ANALYZING OFF-NORMALS IN LARGE DISTRIBUTED CONTROL SYSTEMS USING DEEP PACKET INSPECTION AND DATA MINING TECHNIQUES*

M. Fedorov, G. Brunton, C. Estes, J. Fisher, C. Marshall, E. Stout
Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550, USA

Abstract

Network packet inspection using port mirroring provides the ultimate tool for understanding complex behaviors in large distributed control systems. The timestamped captures of network packets embody the full spectrum of protocol layers and uncover intricate and surprising interactions. No other tool is capable of penetrating through the layers of software and hardware abstractions to allow the researcher to analyze an integrated system composed of various operating systems, closed-source embedded controllers, software libraries and middleware. Being completely passive, the packet inspection does not modify the timings or behaviors. The completeness and fine resolution of the network captures present an analysis challenge, due to huge data volumes and difficulty of determining what constitutes the signal and noise in each situation. We discuss the development of a deep packet inspection toolchain and application of the R language for data mining and visualization. We present case studies demonstrating off-normal analysis in a distributed real-time control system. In each case, the toolkit pinpointed the problem root cause which had escaped traditional software debugging techniques.

INTRODUCTION

In a distributed control system, the system components interact over a network, usually with the help of one of the middleware frameworks, such as CORBA (Common Object Request Broker Architecture). Having full details of such interactions traveling over the network opens up attractive opportunities for monitoring and debugging of the system. The high-end network routers usually provide a port mirroring feature which allows redirection of the interesting network traffic to a dedicated router port. At that port, a computer running a packet capture application (for example, open source tcpdump, [1]) or a dedicated appliance (e.g. OPNET ACE Live, [2]) receives the packets and stores them into a file, typically in the pcap format. After the capture, the offline analysis of the packets can be performed using a deep packet inspection tool, such as open source Wireshark [3]. The deep packet inspection tools are capable of recovering the high level protocol information from the raw capture files. In the case of CORBA, the object method names, object references and parameter values are recoverable. Given that the packet captures are timestamped with millisecond precision, an accurate application level trace of a network event can be reconstructed after an event occurred. An important advantage of the network-based monitoring is that it works equally well across a variety of server and embedded platforms, which is common for large control systems. Achieving the same breadth of monitoring with traditional debugging and logging tools would require significant effort running multiple native tools and then stitching log files together.

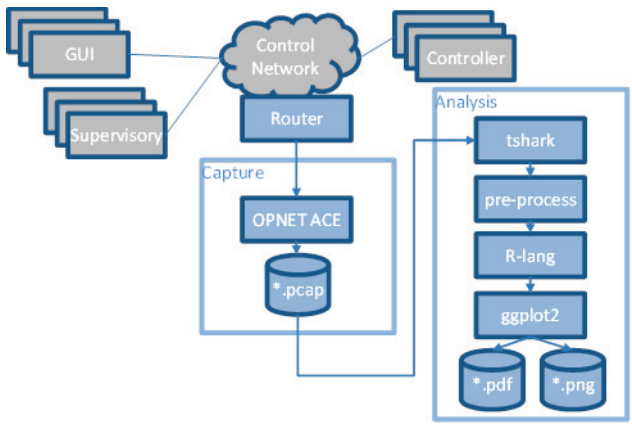

Figure 1: Deep packet inspection toolkit .

For a real-time control system, the introduction of any logging, debugging or profiling tool comes with a risk of drastically modifying the timings or behaviors of the monitored events. Network packet monitoring is stealthy: it is entirely passive; it does not modify the timing or sequencing of events. Since the packet capture and inspection tools run on dedicated computers, the computing resources of the core control system are not affected. The computing capacity allocated for packet monitoring can be increased or decreased based on situational needs.

In addition to the primary control system middleware communications, the network captures can include other protocols. For example, NFS (Network File System) packets are translated to file operations (open, close, read, write) by the deep packet inspection tools. Network-attached hardware events can be derived from the VISA/GPIB and MODBUS/TCP packets and others.

