NETWORK ANALYSER FOR THE EPICS CHANNEL ACCESS PROTOCOL

A. Žagar, K. Žagar, Cosylab, Ljubljana, Slovenia
K. Furukawa, KEK, Ibaraki, Japan
R. Rechenmacher, Fermilab, Batavia, Illinois, USA

Abstract

In this paper, we present a tool which allows capturing Channel Access (CA) traffic directly off the network and interpreting the contents with a graphical or textual user interface. The tool is the widely used Wireshark (former Ethereal) network capture and analysis application, for which we have implemented a plugin that parses (dissects) contents of CA network packets. The tool is freely and openly available for several operating systems, and we have built and tested the CA plugin for Windows, Linux and Darwin (Mac OS X). We first describe the Wireshark framework, followed by the steps needed to implement a dissector plugin. Then, we explain how to install and use the Wireshark application and the CA dissector. Afterwards, we present the features and limitations of our CA dissector implementation. Finally, we present some examples where we have found the tool to be useful.

INTRODUCTION

Wireshark [1] is a packet sniffer and network protocol analyzer running on most computer platforms, including Windows, OS X, Linux, and UNIX. It is entirely written in C and is freely available as open source, released under the GNU General Public License version 2. In June 2006 it was renamed from **Ethereal** due to trademark issues.

For capturing packets, Wireshark uses **pcap** library that allows sniffing traffic from many different network types, including Ethernet, IEEE 802.11, PPP, bluetooth and USB. Captured traffic can be stored and opened from trace files. Besides natively supported tcpdump (libpcap) format also other trace file formats can be used, such as snoop, Microsoft Network Monitor, etc.

For analysis of captured traffic, Wireshark ships with support for dissection of hundreds of different protocols. To extend its initial feature set, Wireshark also provides support for plugins. In next section we will explain how we have implemented a dissector plugin capable of dissecting Channel Access network traffic.

Graphical user interface is based on the cross-platform GTK+ widget toolkit and along with the TShark command line frontend provides a tool for network troubleshooting, analysis, software/communication protocol development and education.

EPICS Channel Access (CA) [2] is a network protocol used by EPICS. It provides mechanisms for automatic discovery of input-output controllers (IOCs) hosting EPICS records (typically via UDP/IP broadcasts), polling or publish-subscribe based read/write access to EPICS record values (process variables – PVs), etc.

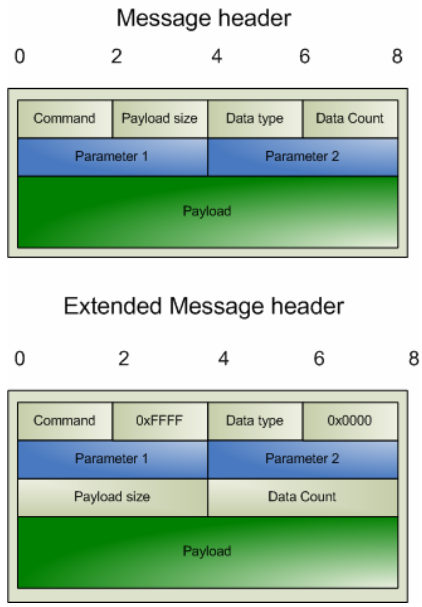

Figure 1: Header of a Channel Access message

All channel access messages are composed of header that is always present, followed by an optional payload. With version CA_V49 of Channel Access protocol, the maximum message size was extended from 16384 to 4294967255 bytes. Figure 1 shows the message header compared to the extended message header. Due to backward compatibility, the regular message form is still valid and should be used whenever the payload size does not exceed 16368 bytes. Extended form is recognized by the 2^{nd},3^{rd}, 6^{th} and 8^{th} bytes of the message header.

Detailed protocol specification can be found at [2].

Original implementation of the CA dissector plugin for Wireshark (at that time still Ethereal) was implemented by Ron Rechenmacher [3]. It was not a complete solution, because only some of the most common CA message types were partially supported and build was no longer compatible with the latest versions of Wireshark.

OBJECTIVES

First objective was to make existing solution work with the latest version of Wireshark and see what exactly was missing. Then, message parsing was implemented according to the specifications [2]. So far, only regular message form is supported. Finally, CA session extraction needed to be fixed. Original solution was extracting sessions only based on source and destination port. In CA,

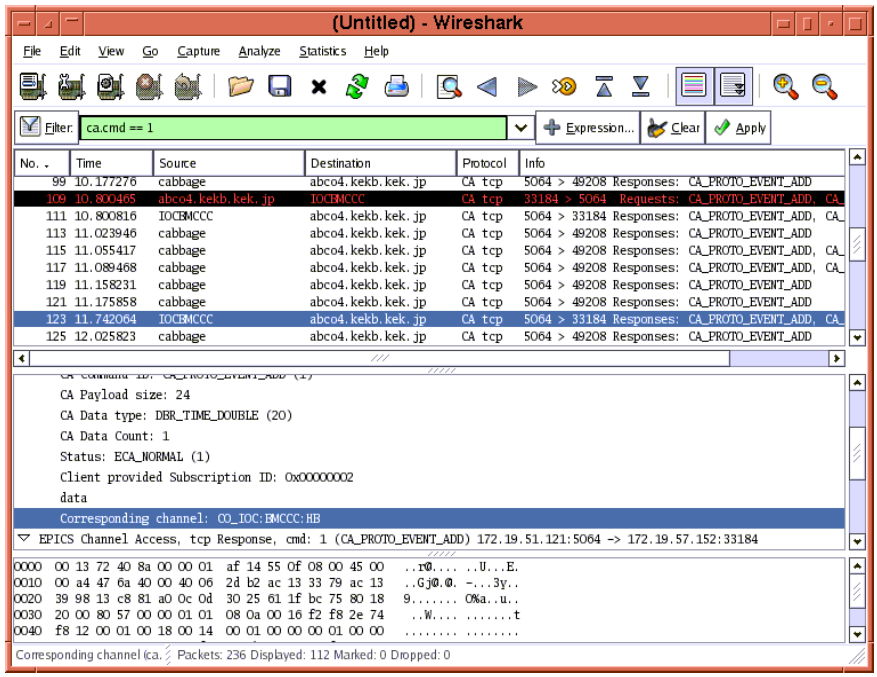

Figure 2: Wireshark with CA Sniffer

however, ports do not uniquely identify a session. Fields like CID (Client ID), SID (Server ID) and Subscription ID needed to be taken into account, as well.

