EQL Driver: Serial IP Load Balancing HOWTO

Simon "Guru Aleph-Null" Janes, simon@ncm.com

v1.1, February 27, 1995

This is the manual for the EQL device driver. EQL is a software device that lets you load-balance IP serial links (SLIP or uncompressed PPP) to increase your bandwidth. It will not reduce your latency (i.e. ping times) except in the case where you already have lots of traffic on your link, in which it will help them out. This driver has been tested with the 1.1.75 kernel, and is known to have patched cleanly with 1.1.86. Some testing with 1.1.92 has been done with the v1.1 patch which was only created to patch cleanly in the very latest kernel source trees. (Yes, it worked fine.)

1. Introduction

Which is worse? A huge fee for a 56K leased line or two phone lines? It's probably the former. If you find yourself craving more bandwidth, and have a ISP that is flexible, it is now possible to bind modems together to work as one point-to-point link to increase your bandwidth. All without having to have a special black box on either side.

The eql driver has only been tested with the Livingston PortMaster-2e terminal server. I do not know if other terminal servers support load- balancing, but I do know that the PortMaster does it, and does it almost as well as the eql driver seems to do it (-- Unfortunately, in my testing so far, the Livingston PortMaster 2e's load-balancing is a good 1 to 2 KB/s slower than the test machine working with a 28.8 Kbps and 14.4 Kbps connection. However, I am not sure that it really is the PortMaster, or if it's Linux's TCP drivers. I'm told that Linux's TCP implementation is pretty fast though.--)

I suggest to ISPs out there that it would probably be fair to charge a load-balancing client 75% of the cost of the second line and 50% of the cost of the third line etc...

Hey, we can all dream you know...

2. Kernel Configuration

Here I describe the general steps of getting a kernel up and working with the eql driver. From patching, building, to installing.

2.1. Patching The Kernel

If you do not have or cannot get a copy of the kernel with the eql driver folded into it, get your copy of the driver from ftp://slaughter.ncm.com/pub/Linux/LOAD_BALANCING/eql-1.1.tar.gz. Unpack this archive someplace obvious like /usr/local/src/. It will create the following files:

```
-rw-r--r-- guru/ncm      198 Jan 19 18:53 1995 eql-1.1/NO-WARRANTY
-rw-r--r-- guru/ncm      30620 Feb 27 21:40 1995 eql-1.1/eql-1.1.patch
-rwxr-xr-x guru/ncm      16111 Jan 12 22:29 1995 eql-1.1/eql_enslave
-rw-r--r-- guru/ncm      2195 Jan 10 21:48 1995 eql-1.1/eql_enslave.c
```

Unpack a recent kernel (something after 1.1.92) someplace convenient like say /usr/src/linux-1.1.92.eql. Use symbolic links to point /usr/src/linux to this development directory.

Apply the patch by running the commands:

```
cd /usr/src
patch </usr/local/src/eql-1.1/eql-1.1.patch
```

2.2. Building The Kernel

After patching the kernel, run make config and configure the kernel for your hardware.

After configuration, make and install according to your habit.

3. Network Configuration

So far, I have only used the eql device with the DSLIP SLIP connection manager by Matt Dillon (-- "The man who sold his soul to code so much so quickly."--) . How you configure it for other "connection" managers is up to you. Most other connection managers that I've seen don't do a very good job when it comes to handling more than one connection.

3.1. /etc/rc.d/rc.inet1

In rc.inet1, ifconfig the eql device to the IP address you usually use for your machine, and the MTU you prefer for your SLIP lines. One could argue that MTU should be roughly half the usual size for two modems, one-third for three, one-fourth for four, etc... But going too far below 296 is probably overkill. Here is an example ifconfig command that sets up the eql device:

```
ifconfig eql 198.67.33.239 mtu 1006
```

Once the eql device is up and running, add a static default route to it in the routing table using the cool new route syntax that makes life so much easier:

```
route add default eql
```

3.2. Enslaving Devices By Hand

Enslaving devices by hand requires two utility programs: eql_enslave and eql_emancipate (-- eql_emancipate hasn't been written because when an enslaved device "dies", it is automatically taken out of the queue. I haven't found a good reason to write it yet... other than for completeness, but that isn't a good motivator is it?--)

