# Linux Foundation Wiki

project collaboration site

# bridge

Translations: russian [http://xgu.ru/wiki/Linux_Bridge] ,Turkish [http://wiki.pardus-linux.org/index.php/Programlar:Bridge-Utilities]

A bridge is a way to connect two Ethernet [http://en.wikipedia.com/wiki/Ethernet] segments together in a protocol independent way. Packets are forwarded based on Ethernet address, rather than IP address (like a router). Since forwarding is done at Layer 2, all protocols can go transparently through a bridge.

The Linux bridge code implements a subset of the ANSI/IEEE 802.1d standard. [1] [http://standards.ieee.org/getieee802/]. The original Linux bridging was first done in Linux 2.2, then rewritten by Lennert Buytenhek. The code for bridging has been integrated into 2.4 and 2.6 kernel series.

## Bridging and Firewalling

A Linux bridge is more powerful than a pure hardware bridge because it can also filter and shape traffic. The combination of bridging and firewalling is done with the companion projectebtables [http://ebtables.sourceforge.net].

# Status

The code is updated as part of the 2.4 and 2.6 kernels available at kernel.org.

Possible future enhancements are:

- Document STP filtering
- Netlink interface to control bridges (prototype in 2.6.18)
- STP should be in user space
- Support RSTP and other 802.1d STP extensions

# Downloading

Bridging is supported in the current 2.4 (and 2.6) kernels from all the major distributors. The required administration utilities are in the bridge-utils package in most distributions. Package releases are maintained on the Download [http://sourceforge.net/project/showfiles.php?group_id=26089] page.

You can also build your own up to date version by getting the latest kernel from kernel.org [http://kernel.org] and build the utilities based from the source code in bridge-utils GIT repository.

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/shemminger/bridge-utils.git
$ cd bridge-utils
$ autoconf
$ ./configure
```

# Kernel Configuration

You need to enable bridging in the kernel. Set "networking → 802.1d Ethernet Bridging" to either yes or module

# Manual Configuration

## Network cards

Before you start make sure both network cards are set up and working properly. Don't set the IP address, and don't let the startup scripts run DHCP on the ethernet interfaces either. The IP address needs to be set after the bridge has been configured.

The command **ifconfig** should show both network cards, and they should be DOWN.

## Module loading

In most cases, the bridge code is built as a module. If the module is configured and installed correctly, it will get automatically loaded on the first **brctl** command.

If your bridge-utilities have been correctly built and your kernel and bridge-module are OK, then issuing a **brctl** should show a small command synopsis.

```
# brctl
# commands:
        addbr           <bridge>                add bridge
        delbr           <bridge>                delete bridge
        addif           <bridge> <device>       add interface to bridge
        delif           <bridge> <device>       delete interface from bridge
        setageing       <bridge> <time>         set ageing time
```

```
setbridgeprio   <bridge> <prio>          set bridge priority
setfd           <bridge> <time>          set bridge forward delay
sethello        <bridge> <time>          set hello time
setmaxage       <bridge> <time>          set max message age
setpathcost     <bridge> <port> <cost>   set path cost
setportprio     <bridge> <port> <prio>   set port priority
show                                     show a list of bridges
showmacs        <bridge>                 show a list of mac addrs
showstp         <bridge>                 show bridge stp info
stp             <bridge> <state>         turn stp on/off
```

## Creating a bridge device

The command

```
brctl addbr "bridgename"
```

creates a logical bridge instance with the name *bridgename*. You will need at least one logical instance to do any bridging at all. You can interpret the logical bridge as a container for the interfaces taking part in the bridging. Each bridging instance is represented by a new network interface.

The corresponding shutdown command is:

```
brctl delbr //bridgename//
```

## Adding devices to a bridge

The command

```
brctl addif //bridgename// //device//
```

adds the network device *device* to take part in the bridging of "bridgename." All the devices contained in a bridge act as one big network. It is not possible to add a device to multiple bridges or bridge a bridge device, because it just wouldn't make any sense! The bridge will take a short amount of time when a device is added to learn the Ethernet addresses on the segment before starting to forward.

The corresponding command to take an interface out of the bridge is:

```
brctl delif//bridgename// //device//
```

## Showing devices in a bridge

The *brctl show* command gives you a summary about the overall bridge status, and the instances running as shown below:

```
# brctl addbr br549
# brctl addif br549 eth0
# brctl addif br549 eth1
# brctl show
bridge name     bridge id               STP enabled     interfaces
br549           8000.00004c9f0bd2       no              eth0
                                                        eth1
```

Once a bridge is running the *brctl showmacs* will show information about network addresses of traffic being forwarded (and the bridge itself).

```
# brctl showmacs br549
port no mac addr                is local?       ageing timer
  1     00:00:4c:9f:0b:ae       no                17.84
  1     00:00:4c:9f:0b:d2       yes                0.00
  2     00:00:4c:9f:0b:d3       yes                0.00
  1     00:02:55:1a:35:09       no                53.84
  1     00:02:55:1a:82:87       no                11.53
 ...
```

The aging time is the number of seconds a MAC address will be kept in the forwarding database after having received a packet from this MAC address. The entries in the forwarding database are periodically timed out to ensure they won't stay around forever. Normally there should be no need to modify this parameter, but it can be changed with (time is in seconds).

```
# brctl setageing //bridgename// //time//
```

Setting ageing time to zero makes all entries permanent.

## Spanning Tree Protocol

If you are running multiple or redundant bridges, then you need to enable the Spanning Tree Protocol (STP) to handle multiple hops and avoid cyclic routes.

