Reprinted from the

Proceedings of the Linux Symposium

July 23th–26th, 2003
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Alan Cox, *Red Hat, Inc.*
Andi Kleen, *SuSE, GmbH*
Matthew Wilcox, *Hewlett-Packard*
Gerrit Huizenga, *IBM*
Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Martin K. Petersen, *Wild Open Source, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Building Enterprise Grade VPNs

A Practical Approach

Ken S. Bantoft
freeswan.ca / MDS Proteomics
ken@freeswan.ca, http://www.freeswan.ca

Abstract

As Linux scales its way into the enterprise, things like high availabilty and redundancy become more and more important.

As is the case with many Open Source applications, you can integrate several applications together to build in as much redundancy and failover as you need for your network.

By combining several Open Source applications, including Linux [1], FreeS/WAN [2], GNU/Zebra [3] and Heartbeat [4] we are able to build a very reliable, robust VPN solution.

1 Building an Enterprise VPN

FreeS/WAN has been used for several years by many Linux system administrators to build Virtual Private Networks (VPNs) between sites, end-users, and business partners. It is very configurable, inter-operates well with other IPSec implementations and is generally quite stable. It has few limits, however two of the more common limits sysadmins run into are:

1. IPSec doesn't tunnel all IP traffic—it does not handle Multicast or Broadcast traffic. This means if we wanted to do Multicasting over our VPN, or run OSPF from Zebra on ipsec0 for dynamic routing, we can't. While it's possible to run OSPF in NMBA mode, we ran into problems with this as well.

2. You need one tunnel per network combination pair—thus if you have 4 IP subnets behind Secure Gateway #1, and 2 behind Secure Gateway #2, you will need to configure 8 separate tunnels (unless you can aggregate your IP network space to /23s, /16s or other CIDR compatible blocks).

Point 1 isn't too much of a limit, since many networks don't need multicast and broadcast traffic to be routed between sites. In large networks, point 2 quickly becomes an adminitrative nightmare to deal with.

This can quickly get out of hand for large networks, especially when more than two sites are involved. The ideal solution is to setup a single tunnel between each site, and then route all traffic from site 1 destined for site 2 over the VPN, and hope the other side accept it. The problem here is that IPSec policies (which FreeS/WAN enforces) prevents this—if there is no explcit tunnel defined for the Source + Destination pair, the packet is dropped.

The solution here is to use GRE—Generic Routing Encapsultion. This has been part of the Linux kernel for quite some time, and allows us to solve both problems identified above. By setting up a GRE tunnel over one of our IPSec tunnels, we can then route any-

thing we want over the GRE tunnel, including Multicast traffic.

Once we are using GRE, we need some way to dynamically inform the other side of the tunnel which networks we know about, and how to get to them. GNU/Zebra can provide this functionality, using either OSPF or BGPv4.

2 Challenges & Solutions

Integration of all of the applications used presented several challenges, since none of them were designed to work together. Some had to be extended, and others had to be scripted around in order for them to notify each other of various different sorts of failures.

Luckily, with source in hand, this was much easier than expected.

2.1 Zebra

Zebra was the easiest to integrate, as no direct code changes were required. There were initially some problems with using OSPF on aliased interfaces (eg: eth0:0) but those were solved by an upgrade to the latest version (0.91a or 0.93 are known to work).

2.2 Heartbeat

Heartbeat needed some modifications so it was aware of the status of the physical interfaces, and then just needed a basic configuration and a lot of scripting.

One of the most significant differences between commercial grade routers and Linux is that if an interface is physically unplugged, or the switch/hub on the other side goes down, a commercial router drops the interface, and all routes that travel over it are removed. Linux desperatly needs this capability, but until recently many network cards were unable to report the link status.

Donald Becker[5] wrote a handy toolset for this—mii-tool/mii-diag. During my tenure at IBM Canada, one of the developers I worked with took this and turned it into a patch against Heartbeat. If Heartbeat detects a physical problem with the network card/cable/switch, Heartbeat initates a failover event. This code supported the Intel 10/100 (eepro.o|e100.o) as well as the Intel Gigabit Ethernet adapters (e1000.o).