Setting up network packet monitoring for a control system does require additional resources. A high-end network router is usually required, as well as fast

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. #LLNL-ABS-631632

ISBN 978-3-95450-139-7

computers (or appliances) with high speed Network Interface Cards (NIC). Large fast storage is also needed, since even filtered network captures generate several gigabytes of pcap files per host per day.

Finding relevant events and visualization of the captured data often presents the biggest challenge. While both the proprietary (e.g. OPNET ACE Analyst, [2]) and open source (e.g. Wireshark) tools come with filters and graphs designed for typical network administration tasks, these tools can be very slow and require manual steps for identifying system-specific events of interest.

To address the need for problem-specific event finding and visualization tools, we have turned to the R language for statistical computing and graphics [4]. The R language is open source software, it is designed to handle large datasets and it has excellent and highly customizable visualization capabilities. As shown on Fig. 1, our toolkit also relies on tshark tool from the Wireshark package [3], a preprocessor script in Perl [5] and the ggplot2 package [6] from the CRAN (Comprehensive R Archive Network, [7]).

Using our toolkit, a number of off-normal behaviors were analyzed, understood and fixes were verified.

TROUBLESHOOTING SPORADIC SLOWDOWNS

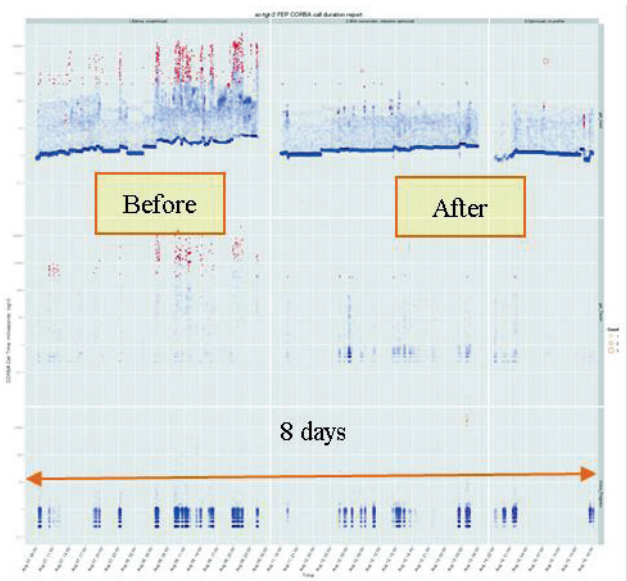

Figure 2: Call duration and retransmit diagram.

Our toolkit was applied to investigate performance degradations in a system with a distributed controller. The controller performed correctly all the time, but with intervals of extremely slow performance. During the slowdowns, the network response times were tens and hundreds times slower than usual. Also, increased TCP retransmissions were seen during the slowdowns.

Understanding the patterns of the slowdowns represented the largest challenge. While a normal transaction time is around 1ms, the slowdowns manifested in the scale of hours and days. Hundreds of thousands of operations were involved, most of them showing normal behaviors even during the slowdowns. With the toolkit, the call durations were plotted, partitioned by the call type and the TCP retransmit overlay was added, as shown on Fig. 2. Once visualized, the slowdowns patterns emerged. With the help of the API Breadth and Volume Diagram, the root cause was identified. A fix was developed, and once deployed, the toolkit confirmed the resolution of the problem.

IDENTIFICATION OF EVENT PATTERNS

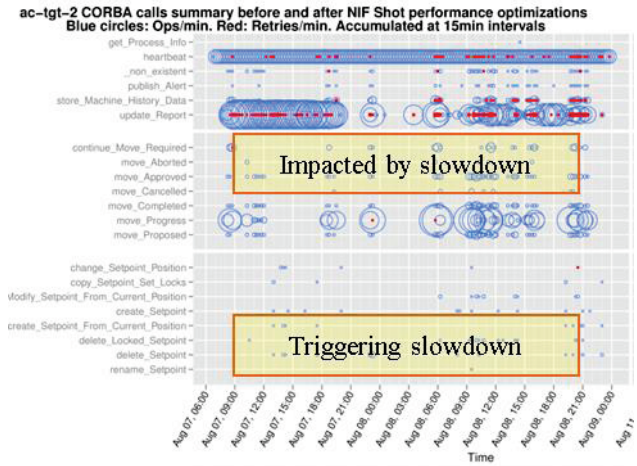

Figure 3: API breadth and volume diagram (fragment).

