# Connection load-balancing of PPP-Slirp serial links over SSH using EQL

Dimitar Ivanov <mailto:dimitar.ivanov@mirendom.org>

v0.9.2, 2007-08-29

This document describes how to load-balance multiple PPP-Slirp links over SSH by using the kernel device driver EQL, and in such a way to increase the connection bandwidth.

## Contents

**1 Introduction** **1**

1.1 Where and when is it useful . . . . . 1

1.2 Copyright . . . . . 2

1.3 Disclaimer . . . . . 2

1.4 Acknowledgements . . . . . 2

**2 Understanding the concept** **2**

**3 Software you need (OpenSSH + PPPD + Slirp + EQL)** **3**

**4 Implementation and configuration** **3**

**5 Performance and conclusions** **7**

**6 Sources and further readings** **8**

# 1 Introduction

## 1.1 Where and when is it useful

Slirp is a TCP/IP emulator which provides an ordinary user account with (C)SLIP/PPP capabilities. Although the original software is no more supported, it still has a wide scope of usage, as far as the evolved code is incorporated and further developed as a network adapter in virtual machines like QEMU and coLinux.

Beside the primary aim of Slirp to endow a dial-up user with an Internet connectivity, it can be applied for routing/tunneling of IP-traffic - for example between hosts in two different LANs with access possible only after authentication on a firewall. If a transparent communication is required in this case, Slirp and PPPD in combination with OpenVPN can suit your need for convenient solution, where eventually the OpenVPN server's host operates as a transparent gateway (router) between the two networks. In the course of working with this scenario, however, a severely limited upload bandwidth of the pppd<->slirp link will be detected, and the slow component in the system can be identified as Slirp.

Fortunately, Slirp supports connection load-balancing, and by means of the linux kernel device EQL, a load-balancer for serial network interfaces, multiple PPP links can be bound together in order to increase the bandwidth significantly. The present document explains how to implement efficiently this load-balancing technique. Finally, for configurations where only the download bandwidth is important, the conclusion is made that a single link gives the best performance.

### 1.2 Copyright

Copyright (c) 2007 Dimitar Ivanov

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at the GNU project web site [http://www.gnu.org/copyleft/fdl.html]

### 1.3 Disclaimer

No liability for the contents of this document can be accepted by the author under any circumstance. Use the concepts, examples and information at your own risk. Please feel free to e-mail me suggestions, corrections or general comments about the document so I can improve it.

All copyrights are held by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

### 1.4 Acknowledgements

I would like to express my gratitude to Alex Hubmann for his, in various ways, friendly support to my effort of writing the present document.

## 2 Understanding the concept

Suppose you can not connect from host_A to host_B directly but only through:

- Authentication on a firewall
- Several hops over various accounts
- Some other tricky way ...

And for a good reason, of course, you want to route in transparent way some or all of the IP-traffic from A to B. Further, you are root on host_A, but your account on host_B is one of an ordinary user. In that situation, Slirp, which is a single binary executable that can be installed on host_B, will help you to establish a PPP connection between the hosts, and hence to solve your problem. Unfortunately, one will discover that the upload speed of the connection is limited to 30-45 KB/s whereas the network allows much higher performance. The good news is that if host_A is a GNU/Linux box there is something you can do to increase the upload bandwidth significantly. With help of the so-called "EQL device" or "equalizer", implemented as a kernel module, you can bind several pppd<->slirp links (over ssh) to a single virtual interface performing a traffic load-balancing. Following diagram shows the described application:

```
---                                                         ---
 |                                                           |
 |      Linux server                     Remote *NIX server  |
 |         Host_A                              Host_B        |
       |     +-----------+                        +-----------+    |
       |     |           |        [ppp0]          |           |    |
       |     |           |---/ ssh via *host* /---|           |    |
LAN A  |     | Load-     |        [ppp1]          | Load-     |    | LAN B
       |-----| balancing |---/ ssh via *host* /---| balancing |----|
       |     | EQL over  |        [ppp2]          | Slirp     |    |
       |     | PPP links |---/ ssh via *host* /---|           |    |
       |     |           |                        |           |    |
       |     |           | ====== Routing ======> |           |    |
       |     +-----------+                        +-----------+    |
       |                                                           |
      ---                                                         ---
```

# 3 Software you need (OpenSSH + PPPD + Slirp + EQL)

An OpenSSH client is used in conjunction with the "pty" option of pppd to establish a secure PPP link between the nodes.

The Point-to-Point Protocol Daemon (pppd) is needed to initiate a PPP link from host_A to Slirp on host_B. Thus the possibility is open of routing TCP/IP-traffic via the set up PPP device.