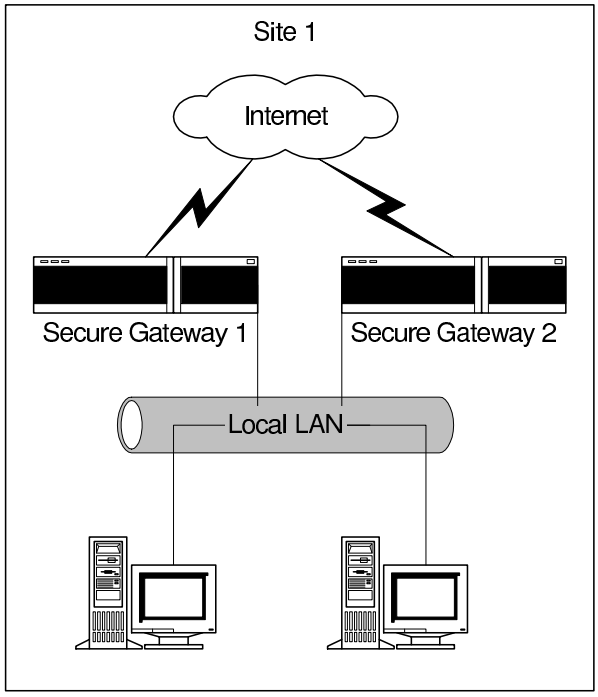
The bulk of the time went into the rewriting of many scripts to properly bring interfaces up and down, to restart Zebra's bgpd correctly, and to cleanly restart FreeS/WAN.

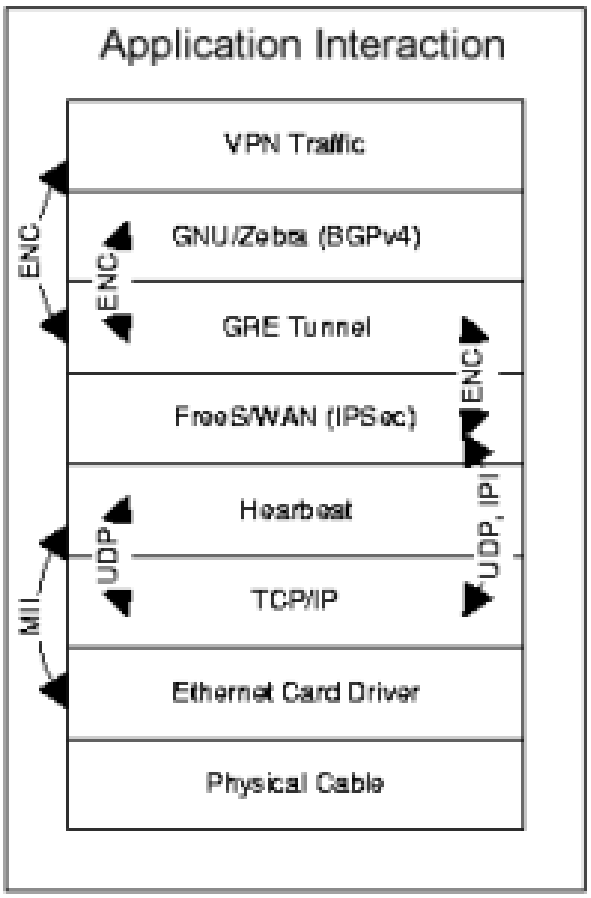
2.3 FreeS/WAN

FreeS/WAN required no code changes, only configuration and some scripting to keep the configuration syncronised between each set of Secure Gateways. We used SSH for this.

3 Gluing it all Together

The complicated part is making all of these applications and protocols work together seamlessly. Our basic network layout is below:





We have two Secure Gateways, each with a connection to the internet. Ideally, each would have its own link to the Internet, preferably redundant, but you could share the link if needed. Each gateway also has a connection to the local lan, which Heartbeat sends keep-alives over. If possible, use a Null-modem cable between each pair of gateways for out of band keep alives. If this isn't possible, Heartbeat also supports UDP keep-alives over any network interface.

