AN2013-1
Gryphon®2 Application Note

Using and Programming Gryphon for Engineers and Programmers

by DG Technologies Product Engineering

Document Revision: 0.3 (DRAFT) of 01/23/2014

Covers basic Gryphon programming tools, including gryphrx gryphtx, gschdtx, and using Wireshark.

CONTENTS

1 INTRODUCTION

This Application Note is for those engineers who want to begin using and writing application programs for the Gryphon® family of network protocol adapters. You will become familiar with verifying protocol communications on the Gryphon, configuring the Gryphon for various protocol modes, and diagnosing potential issue with your setup. This Application Note also introduces some essential command-line utilities that come with the Gryphon product.

2 ASSUMPTIONS AND AUDIENCE

This Application Note assumes that you are familiar with setting up the Gryphon, setting an IP address for the Gryphon, and logging in using Telnet and a web browser. If you are not, refer to the Getting Started PDF files in the Gryphon product directory. This Application Note also assumes you are somewhat familiar with using basic Linux commands and some command shell. You need to be familiar with using the web interface on the Gryphon, and with using telnet to establish a login prompt and log in to the Linux command-line shell on the Gryphon.

3 SCOPE OF THIS APPLICATION NOTE

The Gryphon products are feature-rich, fully functional multi-protocol adapters running Linux. As such, the Gryphon has many features and utilities that allow engineers to develop networked ECUs and diagnose protocol networks and ECUs on those networks. This introductory Application Note covers a few basic functions of using the Gryphon with networks, and using a few command-line utilities that are essential for becoming experienced using the Gryphon.

4 PRODUCT BACKGROUND

The Gryphon family of products are a set of automotive and vehicle network protocol adapters. The Gryphon has protocol card slots that accept protocol adapter cards for various standard automotive and truck protocols. Gryphons are used primary in engineering development applications, and for manufacturing programing and testing of ECUs.

DG Technologies manufactures two versions of the Gryphon product, the Gryphon 2 and the S4. The Gryphon 2 has a faster processor, three (3) protocol card slots, digital I/O, three (3) USB host ports, wired Ethernet, and front panel text and control buttons. The Gryphon 2 can have up to twelve (12) channels of protocol communication by populating the slots with, for example, three quad CAN cards. The S4 has two (2) built-in channels of CAN protocol, and one (1) protocol card expansion slot. The products run the Linux operating system and communicate with a host computer through a standard Ethernet network. The S4 also has optional wireless Ethernet.

The supported protocols include: CAN, GMLAN, GMUART, ISO9141-2, ISO11898 (CAN), ISO15765, J1850 GM (Class 2), J2284, J2411 (GM SWCAN), J2534, KWP2000 (ISO14230), and LIN.

5 BASIC ARCHITECTURE

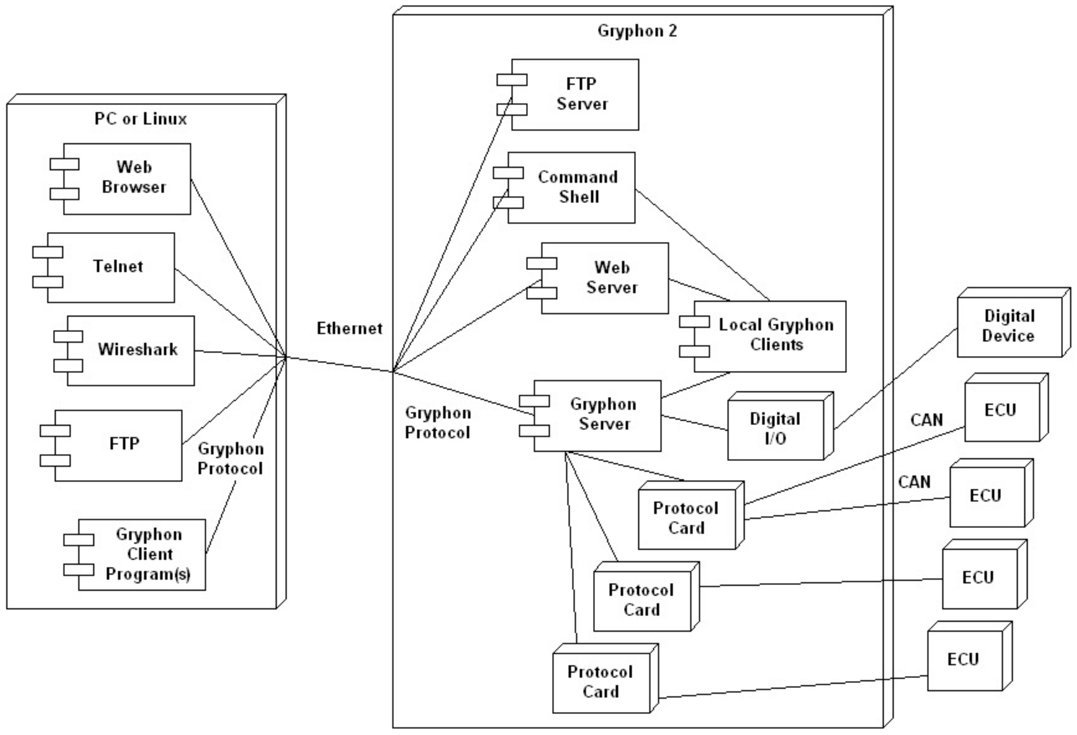
The Gryphon sits in the middle of the communications path between a PC or Linux computer and one or more ECUs. The Gryphon connects to programs running on a PC or Linux computer using Ethernet, and connects to one or more ECUs using one or more ECU network protocols. Each ECU network protocol on the Gryphon is a channel and is given a channel number by the Gryphon beginning with channel 1. You interact with the Gryphon in many different ways, including:

