Ethernet Bridge + netfilter Howto

Table of Contents

Ethernet Bridge + netfilter Howto

Nils Radtke

v0.8, July 2005

Setting up an ethernet bridge gives us the chance to integrate a surveying and/or regulating instance transparently into an existing network. This setup requires no changes to the logical network topology. It is accomplished by plugging the ethernet bridge in the physical network topology between the network itself and the routing instance (that piece of hardware connected to the Internet).

This Howto is available in other formats. Preferably downloadable: documentation tarball. You may find this Howto as part of the Linux Documentation Project.

Looking for other languages? See the German version, then! And as of march 2003 there is a french version available with courtesy of François Romieu and Guillaume Lelarge: French version. Today, some day in july 2005, I stumbled across an italian version of this HOWTO! Unfortunately there's no translator's note in the document, courtesy is unbeknownst so far: Italian version.

2005−07−12:
: Added link to italian version of this HOWTO (see above). Plus a section about making the changes reboot−persistent, a note about 2.6 kernel not working as expected and the user experiences−section. Included the Final note−section. :)

2005−06−06:
: Unfortunately, the linux documentation project does not respond actively to update requests. This results in an outdated version of the on www.tldp.org site. Use this site instead, if you want to stick with the most actual version of the HOWTO. Blame the tldp people.

2004−08−03:
: Carsten (C DOT Lueth AT fielmann DOT com) has found a variant for MS Windows (TM) systems.

2004−06−13:
: Update on used kernel versions, warning about remote administering, netfilter debugging code. New translation available: French (link c. f. above). Discontinued german version.

2002−09−19:
: links about ebtables have been updated in the "Related Topics" Section. Added note about "false positive" br−nf debugging output.

2002−10−08:
: Added section Actual configuration and hints about routing in Setting up the routing, Ping it, Jim! , resp.

1. Introduction

2. Required software

3. Set Linux up to serve

4. Test your new bridged environment!

5. User experiences

6. Links

---

1. Introduction

Ethernet bridges connect two or more distinct ethernet segments transparently.
An ethernet bridge distributes ethernet frames coming in on one port to other ports associated to the bridge interface. This is accomplished with brain: Whenever the bridge knows on which port the MAC address to which the frame is to be delivered is located it forwards this frame only to this only port instead of polluting all ports together.

Ethernet interfaces can be added to an existing bridge interface and become then (logical) ports of the bridge interface.

Putting a netfilter structure on top of a bridge interface renders the bridge capable of servicing filtering mechanisms. This way, a transparent filtering instance can be created. It even needs no IP address assigned to work. Of course, you can assign an IP address to the bridge interface for maintenance purposes (certainly, with ssh only ;−).

The advantage of this system is evident. Transparency alleviates the network administrator of the pain of restructuring the network topology. And users may not notice the existence of the bridge but their connection beeing blocked. Also, users are not disturbed while working (think of a company where network connection loss pays alot).

The other common case is a client beeing connected to the global web via a leased router. As the providers seldomly grant administration privileges on their leasing hardware, the client cannot change the interconnecting configuration. But, of course, the client has a network running, and wants to spend at least as possible, he does not want to reconfigure his entire network. And he does not need to if he uses a bridging device.

2. Required software

This software setup is needed on the ethernet bridge computer. According to our Testing grounds.

2.1 Featured Linux kernel

Use of kernel 2.6 is not yet a good idea. Yes, it's astonishing. The why the bridging code breaks and where it does so has not yet come to my and others attention, I cannot recommend kernels of the 2.6 series. You have the clou? Assure yourself the credit, mail the solution to me (e−mail address at entry page). See also Kernel−Notes for additional information on this. So far, use kernel 2.4 series.
As of kernel version *2.4.18* there's already support for the Ethernet Bridge capability built−in. No patches needed so far. Regarding later kernel versions, it must be stated that *2.4.23* might be less recommendable, especially in conjunction with ebtables and netfilter−bridging. Later versions seem advisable.
The following paragraph is outdated now (2005−07−12) as all we need is present in kernel. You may skip this paragraph, it is only retained for legacy:
But if we intend to use netfilter capabilities, because we want to run iptables on our new Linux router/fw box, we still need to apply a patch. Any patches needed can be found and downloaded on the sourceforge Ethernet Bridge homepage.

```
root@bridge:~> cd /usr/src/
root@bridge:~> wget -c http://bridge.sourceforge.net/devel/bridge-nf/bridge-nf-0.0.7-against-2.4.18.diff
root@bridge:~> cd /usr/src/linux/
root@bridge:~> patch -p1 -i ../bridge-nf/bridge-nf-0.0.7-against-2.4.18.diff
```