Ensure each gateway has all of the applications installed, and the GRE and FreeS/WAN configurations should be syncronised. Heartbeat and GNU/Zebra configurations differ slightly, so they should not be shared.

A diagram showing the key interactions between the applications:

Heartbeat is the control center in this setup, as it monitors each node in the group for failure, and checks its own Ethernet devices via MII calls. Heartbeat also starts and stops various scripts (from /etc/rc.d/init.d) which bring up FreeS/WAN, the GRE tunnels, and GNU/Zebra.

3.1 Dealing with Startup Scripts

FreeS/WAN and GNU/Zebra both provide scripts suitable for use in /etc/rc.d/init.d, so we used those. We also needed to add a GRE tunnel, so we wrote our own startup scripts for that. A quick example:

```
#!/bin/sh
# chkconfig: 2345 50 64
# description: Set up a GRE tunnel from
```

```
# here to somewhere else

case "$1" in
  start)
    ip tunnel add MYTunnel mode gre \
    remote 216.1.1.1 local 116.1.1.1 ttl 255
    ip link set MYTunnel up
    ip addr add 172.16.0.1 dev MYTunnel
    ip route add 172.16.0.2/32 dev MYTunnel
    ;;
  stop)
    ip route del 172.16.0.2/32 dev MYTunnel
    ip addr del 172.16.0.1 dev MYTunnel
    ip link set MYTunnel down
    ;;
  restart)
    $0 stop
    $0 start
    ;;
  *)
    echo "Usage: $0 {start|stop|restart}" >&2
    exit 2
esac

exit 0

}
```

Our script supports the traditional arguments start, stop, and restart, and will bring the GRE tunnel up and down when called from Heartbeat.

3.2 Heartbeat Configuration

Full details on configuring Heartbeat are available from the package itself [7], so I will cover only the /etc/ha.d/haresources config file here. From Heartbeat, we need to control the IP address takeover, and the services (/etc/rc.d/init.d scripts) we start and stop when a node fails. This can be done with a simple, single line entry in /etc/ha.d/haresources:

```
cluster1 116.1.1.1/28 192.168.0.1/24 \
ipsec gre zebra bgpd
}
```

The above line tells Heartbeat to do IP address takeover on 159.18.124.254 (our External IP address) 192.168.0.1 (our Internal IP address). It also lists the scripts (in order) to run when a takeover happens. It passes each of these scripts a parameter—either "start" or "stop" determined by what is occurring—taking over the IP, or releasing it. (I.e., `/etc/rc.d/init.d/ipsec start`.)

3.3 FreeS/WAN Configuration

FreeS/WAN configuration was straightforward. PSK (pre shared secrets) use is not recommended, as recent bugtraq postings have shown some potential security flaws. We recommend RSASig's, however X.509 Digital certificates can also be used. The following example uses PSK authentication for one remote site:

```
config setup
    interfaces="ipsec0=eth0:0"
    klipsdebug=none
    plutodebug=none
    plutoload=%search
    plutostart=%search
    uniqueids=yes

conn %default
    keyingtries=0

conn site1tosite2
authby=rsasig
    left=116.1.1.1
    leftnexthop=116.1.1.30
    leftrsasigkey=0xA0S8PIPI...
    leftid=@site1.company.com
    right=216.1.1.1
    rightnexthop=216.1.1.30
    rightrsasigkey=0xA0QKJ986...
    rightid=@site2.company.com
    auto=start
}
```

The critical line of the config is `interfaces="ipsec0=eth0:0"`, as by default FreeS/WAN won't bind to an aliased interface. Since Heartbeat brings up the service IP addresses on aliases, we need to bind our ipsec interface to the alias.

3.4 GNU/Zebra Configuration

From GNU/Zebra, we use the BGPv4 daemon to handle our dynamic routing. This gives us much more control over which routes we share than OSPF would, as well as makes configuration simple.

