WP-2013-PIC-1

Commissioning the PROFINET Netload Test Tool

Abstract

As the frequency of industrial cyberattacks has increased over the last several years, PROFINET devices are now required to undergo security testing via the "Netload Test." Since this software package can be installed on many different types of test computers, there can be significant variations in test results between installations. This paper describes the results of some baseline performance testing here in our lab, and some recommendations for commissioning test systems for use with the Netload Test.

Introduction

As we've begun using the Netload Test Tool, we have noticed that the test results varied between systems with slightly different hardware characteristics. The performance of the test system is critical to ensuring a successful and reproducible test; if either the NIC or processor lacks the performance to transmit data quickly enough, the netload Test Scripts will not generate the desired traffic load on the network and the test results will be deceivingly positive.

This paper covers the steps we took here in our lab to make sure our test system performed as expected and analyzes our results. Example code is included in Appendix A - Monitoring System Utilization from the Command Line.

Test System Components

The system hardware requirements from [1], section 3.4 are: PC with Quad-Core Processor, 2.20 GHz clock rate; 2048 MB RAM; 250 GB SATA II Harddisk (HDD); DVD +/- R/RW drive; Gigabit Ethernet Card. Our test system here has a dual-core Pentium D processor at 3.4 GHz and a 160 GB IDE HDD – both of these items lack the performance of their specified counterparts. Otherwise, our system met or exceeded the hardware requirements.

Our software was also slightly out of specification. While [1] calls for the Kanotix Operating System version 2.6.32, we installed the most up-to-date version - Kanotix Dragonfire with Kernel 3.2.0 and KDE SC 4.8.4. All other software components were installed according to specification.

Measuring System Performance

Our main concern with our test system's performance was the CPU: our dual-core Pentium D is much older and subpar to the specified "Quad Core Processor." If the Pentium D never reached 100% utilization, we considered the test system performance satisfactory. If, however, the processor's resources were completely consumed by executing the test scripts, we could assume that any disturbance to the system (opening a web browser, for instance) would invalidate our testing.

The Netload tests for Netload Class III were run twice each: once with the KDE GUI running, and once from the command line with no GUI running (server-style execution). The Ethernet interface and CPU activity were monitored during the execution of each case to watch for resource exhaustion. RAM and HDD performance were not monitored, since we determined early on that only ~200MB of RAM was used and the HDD had no active reads or writes during test script execution.

Kanotix comes with a number of built in system monitoring utilities, but most of them run in the X session and are of no use without the GUI. Therefore, we chose to install two command-line utilities, `ifstat` and `iostat`, and record our system's performance via log files (see

About the PROFI Interface Center: The PIC was established in Johnson City, TN to provide easy and direct access to PROFIBUS and PROFINET technology. The PIC provides design and development assistance, device certification, on-site troubleshooting and education for both PROFINET and PROFIBUS systems.

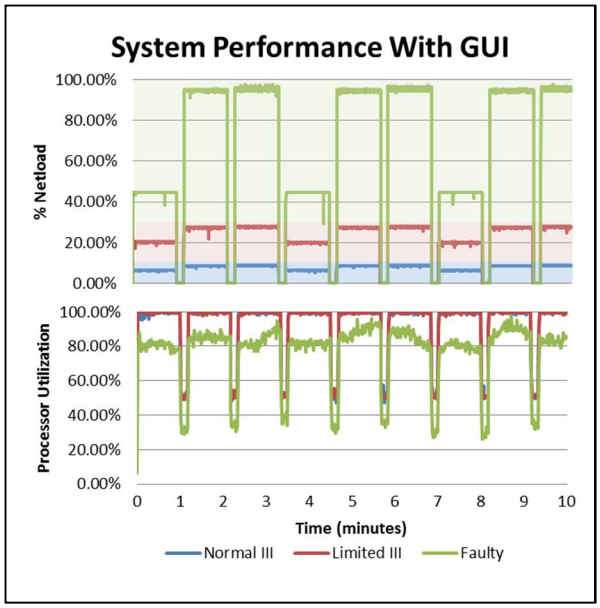

Figure 1 - Bandwidth and processor utilization for 10 different test cases in three operating modes, with the KDE desktop environment.