- You connect with the Gryphon web interface using a standard web browser, just like any other web server running on a Linux computer.
- You execute command-line programs on the Gryphon by logging in to the Linux shell using a remote-shell program such as Telnet or PuTTY.
- You copy files to and from the Gryphon using FTP, just like any other Linux computer.
- Application programs you write, called Gryphon clients, communicate the Gryphon sever using Gryphon Protocol on Ethernet over TCP/IP.

Figure 1 is a UML deployment diagram showing the basic components and connections. It shows a typical configuration with a dual CAN protocol card as an example.

Figure 1: Basic Gryphon Components and Connections

Simplified Gryphon Deployment Diagram



Your application programs receive ECU protocol messages, transmit protocol messages, and get and set protocol card configurations. Your program communicates to the hardware through a software component known as the Gryphon Server. Later sections in this Application Note show specific examples in detail.

Using an open source program called Wireshark, you can monitor the Gryphon Protocol communication between your Gryphon client program on your PC and the Gryphon Server software process running on the Gryphon. Wireshark comes with a built-in plug-in that understands Gryphon Protocol and interprets the byte stream into English. (The plug-ins in Wireshark are called "dissectors".) Later sections in this Application Note show examples using Wireshark.

The remain sections in the Application Note will discuss many of the PC and Linux user programs you need to be familiar with to use the Gryphon products.

6 BASIC PROTOCOL UTILITIES: GRYPHRX AND GRYPHTX

The Gryphon comes with two very useful low-level software diagnostic programs called gryphrx (for receiving) and gryphtx (for transmitting). You use gryphrx to show received messages on the protocol channels, and you use gryphtx to transmit messages on the protocol channels.

These command-line programs allow you to diagnose your protocol networks and verify basic working setup.

To run command-line programs on the Gryphon, Telnet into your Gryphon and login as user "root" (the default password is "dggryphon"). When you login, you get a shell command prompt that looks like:

```
gryphon ~#
```

When you run gryphrx on the command line, you get the following usage message:

```
Usage: gryphrx hostname [-p port ] [-c channel] [-s] [-r]
        -s    option enables timestamp sorting
        -r    option selects relative timestamp display
        -x n  option checks for sequenced data field
              outputs errors and every "n" messages
        -j    uses J1939 protocol layer
```

The "hostname" argument is optional. The "port" argument is optional. An example gryphrx command is:

```
gryphon ~# gryphrx -
```