Supposedly we want netfilter support on our bridge interface and we have already patched the vanillal kernel we may now activate some necessary kernel configuration items. On how to build a private kernel image see the CD−Net−Install−HOWTO, Toolbox. Oh, yeah, it's still in German only. Hm, I should fix this some time, but time lacks... Any volunteers? (deadly silence is cracking.. ;)

Nevertheless, we start by now: In

```
Code maturity level options
```

we activate

```
[*] Prompt for development and/or incomplete code/drivers
```

and in

```
Loadable module support
```

```
[*] Enable loadable module support
[*]   Set version information on all module symbols
[*]   Kernel module loader
```

Ok, so far so good. Now, we go to

```
Networking options
```

and mark

```
[*] Network packet filtering (replaces ipchains)
[ ]   Network packet filtering debugging
```

Note:

Previously, the above debugging option had been selected. For now, unless you want your `/var/log/`–partition being filled up in short–time distance, deactivate this option.
If this options is activated, messages similar to the following appear in counts of thousands in dmesg and `/var/log/{kern.log,debug,syslog,messages}`:

```
skb: pf=2 (unowned) dev=br0 len=52
PROTO=6 156.136.32.121:3709 192.168.101.2:112 L=52 S=0x00 I=35470 F=0x4000 T=51
nf_hook: hook 1 already set.
skb: pf=2 (unowned) dev=br0 len=52
PROTO=6 156.136.32.121:3709 192.168.101.2:112 L=52 S=0x00 I=35470 F=0x4000 T=51
nf_hook: hook 0 already set.
skb: pf=2 (unowned) dev=br0 len=52
PROTO=6 192.168.101.11:2828 192.168.101.2:202 L=52 S=0x10 I=63 F=0x4000 T=64
nf_hook: hook 1 already set.
skb: pf=2 (unowned) dev=br0 len=52
PROTO=6 192.168.101.11:2828 192.168.101.2:202 L=52 S=0x10 I=63 F=0x4000 T=64
nf_hook: hook 3 already set.
skb: pf=7 (owned) dev=eth1 len=1500
```

Furthermore, in

```
  IP: Netfilter Configuration  --->
```

we mark any item we need as module. Now the long awaited item: activate

```
<M> 802.1d Ethernet Bridging
```

as well as

```
[*]   netfilter (firewalling) support
```

Note:

The above entry is available only if we successfully patched our kernel!

Finally, we just need a successful

```
root@bridge:~> make dep clean bzImage modules modules_install
```

cycle and we're done. Don't forget to edit `/etc/lilo.conf` and do

```
root@bridge:~> lilo -t
root@bridge:~> lilo
root@bridge:~> reboot
```

, though.

Hint:

Perhaps we might mark our new kernel as the bridge kernel? We `vi` the toplevel Makefile in our kernel sources and edit the head line called `EXTRAVERSION =`. We may actually set it to, say *bridge*? ;–)
After the `modules_install` we find the fresh modules in `/lib/modules/2.4.18bridge`
For debian users (eventually use `export PATCH_THE_KERNEL=YES` before and −−added_patches your_patches with make−kpkg):

```
root@bridge:~> make-kpkg --revision=tf.1.0 kernel_image
```

2.2 Userspace tool: `brctl`

Once our kernel has the capabilities needed to perform Ethernet Bridge and netfilter actions, we prepare the user space tool `brctl`. `brctl` is the configuration tool we use to set up anything to suit our needs.

We download the source tarball, unpack it and change directory into it.

```
root@bridge:~> wget -c http://bridge.sourceforge.net/bridge-utils/bridge-utils-0.9.5.tar.gz
root@bridge:~> tar xvzf bridge-utils-0.9.5.tar.gz
root@bridge:~> cd bridge-utils-0.9.5
```

At this time, read the `README` and the files in the `doc/` subdirectory. Then do a simple make and copy the resulting `brctl/brctl` executable to `/sbin/`.

```
root@bridge:~> make
root@bridge:~> cp -vi brctl/brctl /sbin/
```

This is it. Go for Setup now.