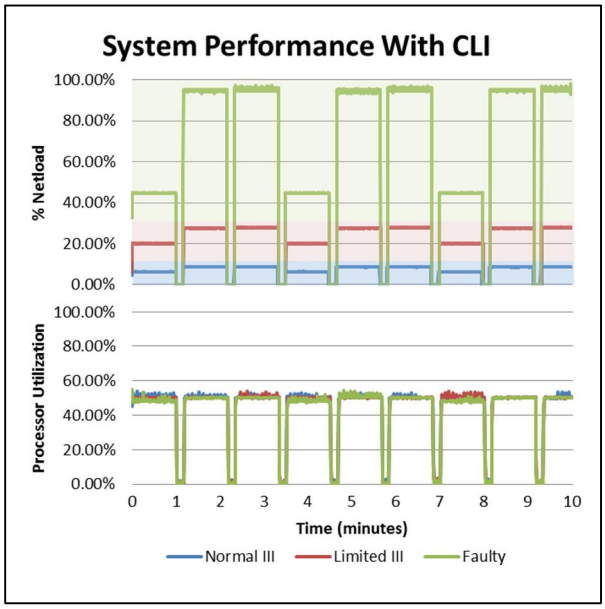

Figure 2 - Bandwidth and processor utilization for 10 different test cases in three operating modes without KDE or other GUI running in the background.

Appendix A - Monitoring System Utilization from the Command Line). These two utilities allowed us to monitor CPU and Network utilization, the two key bottlenecks on our test system, with a temporal resolution of 1 second and 100ms, respectively.

At the conclusion of these tests, we divided the bandwidth utilization by the transmission capacity of the Ethernet interface and compared the result to the "% Netload" specified for each test case from [2].

Analysis

To begin, we started the X server and KDE and ran the Normal, Limited and Faulty test scripts for Netload Class III certification. This run showed some "bursty" transmission characteristics, especially in the test cases that transmitted very small Ethernet frames (Figure 1). The CPU load reached 100% on all test cases.

Next, we ran the same scripts after terminating the X server, with results shown in Figure 2. While the data show only 50% processor usage, this is due to the full utilization of one of two cores executing the Netload test scripts in one process. The Ethernet transmission characteristics improve, and are compared with the previous results in Table 1. A histogram of the data from the first test case – the first minute of measurement - is shown in Figure 3, showing the decreased deviation (decreased "burstiness") of the transmissions using the Command Line Interface.

Table 1 - Transmission Statistics for Faulty Test Case 1.

Trial	Mean (Kbps)	Std. Dev. (Kbps)
With GUI	44606	1681
With CLI	44658	1005

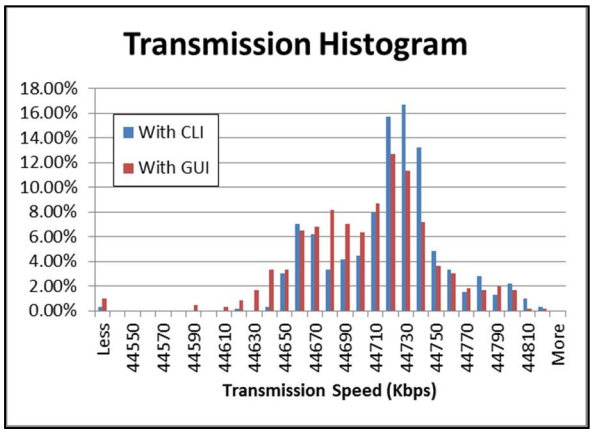

Figure 3- Histogram of transmission speed during the first Faulty Netload Test Case (undirected network load)

Conclusions and Recommendations

Quantifying the performance of the netload tester is critical to eliminate a large source of potential error in the test results. Because the software does not gather any performance data about the device under test, it's impossible to know if your test fixture yields valid results without commissioning it.

We have proposed and demonstrated a simple method to quantify the performance of a Netload test station using standard Linux system utilities. The results of our measurements show limited benefit to increasing the processor core count on test systems. Since the test scripts are executed in a single process, they must be executed sequentially and gain little from parallel processing. However, gains can be seen from reducing the "background" load on each core to allow the Netload test scripts to fully utilize the CPU resources on the test platform.

Work remains to quantify the contribution of an interface's buffer characteristics or other extraneous variables. These unknowns should always be kept in mind when using the Netload Test Tool – knowing the behavior of each individual installation is key to a successful netload test.

References

[1] PI Working Group 2, Committee 2, "Test Cases for "PROFINET IO Security Level 1," Version 1.1.0," PROFIBUS & PROFINET International, Karlsruhe, 2013.

[2] PROFIBUS Working Group 11, Technical Committee 2, "PROFINET IO Net load - Guideline for PROFINET, Version 1.0," PROFIBUS Nutzerorganisation e.V., Karlsruhe, 2010.