This command monitors protocol traffic on all channels, and the first few lines of output for CAN for example looks like the following:

```
Ch  1 * Type 02/17. DG-CAN-SJA1000-C    ver.1.5.4, Ds=00-08, Es=00-00, Hs= 2 4
Ch  2 * Type 02/17. DG-CAN-SJA1000-C    ver.1.5.4, Ds=00-08, Es=00-00, Hs= 2 4

000001 RX Ch 01:  T: 27982866570 H: 01 F3  D: 80 80 00
000002 RX Ch 01:  T: 27982869170 H: 05 41  D: FE 00 00 00 00 00 00 00
000003 RX Ch 01:  T: 27982871600 H: 00 F1  D: 8D 00 00 40
000004 RX Ch 01:  T: 27982874140 H: 04 51  D: 00 00 00 00 00 00
000005 RX Ch 01:  T: 27982881860 H: 05 41  D: FD 00 00 00 00 00 00 00
```

gryphrx reports the type of protocol on each channel (in this example, CAN SJA1000), and then display the protocol traffic. The first column is the line number, then the timestamp follows the T:, and header and data are shown after the H: and D:.

Using the Linux commands, you can pipe the output of gryphrx to Linux programs such as grep. For example,

```
gryphon ~# gryphrx - | grep "H: 07 E0"
```

will filter out all message except the 07 E0 messages. Another example,

```
gryphon ~# gryphrx - | grep -v  "H: 07 E1"
```

will filter out messages with the 07 E1 header. Another example,

```
gryphon ~# gryphrx - | egrep -v  "H: 07 E1| H: 07 E2"
```

will filter out messages with the 07 E1 header and messages with the 07 E2 header.

You will note that gryphrx takes a hostname as the first argument. This allows you to monitor protocol data on another Gryphon. You can use any valid IP address for any Gryphon on your network. To view your Gryphon's IP address, use the command "ifconfig".

When you run gryphtx on the command line, you get the following usage message:

```
Usage: gryphtx hostname arg1 arg2 ...

Arguments:
        -p portno       Set server port on hostname
        -c channel      Set channel number (default=1)
        -i iterations   Set number of transmit iterations (default=1, max=-1)
        -t delay        Set delay between iterations in mS (default=1)
        -s stat         Set status field in message (device specific use)
        -r              Set remote bit in the Mode field
        -j              Use J1939 protocol
        -++             Increment the data field; use delay driver
        -a              Increment the data field; don't use delay driver
        -h byte byte... Message header bytes
        -d byte byte... Message data bytes
        -e byte byte... Message extra bytes
        -k id           Kill schedule id
```

An example gryphtx command for CAN is:

```
gryphon ~# gryphtx -c 1 -h 07 e0 -d 01 02
```

This command sends a single diagnostic tester message on channel 1. To send multiple message, use the -i option. For example,

```
gryphon ~# gryphtx -c 1 -h 07 e0 -d 01 02 -i 10 -t 100
```

This command starts a Gryphon schedule that will send this message 10 times at 100ms between each message. You get output that looks something like this:

```
Scheduler: Handle = 0x0000033f
```

where the 0x0000033f is the process id of your Gryphon schedule. This example schedule will run for 1 second, and then the process will end. To kill a running schedule before it ends, issue another gryphtx command with the -k option, such as:

```
gryphon ~# gryphtx -k 0x0000033f
```

When the schedule is successfully killed, the command will output the following:

```
Scheduler: OK (00000000)
```

In addition to running gryphrx and gryphtx on your Gryphon, DG supplies the source code to the gryphtx and gryphrx utilities, along with a Linux Makefile and library routines, so you can compile and run these utilities on any external Linux computer.

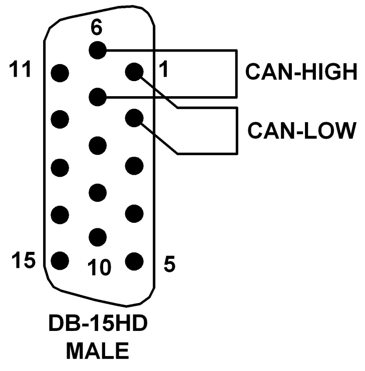
More complex transmissions can be run using the command gschdtx discussed below.

7 USING LOOPBACK TO ANOTHER GRYPHON CHANNEL TO VERIFY COMMUNICATIONS

A good way to verify the basic operation of your protocol bus is to use another Gryphon channel (of the same protocol) to monitor a test message. You can transmit messages from one channel and receive the messages on other channel to verify that your channel is transmitting. You can do the opposite to verify that your channel is receiving.