2.3 Kernel−Notes

Symptom: Anything during setup works but packets do no longer traverse as they did in 2.4 the bridge interfaces.
ipuk s (qasuari_ @ _yahoo.com) wrote (about june 2005):

```
[...]
I have to compile my kernel from 2.4.18-14 to 2.6.0 and activate
bridge-netfilter&ebtables.
After compiling, i can't ping from a host to interface of linux box.
Linux box just have 1 interface.whats wrong with my compilation ???
[...]
```

3. Set Linux up to serve

3.1 Setting up the bridge

We need Linux to know about the bridge. First tell it that we want one virtual ethernet bridge interface: (this is to be executed on host `bridge`, of course. See Testing grounds)

```
root@bridge:~> brctl addbr br0
```

Second, we do not need the STP (Spanning Tree Protocol). I.e. we do only have one single router, so a loop is highly improbable. We may then deactivate this feature. (Results in less polluted networking environment, too):

```
root@bridge:~> brctl stp br0 off
```

After these preparations, we now do finally some effective commands. We add our two (or even more) physical ethernet interfaces. That means, we attach them to the just born logical (virtual) bridge interface `br0`.

```
root@bridge:~> brctl addif br0 eth0
root@bridge:~> brctl addif br0 eth1
```

Important Note:

People sent me emails that it would have helped them if I stressed more clearly the risk of being cut off. So listen at this point to my warnings:
If you read this, you are one (small) step before you _might_ cut yourself off your box you are going to subverse to a bridging device.
If you love living on bleeding edges, it is now the instant to prepare your first aid material. You will likely need it.
If you do not have physical access, nor does another person within your range:
DO NOT PROCEED UNLESS YOUR FINGERS LEFT THE KEYBOARD IN FRONT OF YOU AND YOUR EYES FIXED REFLECTIVELY SOMETHING OTHER THAN YOUR CONSOLE.
You have been warned, now. No responsability is assumed for anything at all.

Now, our two previously physical ethernet interfaces became a logical bridge port each. Erm, ok, there were and will be the physical devices. They are still there, go have a look ;–) But now they became part of the logical bridge device and therefore need no IP configuration any longer. So release the IPs:

```
root@bridge:~> ifconfig eth0 down
root@bridge:~> ifconfig eth1 down
root@bridge:~> ifconfig eth0 0.0.0.0 up
root@bridge:~> ifconfig eth1 0.0.0.0 up
```

Great! We now have a box w/o any IP attached. So if you were configuring your future fw/router via TP, go for your local console now ;–)) You have a serial console? Happy one :–)

Optional:

We tell Linux the new (logical) interface and associate one single IP with it:

```
root@bridge:~> ifconfig br0 10.0.3.129 up
```

And we're done.
Read the Important Note!

3.2 Setting up the routing

In case we are configuring a gateway we enable the forwarding in the linux kernel.

```
root@bridge:~> echo "1" > /proc/sys/net/ipv4/ip_forward
```

Our box already has an IP assigned but no default route. We solve this now:

```
root@bridge:~> route add default gw 10.0.3.129
```

Finally, we should have a working net from, to and through the gateway.

3.3 Make it happen again!

Aka: We need the changes to persist reboots.
To do so, you need some sh−style script and put this in the appropriate system boot−up directory: `/etc/init.d/`
Secondly, you create the link in your runlevel directory. The correct directory depends on your gusto and of course on your linux distribution. Common runlevel values on workstations are `2`, `3` and `5`. Examples are: `/etc/rc?.d/` (replace the ? with the right runlevel)
Also, you need an idea as when your network interfaces are torn up. For now, we assume, your network interfaces are activated at system priority `S` so we need not to care of. If you ever should feel the need to know exactly, look in `/etc/rcS.d/`. We just want the bridge to be up and operable as soon as possible and so chose our priority to be `10`. (Make sure, no service requiring bridging devices is started before, read: with priority−values less than `10`)
For now, we assume, your runlevel is `5`:

```
root@bridge:~> mv −i bridge.sh /etc/init.d/
root@bridge:~> cd /etc/rc5.d/
root@bridge:~> ln −s ../init.d/bridge.sh S10bridge.sh
```