Slirp is a TCP/IP emulator in user space which, when started from shell, emulates (C)SLIP/PPP on the terminal. It is installed on host_B.

The EQL device driver (kernel module "eql.ko") can bundle a few serial links into a virtual network interface "eql" which load-balances the IP-flow over its enslaved devices. In addition to the driver, enslaving utilities are needed. For example, in Debian the "eql"-package provides them.

Eventually, the "iproute2" tools are used to adjust the routing on host_A.

All the required software, except Slirp, must be installed on the Linux host_A. On the peer *NIX host_B, where we assume that a C-compiler is available, one has to install the Slirp binary compiled with the next options defined (in "config.h", respectively "config.h.in") for the purpose of load-balancing:

```
#define FULL_BOLT yes
#define MAX_INTERFACES 4
#define MAX_PPP_INTERFACES 4
```

# 4 Implementation and configuration

Before starting up the system, one must be able to login by ssh from the root account on host_A to the user's account on host_B without a password. This is achieved by creating ssh-keys on A (and any other host on the way between) and adding the public key to the file ".ssh/authorized_keys" on the next peer.

Also configuration files for Slirp have to be prepared and put in the home directory on host_B. The files should include the options:

```
.slirprc: "socket"
.slirprc-0, .slirprc-1, .slirprc-2: "ppp"
```

Now, we are ready for launch. On host_A, several !uncompressed! PPP links to host_B are brought up by pppd over ssh via a fictious account for authentication "fireman@firewall". A link is started by the next command, where the "-l" option of slirp specifies the link number:

```
/usr/sbin/pppd \
         local 10.0.2.15:10.0.2.2 mtu 1500 passive noauth \
         nodeflate novj novjccomp updetach nodefaultroute idle 0 pty \
         "ssh -q -t fireman@firewall ssh -q -t user@host_B slirp -l 0"
```

After establishing three links (for the reason see below), they are tied up to work together by the equalizer, which enslaves the corresponding PPP interfaces:

```
/sbin/ifconfig eql 172.16.1.1 netmask 255.255.255.0 mtu 1500
/usr/sbin/eql_enslave eql ppp0 1
/usr/sbin/eql_enslave eql ppp1 1
/usr/sbin/eql_enslave eql ppp2 1
```

Now, we are be able to route IP-traffic via the "eql" network interface, to which the IP address 172.16.1.1 is assigned. For example, if network destinations 192.168.1.0/24 should be reached through the remote account on host_B, the routing table can be adjusted like:

```
ip route add 192.168.1.0/24 dev eql
```

Further, the load-balancing mechanism causes packets received out of order, but one could remedy the problem by tuning the reordering of tcp packets - here the kernel's ipv4 variable "net.ipv4.tcp_reordering" will be changed to its maximum value:

```
echo 127 > /proc/sys/net/ipv4/tcp_reordering
```

It should be remarked that the number of PPP links is not chosen at random. In our case, after some experimentation with that number, the MTU of the interfaces, and also their qlen, it was found that the best performance is provided when using tree links with default settings.