Sample bgpd.conf file:

```
!
hostname torcofw1
password a_secure_password
enable password a_more_secure_password
log file bgpd.log
log stdout

router bgp 65432
 bgp router-id 172.16.0.1
 network 172.16.0.0/30
 redistribute kernel
 redistribute connected
 redistribute static
 neighbor 172.16.0.2 remote-as 65432
 neighbor 172.16.0.2 next-hop-self
!
}
```

We use a reserved AS number (65432) in case we ever need to do BGP with peers from another company, or the Internet. All of the network and neighbour statements refer to our GRE tunnel IP addressing, as we wish to communicate with our BGP peer over the GRE tunnel—not the IPSec tunnel. neighbour 172.16.0.2 next-hop-self is critical—we need our BGP peer to send any traffic destined for our local networks through us directly, since we have an established tunnel.

4 Conclusions

It took a few weeks to get this setup stable, during which we changed from OSPF to BGPv4, which cleared up several problems we encountered with neighbours failing to exchange routes consistantly. The current design has been running in production at 4 sites for over 2 years now, and we have had several successful failovers (several faulty network cards, a bad switch port, and the more common system administrator error).

Connections that do not pass though netfilter connection tracking (i.e., NAT/MASQ) are usually unaffected—with a keepalive time of 2 seconds, and a deadtime of 10 seconds, dead peer detection is fairly quick. This can be optimized down to about 5 seconds if needed. Changing over the IP addresses, starting FreeS/WAN, GRE tunnels and GNU/Zebra takes less than 10 seconds on modern hardware, so our total time between failover is less than 20 seconds.

5 Future Improvements

There are more improvements to be made that could bring detection and failover down into the 1–3 second range.

Heartbeat seems currently limited to 1-second keepalives—this could be brought down to 1/4 second over the serial interface, meaning a deadtime of 1 second would be reasonable (3 missed polls).

FreeS/WAN has a routing limitation whereby you can't have two tunnels for the same source + destination pair going to two different remote gateways. Hopefully, this limitation will not be present in either kernel 2.6's IPSec implementation, or future versions of FreeS/WAN that implement the MAST [8] device.

Connections that do utilize the netfilter connection tracking are currently cut off, since the secondary firewall is not aware of the current state of the conntrack table on the primary firewall. There was some discussion at the OLS 2002 Netfilter BOF, and on the Netfilter Failover list [9] on how to handle syncronization of the conntrack table, however no code

has emerged.

6 Acknowledgments

I would like to acknowledge the FreeS/WAN team, and extra thanks to Michael Richardson and JuanJo Ciarlante who helped me setup a UML environment setup so I could demonstrate much of this. I'd also to acknowledge IBM Canada, who employed me during the initial development of this solution, and my current employer, MDS Proteomics who allows me to continue to refine it. Also, thanks to Astaro Corporation for funding some of my development of Super FreeS/WAN, and covering my hosting costs for freeswan.ca.

7 Availability

Software:

```
http://www.freeswan.ca
http://www.zebra.org
http://www.linux-ha.org
```

Documentation:

```
http://www.freeswan.ca/docs/HA
```

References

[1] Linux, `http://www.linux.org`

[2] The FreeS/WAN Project,
`http://www.freeswan.org`
`http://www.freeswan.ca`

[3] The GNU Zebra Project,
`http://www.zebra.org`

[4] The Linux-HA Project,
`http://www.linux-ha.org`

[5] Donald Becker, Scyld Computing Corporation, `http://www.scyld.com/diag#mii-diag`

[6] Ken S. Bantoft, *Building HA VPNS with FreeS/WAN*, `http://www.freeswan.ca/docs/HA/`

[7] Getting Started with Heartbeat,
`http://www.linux-ha.org/download/GettingStarted.html`

[8] John S. Denker, *Next-Generation IPsec Packet Handling*, `http://www.quintillion.com/moat/ipsec+routing/mast.html`

[9] Netfilter Failover Archives,
`http://lists.netfilter.org/pipermail/netfilter-failover/`