Virtually any distribution provides you with some runlevel−checker or equivalent tool that assists you in the tedious job of administering runlevel links. Consult your distro−documentation on this.
Hint: debian has update−rc.d, redhat and successors have chkconfig. Finally, SuSE evidentally has also it's own tool, too (of which I don't recall the name easily..).
Wondering about the contents of bridge.sh? ;−)

```
#!/bin/bash
PATH="/sbin:/usr/sbin:/usr/local/sbin";
slaveIfs="1 2 3 4 6 7 8 9 10";
```

```
cmd="$1";
[ -z "$cmd" ] && cmd="start";
case "$cmd" in
  start)
    brctl addbr br0;
    brctl stp br0 on;
    brctl addif br0 eth0;
    brctl addif br0 eth1;
    (ifdown eth0 1>/dev/null 2>&1;);
    (ifdown eth1 1>/dev/null 2>&1;);
    ifconfig eth0 0.0.0.0 up;
    ifconfig eth1 0.0.0.0 up;
    ifconfig br0 10.0.3.129 broadcast 10.0.3.255 netmask 255.255.255.0 up ### Adapt to your needs.
    route add default gw 10.0.3.129; ### Adapt to your needs.
    for file in br0 eth0 eth1;
    do
      echo "1" > /proc/sys/net/ipv4/conf/${file}/proxy_arp;
      echo "1" > /proc/sys/net/ipv4/conf/${file}/forwarding;
    done;
    echo "1" > /proc/sys/net/ipv4/ip_forward;
    ;;
  stop)
    brctl delif br0 eth0;
    brctl delif br0 eth1;
    ifconfig br0 down;
    brctl delbr br0;
    #ifup eth0; ### Adapt to your needs.
    #ifup eth1; ### Adapt to your needs.
    ;;
  restart,reload)
    $0 stop;
    sleep 3;
    $0 start;
    ;;
esac;
```

And, yes, make it executable..

```
root@bridge:~> chmod 700 /etc/init.d/bridge.sh
```

After all, make sure your bridge survives unattended reboots. It's the same story as with backups: you should test it before you need it.

---

4. Test your new bridged environment!

4.1 Testing Grounds

We imagine this scenario or similar:

```
                                                          /\
          Ethernet           Ethernet           ATM    /-/  \
---------          ---------          ---------     /-/      |
|  Box  |----------|Bridge |----------|Router |-----| Inter-  \
---------          ---------          ---------     \  net  ---|
         ^        ^         ^        ^               \     /
         |        |         |        |                \---/
```

```
           eth0      eth0       eth1      if0                ^
            |         |          |         |                 |
         10.0.3.2   none/10.0.3.1      195.137.15.7    anything else
                      \         /
                       \       /
      ^                 \-br0-/
      |                                        ^             ^
      |                    ^                   |             |
      |                    |                   |             |
     own                  own               foreign       hostile
```

Our administrative power includes only machines marked with `own`, the Router is completely off−limits and so is the Internet, of course.
That means, if we want to control the flying bits'n'bytes on the ethernet wire we can chose to integrate a common firewall or file in a bridge.
Drawback of the standard way is you have to change the default gateway route on every and any single host in your net. And this is really a heavy weighting drawback, nobody wants to change more than 5 default routes on 5 different hosts more than one time. Keep the time in mind, this will consume, also! Not to forget, this is a error−prone way to handle the more about security..
The other way is clean, less time−consuming, more secure and less error−prone. More secure in that we won't have the need to assign any IP address. No IP, no danger. So far the theory, we hope, our stacks are safe. (Although this hope should better not relied on..) The overall advantage is, this bridge−setup is completely transparent, no IP, MAC, .. changes at all.
So it's up to you to chose your preferred method. But we will handle just the fancy one here ;−)

4.2 Ping it, Jim!

We will configure the Box' eth0 as usual. The bridge's interfaces are configured as described in Setup.
If we are to use forwarding we might perhaps do this one: ;−)

```
root@bridge:~> echo "1" > /proc/sys/net/ipv4/ip_forward
```