IMPLEMENTATION

Wireshark is entirely implemented in C. Although it is not object-oriented, it provides enough modularity to easily add support for new dissectors or additional trace file formats. Such support can either be statically linked into Wireshark, or delivered as a separate library file that gets automatically loaded at runtime just by having it located in Wireshark's plugins directory.

Implementing a plugin requires some changes of the Wireshark build in the original source tree and adding the directory containing the plugin's source code in the `plugins` directory. Both, patch for the Wireshark sources and the archive containing the CA plugin implementation, can be found at [4].

Aside of several documentation files and build-related files that normally require little adjustment, the plugin consists of the following relevant source files:

- `moduleinfo.h` defines the module/plugin version and name.
- `plugin.c` implements a couple of methods that register the methods that do the actual dissector registration and initialization. These other methods are implemented in packet-ca.c.
- `packet-ca.c` is the actual dissector implementation. Besides registration and initialization methods mentioned above, this source file also provides the method that does the actual dissection (`dissect_ca`). This method is performed by Wireshark in two iterations. In first iteration it only collects the basic data about all the packets displayed in the list of packets (see figure 2) and extracts the CA sessions (called conversations in Wireshark). The second iteration parses the packet details displayed only for the selected packets (the tree in the figure 2).

Limitations

Implementation of CA plugin for Wireshark makes it easy to analyse the CA traffic on the network. It dissects all CA packet header information, i.e. requests and responses with all their parameters, in compliance with the CA protocol specification from [2]. Note, however, that this specification document might not be fully compliant and updated with the current versions of the actual protocol implementations.

Dissector also tracks PV/Channel names along the virtual circuits based on the packets' client, server or subscription IDs which is indispensable for human-readable analysis. Yet, it lacks support for payload and extended message header dissection, however, absence of these features has not been found too limiting so far.

INSTALLATION

Binary installation package is currently available for Windows, Linux and MacOSX (x86/ppc). To install, proceed as follows:

- Install normal Wireshark 0.99.8 or 0.99.7
- Install CA plugin binary into Wireshark's plugins directory [4]

To build the plugin from source:

- Install GTK+ and pcap libraries
- Get Wireshark (0.99.8 or 0.99.7) [1]
- Extract CA plugin source into Wireshark directory
- Apply patch
- Commence with normal building procedure (see [4] for details)

USAGE

Simple usage instructions:

- Start Wireshark
- Adjust capturing properties [Capture → Options (Ctrl+K)]: select the correct interface and optionally set the capturing filter (aside the 'Capture Filter:' button you may enter "port 5064 or port 5065")
- Start capturing [Capture → Start]
- Generate EPICS CA traffic on the network
- Stop capturing [Capture → Stop (Ctrl+E)]
- Apply display/analysis filter (aside the 'Filter:' button).

Some useful analysis filter examples:

- `ca.cmd=1`
  displays only CA_PROTO_SEARCH messages
- `ca.chanName=="fred"`
  `ca.channel=="fred"`
  messages related to a PV named "fred"
- `ca.channel matches "^VAC:IP.*:Pressure"`
  `ca.channel contains "VAC:IP"`

USE CASES

CA protocol dissector for Wireshark is primarily useful for troubleshooting of EPICS deployments and development.

In this section, we present some hints on the usage of the CA protocol dissector:

- Combination with CA Snooper may enhance network trouble shooting, especially when more information is required than just record name resolution requests.
- Filtering of packets is extremely useful (the Expression button in the Wireshark user interface), as it reduces the clutter and allows the user to focus only on relevant packets. In particular, filtering by channel name and source/destination IP addresses are frequent.
- tshark can be used to capture packets, which can later be analyzed with Wireshark. This allows also for unattended capturing.

Currently, the Wireshark's CA dissector does not yet parse the data contents of the packet. Usually, this feature is not needed, as once networking issues are resolved, classical CA tools can be used to obtain the data more conveniently.

CONCLUSION

Fortunately, *EPICS Channel Access* and implementations that use it are now very stable and mature, thus it is only rarely required to look "under the hood" at what is going on at the network. Nonetheless, when an in-depth look into network-level activity was required, one had to have a thorough understanding of the structure of CA messages and the operation of the CA protocol.

With the dissector that we have developed, we hope that this task will now be greatly simplified, as most CA messages are now understood by the Wireshark tool and presented in a user-readable form. Furthermore, higher-level analysis of the communication is performed, so that instead of connection-specific channel IDs one sees channel names.

REFERENCES

[1] Wireshark, http://www.wireshark.org
[2] EPICS Documentation: Channel Access, http://www.aps.anl.gov/epics/docs/ca.php
[3] Channel Access Protocol Specification, http://epics.cosylab.com/cosyjava/JCA-Common/Documentation/CAproto.html
[4] Wireshark with EPICS Channel Access Dissector, http://www-linac.kek.jp/cont/epics/wireshark/