Once the general slowdown patterns emerged, the API Breadth and Volume Diagram was employed to identify the triggering sequences. This visualization is capable of covering millions of distributed operations over days of operations on one page. The volume of the high-frequency events was accurately represented, yet single and low-frequency events were not lost. Additional attributes (e.g. TCP retransmits) were overlaid and color-coded, as shown on Fig. 3.

Due to the comprehensive breadth of this diagram, even the low-frequency and single API operations were plotted. Usually, these would be lost in millions of high frequency events. Unexpectedly, certain low frequency operations appeared to be related to the beginnings of the slowdowns. A focused code and data inspection of these operations has uncovered CPU intensive program fragments which were processing an abnormally oversized dataset. Once the datasets were cleaned up and task scheduling was adjusted, the slowdown problem got resolved.

UNWINDING CONCURRENT INTERACTIONS

The biggest challenge of analyzing a previously unseen off-normal situation is to decide what metrics represent an informative event. The power and expressiveness of the R programming language and richness of its visualization tools allow the investigator to quickly iterate through the hypotheses. For example, when regularly occurring events (such as periodic polling) are analyzed, the previously considered Call Duration Diagram and API

ISBN 978-3-95450-139-7

Breadth and Volume Diagram may show very little change over time. However, when the exact timing of the event within the poll period is used as a metric, an informative visualization can emerge, such as shown on the Rigid Periodic Timeframe Diagram, Fig. 4. Here the poll rate is 1.0 second, so the relative event time varies between 0.0 and 1.0 seconds. The events which were originated from the two programs are color-coded green and blue. The strokes of these two poller tasks are highly regular when the system operates normally, and they are chaotic when the system is misbehaving.

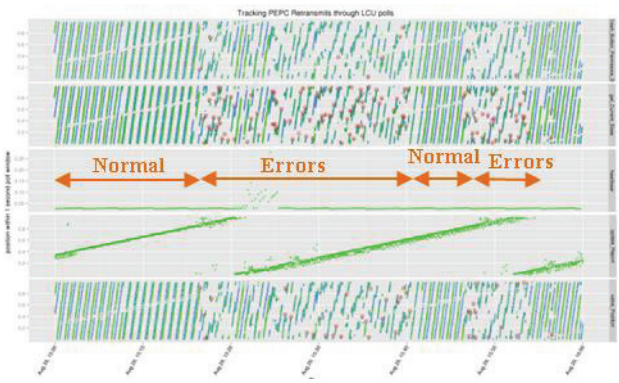

Figure 4: Rigid periodic timeframe diagram.

The first guess about the poller tasks interaction problem was further clarified by switching to a timeframe which is attached to the "blue" poller, as shown on Fig. 5.

The analysis and discovery of this concurrency issue was completed without significant knowledge of the system design and without accessing the source code of the affected program.

Figure 5: Floating timeframe diagram.

SUMMARY

By integrating commercial and open source tools, we have developed a data analysis and visualization toolkit for network packet captures. Our toolkit effectively processes raw network captures of several gigabytes per host. The R language is used to generate informative and intuitive visualizations. The toolkit is used to understand and address performance and timing issues in a large distributed control system.

REFERENCES

[1] tcpdump and libpcap, http://www.tcpdump.org.
[2] Riverbed Performance Management, OPNET http://www.riverbed.com
[3] Wireshark. Go Deep. http://www.wireshark.org
[4] The R Project for Statistical Computing, http://www.r-project.org
[5] The Perl Programming Language, http://www.perl.org
[6] ggplot2, http://ggplot2.org
[7] The Comprehensive R Archive Network, http://cran.r-project.org

ISBN 978-3-95450-139-7