Optionally, we set up a default route:

```
root@bridge:~> route add default gw 10.0.3.129
```

Then we set up some iptables rules on host `bridge`:

```
root@bridge:~> iptables -P FORWARD DROP
root@bridge:~> iptables -F FORWARD
root@bridge:~> iptables -I FORWARD -j ACCEPT
root@bridge:~> iptables -I FORWARD -j LOG
root@bridge:~> iptables -I FORWARD -j DROP
root@bridge:~> iptables -A FORWARD -j DROP
root@bridge:~> iptables -x -v --line-numbers -L FORWARD
```

The last line gives us the following output:

```
Chain FORWARD (policy DROP 0 packets, 0 bytes)
num      pkts      bytes target   prot opt in     out     source   destination
1           0         0 DROP       all  --  any    any     anywhere anywhere
```

```
2            0         0 LOG        all  --  any    any     anywhere anywhere      LOG level warning
3            0         0 ACCEPT     all  --  any    any     anywhere anywhere
4            0         0 DROP       all  --  any    any     anywhere anywhere
```

The `LOG` target logs every packet via `syslogd`. Beware, this is intended for testing purposes only, remove in production environment. Else you end up either with filled logs and harddisk partitions by you yourself or anyone else does this Denial of Service to you. You've been warned.
Test this ruleset now. Ping the router interface's IP (195.137.15.7) on host `box`:

```
root@box:~> ping -c 3 195.137.15.7
PING router.provider.net (195.137.15.7) from 10.0.3.2 : 56(84) bytes of data.
--- router.provider.net ping statistics ---
3 packets transmitted, 0 received, 100% loss, time 2020ms
^C
root@box:~>
```

By default, we `DROP` everything. No response, no logged packet. This netfilter setup is designed to `DROP` all packets unless we delete the rule that drops every packet (rule no. 1 above) before the `LOG` target matches:

```
root@bridge:~> iptables -D FORWARD 1
root@bridge:~> iptables -x -v --line-numbers -L FORWARD
```

Now, the rules are:

```
Chain FORWARD (policy DROP 0 packets, 0 bytes)
num      pkts      bytes target   prot opt in     out     source   destination
2           0         0 LOG        all  --  any    any     anywhere anywhere      LOG level warning
3           0         0 ACCEPT     all  --  any    any     anywhere anywhere
4           0         0 DROP       all  --  any    any     anywhere anywhere
```

And any packet may pass through. Test it with a ping on host `box`:

```
root@box:~> ping -c 3 195.137.15.7
PING router.provider.net (195.137.15.7) from 10.0.3.2 : 56(84) bytes of data.
64 bytes from router.provider.net (195.137.15.7): icmp_seq=1 ttl=255 time=0.103 ms
64 bytes from router.provider.net (195.137.15.7): icmp_seq=2 ttl=255 time=0.082 ms
64 bytes from router.provider.net (195.137.15.7): icmp_seq=3 ttl=255 time=0.083 ms

--- router.provider.net ping statistics ---
3 packets transmitted, 3 received, 0% loss, time 2002ms
rtt min/avg/max/mdev = 0.082/0.089/0.103/0.012 ms
root@box:~>
```

Yippeah! The router is alive, up and running. (Well it has been all day long.. ;−)

Important Note:

When we just fired up the bridge interface it takes about roughly 30 seconds until the bridge is fully operational. This is due the 30−seconds−learning phase of the bridge interface. During this phase, the bridge ports are learning what MAC addresses exist on what port. The bridge author, Lennert, tells us in his TODO file, the 30−seconds−learning phase is subjected to some improvement in a timely manner some time.

During the test phase, no packet will we forwarded. No ping be answered. Remind this!

4.3 Actual configuration

This section is intended to give you, dear reader, some hints about how your system should look and feel after having processed this howto successfully.

