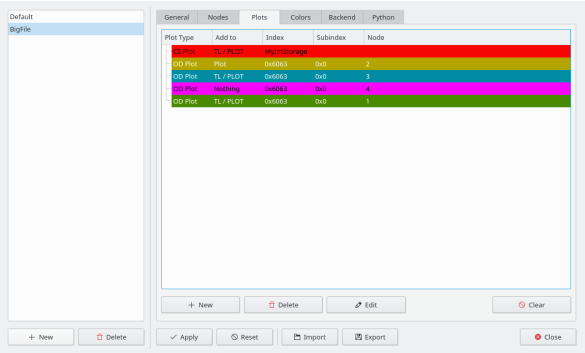

Fig. 8: The settings for plots.

The plots settings window provides an overview of all plots and their parameters. The settings provide several other configuration options for EPL-Viz itself. These include setting colors for identifying certain node states and packet types, as well as configuration of the EPL_DataCollect backend. The user can also add nodes manually, allowing custom device profiles to be loaded and used for the nodes. The EPL-Viz settings enable the user to choose and create multiple profiles, which store all previously mentioned program settings and enable to store and recall different setups of plots and settings for handling specific problems. The profiles can then be shared among users or be used to restore the configuration on a different workstation. This can be helpful when a team is observing a certain capture file or analyzing recurring problems of a specific network. Team members can thus create and share setups that target relevant data and make the analysis of the network easier.

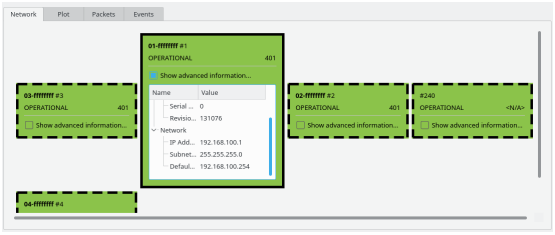

Fig. 9: The network node graph.

Another important aspect of EPL-Viz is the network node graph. It gives a quick overview of the devices that were either recognized by analyzing the traffic or added manually by the user. It summarizes information about CNs and their state. In situations where a faulty device has to be identified, the color coded visualization of each device's status makes it easy to spot faults. The displayed identity information of the nodes helps to locate the exact device in the network. Furthermore, widgets that display node-specific data make extensive use of this graph, as a user can select the active node by simply clicking on it. The node OD widget is one such plugin, showing the Object Dictionary entries of the currently selected node. The OD description widget further makes use of the device profile of the currently selected node, to show the user what Object Dictionary descriptions exist. This however requires that the required device profile file has been specified.

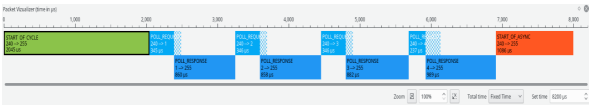

Fig. 10: The Packet Visualizer.

The packets of the currently active cycle are represented by another main widget, the Packet Visualizer (see figure 10). It represents an accurate overview over the temporal structure of a cycle. The user can use it to see the interaction between the nodes and spot potential problems in the timings. Packet types are color coded. Poll requests and poll responses are placed in a way to render the communication within the cycle more understandable. Furthermore, individual packets can be selected for closer inspection. Clicking on one of the packets will show its Wireshark-provided dissection. Changing the currently selected packet can also be achieved by selecting a packet via the full list in the "Packets" widget or choosing one from the current cycle in the Cycle Viewer widget. The latter grants a table view of the same information held within the Packet Visualizer to give a quick, minimal overview over the essential cycle information. Here again one major design principle of EPL-Viz is shown: Provide multiple interlinked and synchronized visualizations of the underlying communication data to aid human understanding.

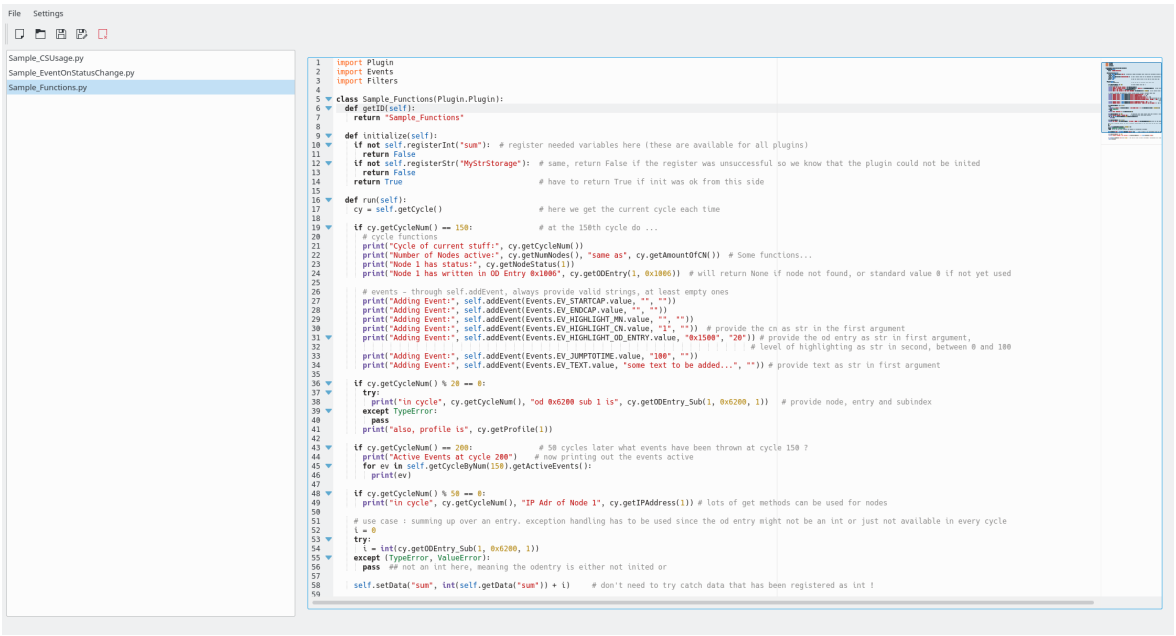

Fig. 11: The KTextEditor based Plugin Editor.

Additionally, EPL-Viz has a built-in Python plugin editor (see figure 11) that allow users to create Python plugins on the fly. This gives the user the ability to create custom tools to better handle specific situations and cater to specific needs for detecting and resolving issues in a network. The editor optionally uses the KDE's feature rich text editor KTextEditor, which is enabled in official EPL-Viz releases.

III. Summary

EPL-Viz is a powerful and extensible approach to analyzing, understanding and debugging POWERLINK setups. Through a visually appealing graphical user interface, a Python scripting interface, numerous plotting features and more, EPL-Viz boosts efficiency in developing and maintaining POWERLINK networks.

The client-server model of Wireshark, whereby a separate privileged binary does the network sniffing and the GUI runs with normal privileges, not only hardens the security of EPL-Viz, but also makes EPL-Viz extendable for remote live capture.

Apart from the Wireshark backend which is licensed under the GPL 2.0, the EPL-Viz project is licensed under the BSD 3-Clause license. The software builds out of the box on Windows, macOS and Linux and should be straightforward to port to other systems supported by Wireshark.

The project's source code and binary releases for Windows and Linux are available at [1], where we hope to get valuable feedback from the Ethernet POWERLINK community.

A screencast showcasing usage of the GUI is available at [2].

References

[1] "EPL-Viz Repositories." https://github.com/epl-viz.
[2] "EPL-Viz YouTube Channel." https://www.youtube.com/channel/UC4f3HO4aGFd5DGUphm2LKPg.