The syntax for enslaving a device is "eql_enslave <master-name> <slave-name> <estimated-bps>". Here are some example enslavings:

```
eql_enslave eql sl0 28800
eql_enslave eql ppp0 14400
eql_enslave eql sl1 57600
```

When you want to free a device from its life of slavery, you can either down the device with ifconfig (eql will automatically bury the dead slave and remove it from its queue) or use eql_emancipate to free it. (-- Or just ifconfig it down, and the eql driver will take it out for you.--):

```
eql_emancipate eql sl0
eql_emancipate eql ppp0
eql_emancipate eql sl1
```

3.3. DSLIP Configuration for the eql Device

The general idea is to bring up and keep up as many SLIP connections as you need, automatically.

3.3.1. /etc/slip/runslip.conf

Here is an example runslip.conf:

```
name          sl-line-1
enabled
baud          38400
mtu           576
ducmd         -e /etc/slip/dialout/cua2-288.xp -t 9
command        eql_enslave eql $interface 28800
address        198.67.33.239
line          /dev/cua2

name          sl-line-2
enabled
baud          38400
mtu           576
ducmd         -e /etc/slip/dialout/cua3-288.xp -t 9
command        eql_enslave eql $interface 28800
address        198.67.33.239
line          /dev/cua3
```

3.4. Using PPP and the eql Device

I have not yet done any load-balancing testing for PPP devices, mainly because I don't have a PPP-connection manager like SLIP has with DSLIP. I did find a good tip from LinuxNET:Billy for PPP performance: make sure you have asyncmap set to something so that control characters are not escaped.

I tried to fix up a PPP script/system for redialing lost PPP connections for use with the eql driver the weekend of Feb 25-26 '95 (Hereafter known as the 8-hour PPP Hate Festival). Perhaps later this year.

4. About the Slave Scheduler Algorithm

The slave scheduler probably could be replaced with a dozen other things and push traffic much faster. The formula in the current set up of the driver was tuned to handle slaves with wildly different bits-per-second "priorities".

All testing I have done was with two 28.8 V.FC modems, one connecting at 28800 bps or slower, and the other connecting at 14400 bps all the time.

One version of the scheduler was able to push 5.3 K/s through the 28800 and 14400 connections, but when the priorities on the links were very wide apart (57600 vs. 14400) the "faster" modem received all traffic and the "slower" modem starved.

5. Testers' Reports

Some people have experimented with the eql device with newer kernels (than 1.1.75). I have since updated the driver to patch cleanly in newer kernels because of the removal of the old "slave- balancing" driver config option.

- icee from LinuxNET patched 1.1.86 without any rejects and was able to boot the kernel and enslave a couple of ISDN PPP links.

5.1. Randolph Bentson's Test Report

```
From bentson@grieg.seaslug.org Wed Feb  8 19:08:09 1995
Date: Tue, 7 Feb 95 22:57 PST
From: Randolph Bentson <bentson@grieg.seaslug.org>
To: guru@ncm.com
Subject: EQL driver tests


I have been checking out your eql driver.  (Nice work, that!)
Although you may already done this performance testing, here
are some data I've discovered.

Randolph Bentson
bentson@grieg.seaslug.org
```

A pseudo-device driver, EQL, written by Simon Janes, can be used to bundle multiple SLIP connections into what appears to be a single connection. This allows one to improve dial-up network connectivity gradually, without having to buy expensive DSU/CSU hardware and services.

I have done some testing of this software, with two goals in mind: first, to ensure it actually works as described and second, as a method of exercising my device driver.

The following performance measurements were derived from a set of SLIP connections run between two Linux systems (1.1.84) using a 486DX2/66 with a Cyclom-8Ys and a 486SLC/40 with a Cyclom-16Y. (Ports 0,1,2,3 were used. A later configuration will distribute port selection across the different Cirrus chips on the boards.) Once a link was established, I timed a binary ftp transfer of 289284 bytes of data. If there were no overhead (packet headers, inter-character and inter-packet delays, etc.) the transfers would take the following times:

```
bits/sec  seconds
345600    8.3
234600    12.3
172800    16.7
153600    18.8
76800     37.6
57600     50.2
38400     75.3
28800     100.4
19200     150.6
9600      301.3
```

A single line running at the lower speeds and with large packets comes to within 2% of this. Performance is limited for the higher speeds (as predicted by the Cirrus databook) to an aggregate of about 160 kbits/sec. The next round of testing will distribute the load across two or more Cirrus chips.