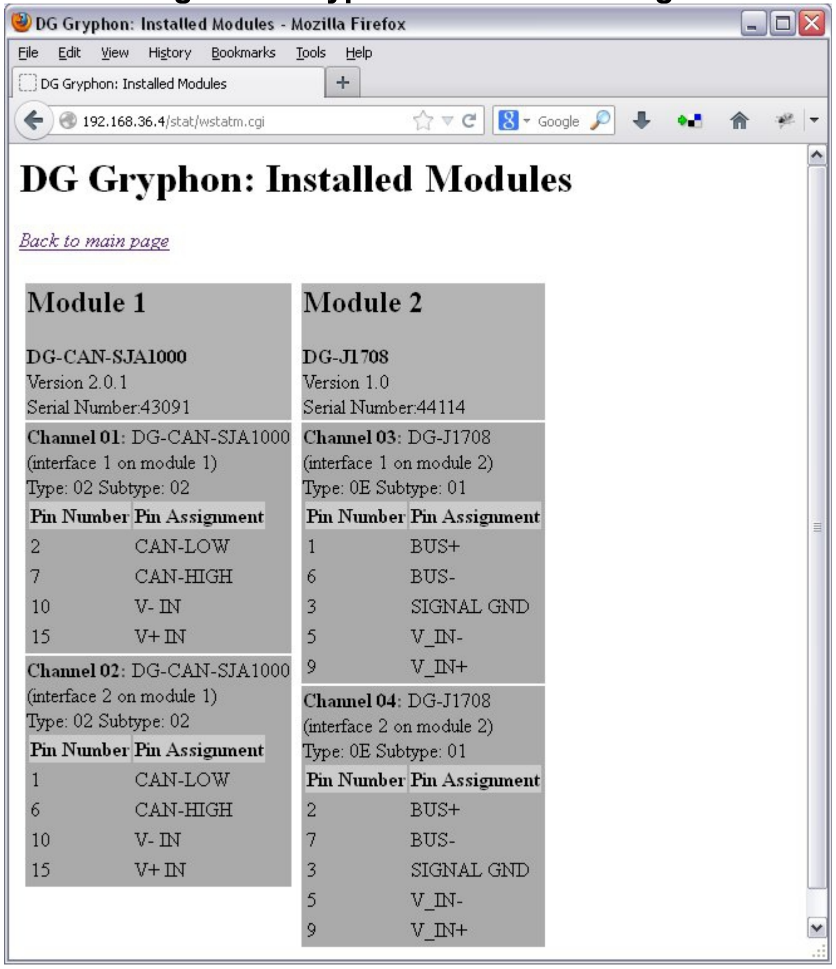
Using a CAN card for example, a channel 1 and channel 2 loopback connector would look like the following:

Figure 2: CAN Protocol Loopback Connector for Channels 1 and 2



You create a loopback connector by looking up the connector pinout of your card connector on the Gryphon web page, which looks like the following:

Figure 3: Gryphon Pinout Web Page


DG Gryphon: Installed Modules - Mozilla Firefox
File Edit View History Bookmarks Tools Help
DG Gryphon: Installed Modules
192.168.36.4/stat/wstatm.cgi
Google

DG Gryphon: Installed Modules

Back to main page

Module 1

DG-CAN-SJA1000
Version 2.0.1
Serial Number:43091

Channel 01: DG-CAN-SJA1000
(interface 1 on module 1)
Type: 02 Subtype: 02

Pin Number	Pin Assignment
2	CAN-LOW
7	CAN-HIGH
10	V- IN
15	V+ IN

Channel 02: DG-CAN-SJA1000
(interface 2 on module 1)
Type: 02 Subtype: 02

Pin Number	Pin Assignment
1	CAN-LOW
6	CAN-HIGH
10	V- IN
15	V+ IN

Module 2

DG-J1708
Version 1.0
Serial Number:44114

Channel 03: DG-J1708
(interface 1 on module 2)
Type: 0E Subtype: 01

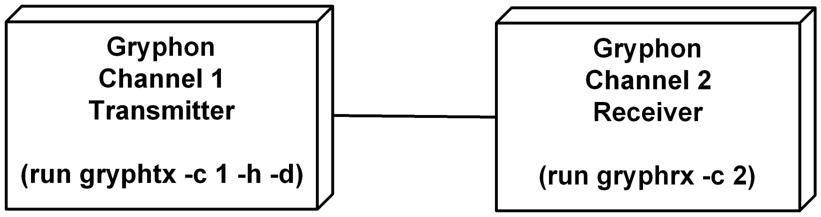
Pin Number	Pin Assignment
1	BUS+
6	BUS-
3	SIGNAL GND
5	V_IN-
9	V_IN+