Appendix A - Monitoring System Utilization from the Command Line

Linux can get complicated – quickly – but that shouldn't be an impediment to using the Netload Test Tool. We've put together a step-by-step tutorial to help users new to Linux get comfortable on the command line by monitoring your system utilization and highlighting some common Linux tips and tricks along the way. In doing so, we hope to simplify and clarify many of the basic command line operations and give you the basic techniques necessary to begin tackling more involved issues and complex goals.

This example uses two common programs: `ifstat` to monitor the Network Interface Card (NIC) and `iostat` to monitor CPU or RAM utilization. `ifstat` is available as a standalone package, `iostat` is part of the `sysstat` package.

So let's start with installing the tools we'll need. We'll use the program `apt-get` to install these from the internet. Open a terminal window by clicking on the "K" in the lower ribbon bar of the desktop, selecting "Applications", then "System" and finally "Terminal."

Once the terminal is open, type

```
~$ sudo apt-get install ifstat sysstat
```

This is really several commands rolled in to one. The first, `sudo`, tells the system to execute the rest of the command as a "superuser." The second, `apt-get`, is a program used to install and uninstall programs in Kanotix. It handles dependencies, versioning, and all the other software management details for you. `install` is an argument to the function `apt-get`, and finally the list of packages to be installed, `ifstat` and `sysstat`.

Next, you'll need to enter the `sudo` password for your user account. By default, this is the same as the password you use to log in to the machine as a regular user. You'll also have to proceed through a pretty straightforward installation process.

So, once the packages are installed, we can see what they do with the `man` command.

`~$ man iostat` or `~$ man ifstat`

This is a pretty standard "manual" page that most Linux programs include in their installation package. Type "`q`" to exit the manual.

From the man page, the command and options to sample the CPU utilization at the fastest speed (1 second) is

```
~$ iostat –c 1
```

This will start a rolling display of the CPU utilization (exit with "CTRL+C"). The information is nice, but we need to capture it to do anything with it. We can save the output to a text file by redirecting it with the ">" symbol.

```
~$ iostat –c 1 > logfile.txt
```

This will create (or overwrite) the file `logfile.txt` in the working directory and fill it with the text that is normally sent to the terminal window. Unfortunately, this holds the command prompt and keeps you from doing any useful work while you're monitoring your system. To work around this, you can append the "&" symbol at the end of the command to launch the process in the background.

```
~$ iostat –c 1 > logfile.txt &
```

The program will report its job number ([1], [2], [3], etc) and Process ID (PID) number. You're free to run whatever other programs you like from this point on, but you can always bring this background process back to the foreground with the command

```
~$ fg
```

If you like, you can pass `fg` the job number as an argument to select from multiple background processes.

```
~$ fg 1
```

Once the process has been moved to the foreground, you can kill it as usual with "CTRL+C".

We can do something similar with `ifstat`, sampling from interface `eth1` at 10 times per second and recording the results in units of Kbps.

```
~$ ifstat –i eth1 –b 0.1
```

```
~$ ifstat –i eth1 –b 0.1 > logfile.txt
```

So, we have covered how to monitor your processor and NIC. But there's still one last problem: how to work as a Superuser without typing `sudo` all the time while you're trying to run the Netload Test Scripts. To open a persistent Superuser prompt, use the command

```
~$ sudo –s
```

This will elevate you to "root" privileges (your command prompt will change from `~$` to `~#`. To exit your root session, just type `exit`.

To put it all together, you can gain root access to the computer, move to your netload test directory, start your monitoring processes, and then start your netload test script. Here's an example of the code needed to run a test script, written as though the code was executed from a fresh terminal window opened on the desktop or in a completely new session.

```
~$ sudo –s
~# cd
~# cd KTpnio_NetzlastSequenzen/[SOME TEST DIRECTORY]/
~# iostat –c 1 > /home/[USERNAME]/Desktop/iostatlog.txt &
~# ifstat –i eth1 –b 0.1 > /home/[USERNAME]/Desktop/ifstatlog.txt &
~# ./Batch_*Communication_01-06-2010
```

And for bonus points, you can roll those last three commands in to one by separating them with spaces. (An added benefit of launching `iostat` and `ifstat` as background processes.)

```
~# iostat –c 1 > /home/[USERNAME]/Desktop/iostatlog.txt & ifstat –i eth1 –b 0.1 >
/home/[USERNAME]/Desktop/ifstatlog.txt & ./Batch_*Communication_01-06-2010
```

This will set you off and running with your Netload Test script while still monitoring your system performance. The log files can be analyzed for any performance bottlenecks during the script's run time.