The good news is that one gets nearly the full advantage of the second, third, and fourth line's bandwidth. (The bad news is that the connection establishment seemed fragile for the higher speeds. Once established, the connection seemed robust enough.)

#lines	speed kbit/sec	mtu	seconds duration	theory speed	actual speed	%of max
3	115200	900	_	345600		
3	115200	400	18.1	345600	159825	46
2	115200	900	_	230400		
2	115200	600	18.1	230400	159825	69
2	115200	400	19.3	230400	149888	65
4	57600	900	_	234600		
4	57600	600	_	234600		
4	57600	400	_	234600		
3	57600	600	20.9	172800	138413	80
3	57600	900	21.2	172800	136455	78

#lines	speed kbit/sec	mtu	seconds duration	theory speed	actual speed	%of max
3	115200	600	21.7	345600	133311	38
3	57600	400	22.5	172800	128571	74
4	38400	900	25.2	153600	114795	74
4	38400	600	26.4	153600	109577	71
4	38400	400	27.3	153600	105965	68
2	57600	900	29.1	115200	99410.3	86
1	115200	900	30.7	115200	94229.3	81
2	57600	600	30.2	115200	95789.4	83
3	38400	900	30.3	115200	95473.3	82
3	38400	600	31.2	115200	92719.2	80
1	115200	600	31.3	115200	92423	80
2	57600	400	32.3	115200	89561.6	77
1	115200	400	32.8	115200	88196.3	76
3	38400	400	33.5	115200	86353.4	74
2	38400	900	43.7	76800	66197.7	86
2	38400	600	44	76800	65746.4	85
2	38400	400	47.2	76800	61289	79
4	19200	900	50.8	76800	56945.7	74
4	19200	400	53.2	76800	54376.7	70
4	19200	600	53.7	76800	53870.4	70
1	57600	900	54.6	57600	52982.4	91
1	57600	600	56.2	57600	51474	89
3	19200	900	60.5	57600	47815.5	83
1	57600	400	60.2	57600	48053.8	83
3	19200	600	62	57600	46658.7	81
3	19200	400	64.7	57600	44711.6	77
1	38400	900	79.4	38400	36433.8	94
1	38400	600	82.4	38400	35107.3	91
2	19200	900	84.4	38400	34275.4	89
1	38400	400	86.8	38400	33327.6	86
2	19200	600	87.6	38400	33023.3	85
2	19200	400	91.2	38400	31719.7	82
4	9600	900	94.7	38400	30547.4	79
4	9600	400	106	38400	27290.9	71
4	9600	600	110	38400	26298.5	68
3	9600	900	118	28800	24515.6	85
3	9600	600	120	28800	24107	83
3	9600	400	131	28800	22082.7	76
1	19200	900	155	19200	18663.5	97
1	19200	600	161	19200	17968	93
1	19200	400	170	19200	17016.7	88
2	9600	600	176	19200	16436.6	85
2	9600	900	180	19200	16071.3	83
2	9600	400	181	19200	15982.5	83
1	9600	900	305	9600	9484.72	98
1	9600	600	314	9600	9212.87	95
1	9600	400	332	9600	8713.37	90

5.2. Anthony Healy's Report

```
Date: Mon, 13 Feb 1995 16:17:29 +1100 (EST)
From: Antony Healey <ahealey@st.nepean.uws.edu.au>
To: Simon Janes <guru@ncm.com>
Subject: Re: Load Balancing

Hi Simon,
      I've installed your patch and it works great. I have trialed
      it over twin SL/IP lines, just over null modems, but I was
      able to data at over 48Kb/s [ISDN link -Simon]. I managed a
      transfer of up to 7.5 Kbyte/s on one go, but averaged around
      6.4 Kbyte/s, which I think is pretty cool.  :)
```