Channel 04: DG-J1708
(interface 2 on module 2)
Type: 0E Subtype: 01

Pin Number	Pin Assignment
2	BUS+
7	BUS-
3	SIGNAL GND
5	V_IN-
9	V_IN+

Using the loopback connector, transmit on one channel and receive on the other channel, for example:

Figure 4: CAN Protocol Loopback



8 SCHEDULING TRANSMITS: USING GSCHDTX

Another useful utility program that comes with the Gryphon is gschdtx. This is more advanced version of gryphtx because it allows you to schedule more complicated transmits than you could by using just gryphtx alone. When you run gschdtx on the command line, you get the following usage information:

```
Usage: gschdtx hostname [-p port] [-i iterations] [-C] [-w] [-f file]

        Default port is 7000.
        Default channel is 1, normally overridden in each message.
        -i <integer>  specifies number of times to send the whole list,
                      default is 1.
        -C  selects critical scheduler.
        -w  wait for schedule to finish before exiting.
        -f  specifies datafile, default is stdin.

Datafile takes one message per line, in the following format:

[-s sleep] [-q quantity] [-p period] [-c channel] [-h hdrbyte hdrbyte ...] [-d
databyte databyte ...] [-e extrabyte extrabyte ...]

        -s <integer>  specifies time to wait before starting this
                      message loop, in milliseconds.  Default is zero.
        -q <integer>  specifies number of times to transmit this message.
                      Default is one.
        -p <integer>  specifies time to wait after each transmit,
                      in milliseconds.  Default is zero.

Header, data, and extra bytes are (pairs of) hex digits, e.g. "0a db"

Alternate usage:  gschdtx hostname [-p port] -k id
kills the schedule with job id specified by -k.
```

The gschdtx command takes as an argument a file that contains your message data and message timing information. An example command-line might look like:

```
gryphon ~# gschdtx localhost -i 10 -f mydata.txt
```

This example command will run the file mydata.txt 10 times and then end the schedule. The data file might look like:

```
-p 20 -c 1 -h 07 e0 -d 01 02
-p 30 -c 1 -h 07 e2 -d 03 04
-p 20 -c 1 -h 07 e0 -d 05 06 07
```

This will transmit the first message with data 01 02, wait 20ms, transmit the second message with data 03 04, wait 30ms, transmit the third message 05 06 07, and wait for 20ms. If you use the -i option with gschdtx as in the example above, it will transmit this sequence of three message 10 times, and then the schedule will end.

Use FTP to copy your message data file to the Gryphon. Please note that you will need write-access to the file system to write files. To obtain file system write-access, see the next section below.

Just like gryphtx, if you want to kill the schedule before it finishes, use the -k option.

9 FILE SYSTEM WRITE ACCESS

The Linux file system on the Gryphon is stored on the internal SD card flash. By default, the flash is read-only on all partitions except for the /data partition. When you want to write to files in the root file system, you need to first turn off the flash write-protection, make your changes, then turn back on the flash write-protection. The Gryphon comes with two commands to do this, wpflashoff and wpflashon. They are run with no arguments, for example,