Here's a script that automates the application control. It assumes that the remote peer is also GNU/Linux, or equivalently, that the "killall" command exists on the remote host:

```
#!/bin/sh -e
#
# Script for installing an EQL load-balanced PPPD-Slirp connection
#
VERSION=1.0
PATH=/sbin:/usr/sbin:$PATH
MYNAME=‘basename $0‘

##############################################################################
#
#  General configuration
#

# For single link connection uncomment next line
#FastDownload_SlowUpload=yes
LINKS="0 1 2"
[ $FastDownload_SlowUpload ] && LINKS=0

   # Slirp is expected to be installed on remote account in $HOME/bin
REMOTE_SLIRP=bin/slirp
```

```
REMOTE_ACCOUNT=user@remotehost
   # Go first through authentication on firewall
VIA="ssh -q -t fireman@firewall"
   # PPP configuration
MTU=1500
PPPD=pppd
PPPD_OPTS="local 10.0.2.15:10.0.2.2 mtu $MTU passive noauth nodeflate \
           novj novjccomp updetach nodefaultroute idle 0"
EQL_IP=172.16.1.1
REMOTE_LANS="192.168.1.0/24"

   # If existing, load $HOME/.<script name>rc
   # where you can define your own configuration
CFG_FILE=$HOME/.${MYNAME}rc
test -f $CFG_FILE && . $CFG_FILE

CHECK_STATUS=no

###############################################################################
#
#  Functions
#
Status_EQL() {

   set l 0
   set -- $LINKS

      # Count the ppp links
   while [ -n "$1" ]; do let l=l+1; shift ; done
      # Verify this number against the setup value
   [ `ip link list |grep "[pp]pp.*SLAVE" |wc -l` -eq $l ] \
      && { echo "EQL up" ; return 1 ; } \
      || [ $ACT = status ] && { echo "EQL down" ; return 1 ; }

return 0
}

###############################################################################
#
#  MAIN
#
[ $# -eq 0 ] && set -- -h
while [ $# -gt 0 ]; do
   ACT=$1
   case $ACT in
       -h) cat << EOH

Usage: $MYNAME [OPTIONS] start|stop|restart|status

Options:
   -h        Help
   -v        Print script version and exit

Actions:
   start     Start EQL if not running, or if the number of links less then
             defined (useful also for monitoring)
   stop      Stop EQL
   status    Status of EQL
   restart   Restart EQL

EOH
           exit
           ;;
       -v) echo $VERSION
           exit
           ;;
    start) CHECK_STATUS=yes
           ;;
     stop) CHECK_STATUS=no
           ;;
  restart) CHECK_STATUS=no
           ;;
   status) CHECK_STATUS=yes
           ;;
        *) set -- -h -h
           ;;
   esac
   shift
done


[ $CHECK_STATUS = yes ] && { Status_EQL || exit ; }

   # Clean up current links (remote and local)
ssh -q $REMOTE_ACCOUNT killall -q $REMOTE_SLIRP || true
pkill -f  "pppd.*slirp" || true

   # Unload the EQL module, it is then automatically loaded by ifconfig
rmmod eql || true

[ $ACT = stop ] && echo "EQL stopped" && exit

echo "EQL ${ACT}ing ..."
echo ""

for i in $LINKS
do
    $PPPD $PPPD_OPTS pty "$VIA ssh -q -t $REMOTE_ACCOUNT $REMOTE_SLIRP -l $i"
done
```

```
sleep 2
ifconfig eql $EQL_IP netmask 255.255.255.0 mtu $MTU
sleep 2

   # Enslave ppp's
for i in $LINKS
do
    eql_enslave eql ppp$i 1
done

echo 127 > /proc/sys/net/ipv4/tcp_reordering

   # Slirp's management IPs
ip route add 10.0.2.0/28 via 10.0.2.2

   # Add new routes
for i in $REMOTE_LANS
do
    ip route add $i via $EQL_IP
done

echo ""
echo "... EQL ready"

exit 0
```

---

In a special case you can also connect two hosts directly without going through a third host (like a firewall). Here, it means to set the "VIA" variable in the "General configuration" paragraph to an empty string or to comment it out.

# 5 Performance and conclusions

With the applied technique of serial line load-balancing in various configurations in standard 100 Mbit/s (12.5 MB/s) LANs, the upload data throughput was increased 15-35 times (0.45-1.1 MB/s) compared to a single-link connection; although it varied for download and upload. On the other hand, in the "real world" with WAN connections in Internet, most probably the rate will be of a lower extent - in one particular case tested, the bandwidth reached 5-7 times higher values then its original rate of 30KB/s.

The performance measurements show that the upload rate increases strongly in all cases. The disadvantage, however, is that the download rate decreases by 20-30% compared to one-link configuration! Consequently, if you connect two hosts by pppd<->slirp, and only a high download speed is required but upload is irrelevant, then don't load-balance the connection - a single link is the best solution; still compile Slirp with defined FULL_BOLD option. Regarding the script above, the line "FastDownload_SlowUpload=yes" has to be uncommented then.

Eventually, when both directions play role, we have to accept that the load-balancing will boost the upload, yet impede to some extend the download. Also upload could be faster than download.

During a period of about one month the system has been observed to break few times such that the equalizer ceased to load-balance the connection as normal and data flow over one of the enslaved links but not the others. The trivial work-around is just to monitor the connection and reestablish it in case of failure.

Finally, in a more sophisticated variant of the method, the load-balancing by defining different network paths for the enslaved links and going over different physical interfaces, can provide you with a simple failover solution (at least on the initiating side of the connection).

# 6 Sources and further readings

The latest version of this HOWTO can be found at http://www.mirendom.org/docs.html

You could find helpful following documents and links concerning Slirp, PPP and EQL:

- NET3-4-HOWTO
- PPP-HOWTO
- SLIP-PPP-Emulator
- (url url="http://slirp.sourceforge.net/">
- Using EQL With Linux - Primitive Load Balancing
  <`http://www.indyramp.com/eql/eql.html`>
- (HOWTO) Tunneling the hard way: using slirp, pppd and socat
  <`http://forums.gentoo.org/viewtopic-t-400679.html`>