Interface configuration

The output of your `ifconfig` command might look similar to this:

```
root@bridge:~> ifconfig
br0       Link encap:Ethernet  HWaddr 00:04:75:81:D2:1D
          inet addr:10.0.3.129  Bcast:195.30.198.255  Mask:255.255.255.128
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:826 errors:0 dropped:0 overruns:0 frame:0
          TX packets:737 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:161180 (157.4 Kb)  TX bytes:66708 (65.1 Kb)

eth0      Link encap:Ethernet  HWaddr 00:04:75:81:ED:B7
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5729 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3115 errors:0 dropped:0 overruns:0 carrier:656
          collisions:0 txqueuelen:100
          RX bytes:1922290 (1.8 Mb)  TX bytes:298837 (291.8 Kb)
          Interrupt:11 Base address:0xe400

eth1      Link encap:Ethernet  HWaddr 00:04:75:81:D2:1D
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:1 frame:0
          TX packets:243 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:342 (342.0 b)  TX bytes:48379 (47.2 Kb)
          Interrupt:7 Base address:0xe800

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1034 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1034 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:82068 (80.1 Kb)  TX bytes:82068 (80.1 Kb)
```

Routing configuration

The output of your `route` command might look similar to this:

```
root@bridge:~> route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.3.129      0.0.0.0         255.255.255.128 U     0      0        0 br0
0.0.0.0         10.0.3.129      0.0.0.0         UG    0      0        0 br0
root@bridge:~>
```

Iptables configuration

Please have a look at the Ping it, Jim! section.

4.4 Final note (Important!)

I'd like to hear from you! :−)
Did you enjoy the trip?
Do you miss anything?
Need help? (Call you local assistant ;−) or rtfm.
You are still online? Then drop me a msg via email. I'd be really glad.
Wanna send me a cheque? Pitty, Don't accept these.. (Just kidding;)
Make it worth my time, just send me some nice words, that's enough.
Nothing motivates more than happy participants giving you valuable feedback.
So, go on, invest a minute and hack me a mail!
Thank you!

```
Nils
```

4.5 Bug−Notes

Apparently, there must have been a bug in the br−nf code:

```
From: Bart De Schuymer <bart.de.schuymer_@_pandora.be>
Date: Sun, 1 Sep 2002 21:52:46 +0200
To: Nils Radtke <Nils.Radtke_@_Think-Future.de>
Subject: Re: Ethernet-Brigde-netfilter-HOWTO

Hello Nils,

[...]
Also, network packet filtering debugging is generally a bad idea with the
br-nf patch. It can gives a lot of false warnings (about bugs) in the logs.
[...]
```

Personally, I never had false positives in my log. Maybe, that bug has been fixed. This mailed to Bart, he wrote:

```
From: Bart De Schuymer <bart.de.schuymer_@_pandora.be>
Date: Mon, 2 Sep 2002 18:30:25 +0200
To: Nils Radtke <Nils.Radtke_@_Think-Future.de>
Subject: Re: Ethernet-Brigde-netfilter-HOWTO

On Monday 02 September 2002 00:39, Nils Radtke wrote:
> Will the revision of the nf-debug code in br-nf be subject of improvement?

I must admit I haven't been running any kernel with netfilter debugging
lately. It sure used to give false positives a few months ago (the bridge
mailing list has posts about that), I've been lacking time to see why and if
it is still the case. It's on my todo list.
[...]
```

But (as of writing this 2002−09−19) I haven't found an official announcement, this particular bug has been closed. So have a constant look at this topic on the ethernet bridge mailinglist , if you are interested in it's cure.

5. User experiences

5.1 Fedora Core 3

James Dinkel (jdinkel_ @ _gmail.com) wrote on Tue, 8 Mar 2005 10:59:22 −0600:

```
[...]
I am using Fedora Core 3 and all I had to do was "yum install bridge-utils"
to use the brctl command. I didn't have to do any kernel recompiling or
configurations or messing with kernel modules.
It was very easy.
[...]
```

6. Links

The Howto's author may be contacted via e−mail.
Howto Author's homepage.

6.1 Ethernet−Bridge

- `Ethernet Bridge Mailinglist`
- User space utilities, patches, etc.: `Home of Linux kernel Ethernet Bridge`
- `Bridge-STP-HOWTO`
- `Firewalling for Free, Shawn Grimes`

6.2 Related Topics

- Filtering on frame level, Ethernet−Bridging−Tables:
  `ebtables, sourceforge`
  `ebtables, supported features`
  `ebtables, examples: basic, advanced`
- IP mode, Linux Bridge extension: `IP mode, LVS`
- Linux in High−Availability environments: `High-Availability Linux`
- Linux Virtual Server: `LVS`