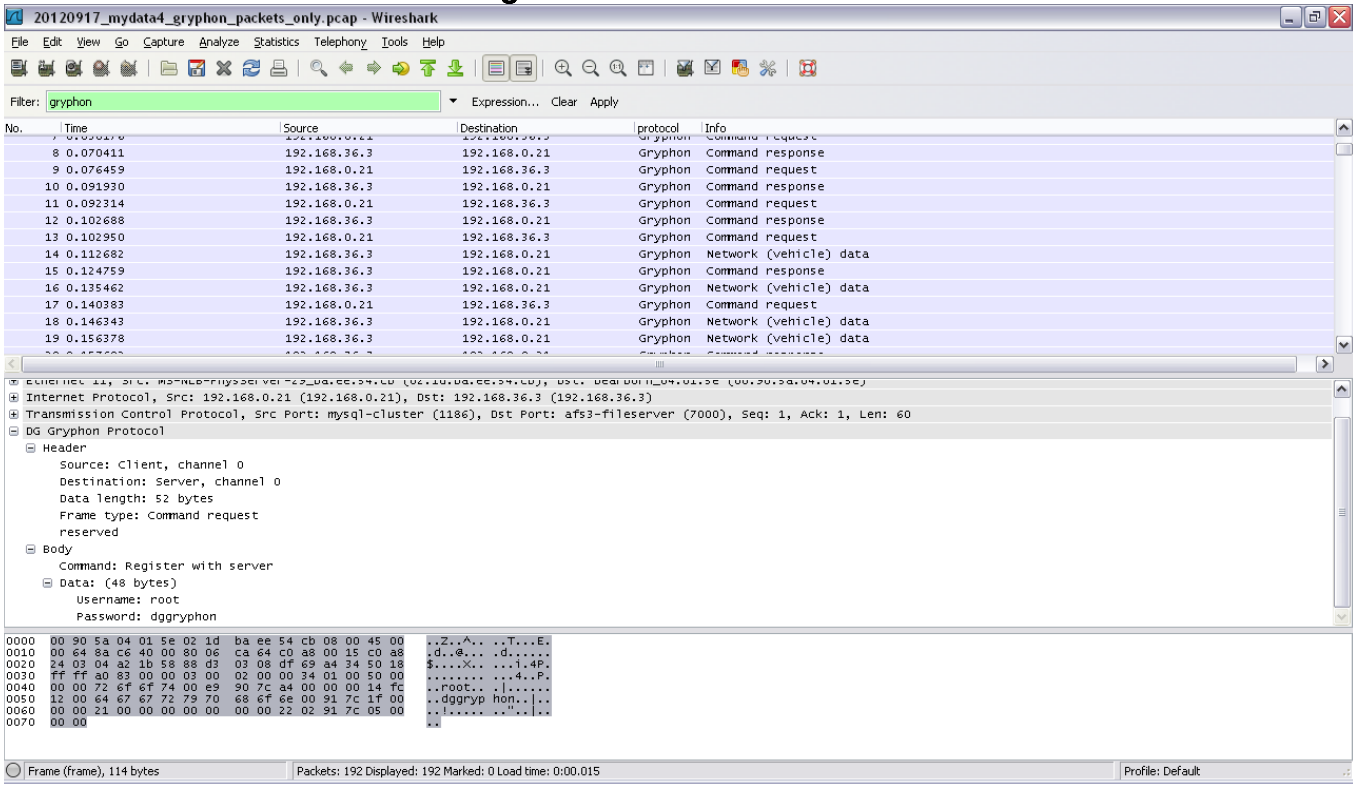
```
gryphon ~# wpflashoff
gryphon ~# wpflashon
```

Your data and log files stored on the /data partition are always writable and you do not need to run these commands to delete files in the /data directory.

10 GRYPHON PROTOCOL AND WIRESHARK

Wireshark is an open-source tool that allow you to see the network traffic between your Gryphon and your client application program. Wireshark runs on either Windows or Linux. A typical output window for CAN looks like the following:

Figure 5: Wireshark Screen


20120917_mydata4_gryphon_packets_only.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: gryphon
Expression... Clear Apply

```
No.  Time       Source         Destination    protocol Info
   8 0.070411   192.168.36.3   192.168.0.21   Gryphon  Command response
   9 0.076459   192.168.0.21   192.168.36.3   Gryphon  Command request
  10 0.091930   192.168.36.3   192.168.0.21   Gryphon  Command response
  11 0.092314   192.168.0.21   192.168.36.3   Gryphon  Command request
  12 0.102688   192.168.36.3   192.168.0.21   Gryphon  Command response
  13 0.102950   192.168.0.21   192.168.36.3   Gryphon  Command request
  14 0.112682   192.168.36.3   192.168.0.21   Gryphon  Network (vehicle) data
  15 0.124759   192.168.36.3   192.168.0.21   Gryphon  Command response
  16 0.135462   192.168.36.3   192.168.0.21   Gryphon  Network (vehicle) data
  17 0.140383   192.168.0.21   192.168.36.3   Gryphon  Command request
  18 0.146343   192.168.36.3   192.168.0.21   Gryphon  Network (vehicle) data
  19 0.156378   192.168.36.3   192.168.0.21   Gryphon  Network (vehicle) data
```

```
⊞ Internet Protocol, Src: 192.168.0.21 (192.168.0.21), Dst: 192.168.36.3 (192.168.36.3)
⊞ Transmission Control Protocol, Src Port: mysql-cluster (1186), Dst Port: afs3-fileserver (7000), Seq: 1, Ack: 1, Len: 60
⊟ DG Gryphon Protocol
  ⊟ Header
      Source: Client, channel 0
      Destination: Server, channel 0
      Data length: 52 bytes
      Frame type: Command request
      reserved
  ⊟ Body
      Command: Register with server
    ⊟ Data: (48 bytes)
        Username: root
        Password: dggryphon
```

```
0000  00 90 5a 04 01 5e 02 1d  ba ee 54 cb 08 00 45 00   ..Z..^.. ..T...E.
0010  00 64 8a c6 40 00 80 06  ca 64 c0 a8 00 15 c0 a8   .d..@... .d......
0020  24 03 04 a2 1b 58 88 d3  03 08 df 69 a4 34 50 18   $....X.. ...i.4P.
0030  ff ff a0 83 00 00 03 00  02 00 00 34 01 00 50 00   ........ ...4..P.
0040  00 00 72 6f 6f 74 00 e9  90 7c a4 00 00 00 14 fc   ..root.. .|......
0050  12 00 64 67 67 72 79 70  68 6f 6e 00 91 7c 1f 00   ..dggryp hon..|..
0060  00 00 21 00 00 00 00 00  00 00 22 02 91 7c 05 00   ..!..... .."..|..
0070  00 00                                              ..
```

```
Frame (frame), 114 bytes    Packets: 192 Displayed: 192 Marked: 0 Load time: 0:00.015    Profile: Default
```

Wireshark in the figure has three output window areas. The top window shows the Gryphon protocol commands, the middle window shows the English interpretation of the selected command, and low windows shows the raw data bytes. In the middle window, you see an example Gryphon protocol command with a header section and body. Each Gryphon protocol packet has a header and body. The header lists the source and destination of the command, as well as the data size. The example in the figure shows the "register with server" Gryphon command. You can find a full listing of the Gryphon protocol commands in the Gryphon web page documentation. Wireshark is very valuable tool for debugging your client application programs.

11 USING VIRTUALBOX LINUX ON WINDOWS AND WIRESHARK

Even if you run Linux in VirtualBox on Windows, you can use Wireshark to capture the Gryphon Protocol packets between your Linux application running in VirtualBox and the Gryphon.

12 APPLICATION PROGRAM C AND C++ LIBRARIES FOR WINDOWS

DG provides C and C++ Windows libraries for building Windows client programs for Gryphon. See the C and C++ library API documentation. Your client programs can also be debugged using Wireshark.

13 GRYPHON DEVICE DRIVER INFO FILES

The Gryphon several log files in the /gryphon/dev directory. The main log file is called gerrout, and it contains all error messages from all Gryphon software. You view the log file by using a Linux command such as cat (or a command such as more) as follows:

```
gryphon ~# cat /gryphon/dev/gerrout
```

Once you view the messages, the log file is cleared, so if you cat gerrout again it will contains no messages and you see no output.

Each channel on the Gryphon has it's own status file in the /gryphon/dev directory, called ginfo01 through ginfo31. Files ginfo01 through ginfo30 are for the protocol card channels, and file ginfo31 is the status file for the Gryphon Delay Driver, which controls the real-time transmitting of all messages. You read the status by cat-ing the file, for example:

```
gryphon ~# cat /gryphon/dev/ginfo01
```

Some example output might look like the following:

```
Gryphon SJA1000 CAN driver
Based on the Gryphon 82527 CAN driver by R.Arnold
Copyright 2001 Dearborn Group, Inc.
Compiled: Mar 31 2011, HZ=250
Auto resets=0

qtybusoff=0 qtywarn=0 qtyunknownerrint=0 lastunknownerrstat=0x00
irq=5 qtyunknownir=0 lastunknownir=0x00 MAXQTYUNKNOWNIR=5000000


SJA1000 @ c00d0000  -  ok, open, loop on, multi msg reads, err active,
bus  load  monitor  off,  errlevel  0  (0),  internal  transceiver,  external
termination
btrs=0x80,0x2b (0x80,0x2b)   bc=0xda (0xda)   rfmode=0 (0)

irqs none: 0  A: 44597  B: 0  A&B: 0
Chip status: 0C   flag: 0   IRQ Enable: 07    IRQs: 00
On counters: 0  0    Off counters: 0  0  0
IRQs: 44493, 104, 0, 0, 0, 0, 0, 0, 0   empty=0

out.putend=0 out.getend=0 (0)
out.msgs=c2912f7c CANBUFSIZE=1001 (x 28 bytes)
out.total=1   out.bustotal=1   out.chip_dropped=0   out.dropped=0   out.bad=0
out.neterrors=0 out.netbits=0
out.arblost=0 out.overrun=0 out.partial=0
out.sleeping=0 out.wqp=c0a69088

in.putend=557 in.getend=557 (0)
in.msgs=c1078040 CANBUFSIZE=1001 (x 28 bytes)
in.total=44601   in.bustotal=44493   in.chip_dropped=0   in.dropped=0   in.bad=0
in.neterrors=0 in.netbits=0
in.arblost=0 in.overrun=0 in.partial=0
in.sleeping=0 in.wqp=c0a6906c

crit.putend=0 crit.getend=0 (0)
crit.msgs=c2912f0c CRITBUFSIZE=11 (x 4 bytes)
crit.total=103 crit.dropped=0 crit.bad=0
```

This example is for a CAN SJA1000 protocol card. The status shows the type of channel, the overall status of the channel, the count of the number of bus-off and bus-warnings seen, and the internal buffer status of the output, the input, and the critical output scheduler buffers. You can tell if there are any message in the queue waiting to be transmitted, or any messages in the receive buffer waiting to be read by your client application program.

14 GRYPHON LINUX CONSOLE

If you ever need access to the Linux console status messages, the Gryphon contains an RS232 serial port header inside the Gryphon. The header is a standard DCE serial port.

15 SOFTWARE UPGRADES AND NEW RELEASES

From time to time, DG Technologies puts out new Gryphon software releases. You can obtain these releases on the DG Technologies web site and load a new release into the Gryphon using a Windows-based reflash program. See the Gryphon User Manual for more details.

FURTHER READING

AN2013-02 "Gryphon®2 Application Note "Gryphon Programming 2 for Engineers and Programmers", 2013 DG Technologies.

"Gryphon C++ Class Library User's Manual", 2006 Dearborn Group, Inc.

"Gryphon C Library User's Manual", 2006 Dearborn Group, Inc.

REFERENCES

Gryphon web page documentation.

"Gryphon C++ Class Library User's Manual", 2006 Dearborn Group, Inc.

Wireshark User's Guide for Wireshark 1.4, Copyright 2004-2010 Ulf Lamping, Richard Sharpe, Ed Warnicke.

ATTRIBUTIONS

Linux is licensed under the GNU General Public License (GPL) and the full name of the software system is GNU/Linux.

Ethernet is the name of the IEEE 802.3 standard LAN technology.

Wireshark is a free and open-source network packet analyzer released under the terms of the GNU General Public License (GPL).

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

VirtualBox is a registered trademark of Oracle Corporation.

The stylized name *Gryphon*®2 is a registered trademark of DG Technologies, Inc. The name Gryphon, Gryphon 2, and Gryphon S4 are trademarks of DG Technologies, Inc.

Unified Modeling Language (UML) is a standardized general-purpose modeling language adopted by the Object Management Group (OMG) and accepted by the International Organization for Standardization (ISO) as industry standard for modeling software-intensive system.

COPYRIGHT NOTICE

CONTACT INFORMATION

U.S. Sales and Corporate Office
DG Technologies, Inc.
33604 West Eight Mile Road
Farmington Hills, Michigan 48335-5202
Phone: 248.888.2000
sales@dgtech.com
http://store.dgtech.com

Heavy-Duty Development Center
DG Technologies, Inc.
2415 Directors Row, Suite G
Indianapolis, IN 46241
Phone: 317.248.9332

World-Wide Technical Support
Phone: 248.888.2000
http://www.dgtech.com/support/support_guide.html
techsupp@dgtech.com