```
# brctl stp br549 on
```

You can see the STP parameters with:

```
# brctl showstp br549
br549
```

```
 bridge id              8000.00004c9f0bd2
 designated root        0000.000480295a00
 root port                 1                    path cost                 104
 max age                  20.00                 bridge max age            200.00
 hello time                2.00                 bridge hello time          20.00
 forward delay           150.00                 bridge forward delay       15.00
 ageing time             300.00                 gc interval                 0.00
 hello timer               0.00                 tcn timer                   0.00
 topology change timer     0.00                 gc timer                    0.33
 flags

eth0 (1)
 port id                8001                   state                   forwarding
 designated root        0000.000480295a00      path cost                 100
 designated bridge      001e.00048026b901      message age timer          17.84
 designated port        80c1                   forward delay timer         0.00
 designated cost           4                   hold timer                  0.00
 flags

eth1 (2)
 port id                8002                   state                   disabled
 designated root        8000.00004c9f0bd2      path cost                 100
 designated bridge      8000.00004c9f0bd2      message age timer           0.00
 designated port        8002                   forward delay timer         0.00
 designated cost           0                   hold timer                  0.00
 flags
```

## STP tuning

There are a number of parameters related to the Spanning Tree Protocol that can be configured. The code autodetects the speed of the link and other parameters, so these usually don't need to be changed.

**Bridge priority**

Each bridge has a relative priority and cost. Each interface is associated with a port (number) in the STP code. Each has a priority and a cost, that is used to decide which is the shortest path to forward a packet. The lowest cost path is always used unless the other path is down. If you have multiple bridges and interfaces then you may need to adjust the priorities to achieve optimium performance.

```
# brctl setbridgeprio //bridgename// //priority//
```

The bridge with the lowest priority will be elected as the root bridge. The root bridge is the "central" bridge in the spanning tree.

**Path priority and cost**

Each interface in a bridge could have a different speed and this value is used when deciding which link to use. Faster interfaces should have lower costs.

```
# brctl //setpathcost bridge port cost//
```

For multiple ports with the same cost there is also a priority

**Forwarding delay**

Forwarding delay time is the time spent in each of the Listening and Learning states before the Forwarding state is entered. This delay is so that when a new bridge comes onto a busy network it looks at some traffic before participating.

```
#  brctl setfd //bridgename// //time//
```

**Hello time**

Periodically, a hello packet is sent out by the Root Bridge and the Designated Bridges. Hello packets are used to communicate information about the topology throughout the entire Bridged Local Area Network.

```
# brctl sethello //bridgename// //time//
```

**Max age**

If a another bridge in the spanning tree does not send out a hello packet for a long period of time, it is assumed to be dead. This timeout is set with:

```
# brctl maxage//bridgename// //time//
```

## Multicast (IGMP) snooping

IGMP snooping support is not yet included in bridge-utils or iproute2, but it can be easily controlled through sysfs interface. For brN, the settings can be found under /sys/devices/virtual/net/brN/bridge.

```
multicast_snooping
```

This option allows the user to disable IGMP snooping completely. It also allows the user to reenable snooping when it has been automatically disabled due to hash collisions. If the collisions have not been resolved however the system will refuse to reenable snooping.

`multicast_router`

This allows the user to forcibly enable/disable ports as having multicast routers attached. A port with a multicast router will receive all multicast traffic.

The value 0 disables it completely. The default is 1 which lets the system automatically detect the presence of routers (currently this is limited to picking up queries), and 2 means that the ports will always receive all multicast traffic.

Note: this setting can be enabled/disable on a per-port basis, also through sysfs interface (e.g. if eth0 is some bridge's active port, then you can adjust /sys/…../eth0/brport/multicast_router)

`hash_{max,elasticity}`

These settings allow the user to control the hash elasticity/max parameters. The elasticity setting does not take effect until the next new multicast group is added. At which point it is checked and if after rehashing it still can't be satisfied then snooping will be disabled.

The max setting on the other hand takes effect immediately. It must be a power of two and cannot be set to a value less than the current number of multicast group entries. This is the only way to shrink the multicast hash.

`remaining multicast_* options`

These allow the user to control various values related to IGMP snooping.

More details about the options, some discussions and rationale can be found in http://thread.gmane.org/gmane.linux.network/153338

## Sample setup

The basic setup of a bridge is done like:

```
# ifconfig eth0 0.0.0.0
# ifconfig eth1 0.0.0.0
# brctl addbr mybridge
# brctl addif mybridge eth0
# brctl addif mybridge eth1
# ifconfig mybridge up
```

This will set the host up as a pure bridge, it will not have an IP address for itself, so it can not be remotely accessed (or hacked) via TCP/IP.

Optionally you can configure the virtual interface mybridge to take part in your network. It behaves like one interface (like a normal network card). Exactly that way you configure it, replacing the previous command with something like:

```
# ifconfig mybridge 192.168.100.5 netmask 255.255.255.0
```

If you want your bridge to automatically get its IP address from the ADSL modem via DHCP (or a similar configuration), do this:

```
# ifconfig eth0 0.0.0.0
# ifconfig eth1 0.0.0.0
```

```
# brctl addbr mybridge
# brctl addif mybridge eth0
# brctl addif mybridge eth1
# dhclient mybridge
```

If you do this many times, you may end up with lots of dhclient processes. Either kill them impolitely or learn about omshell(1) [http://linuxreviews.org/man/omshell/].

# Configuration with /etc/net

In /etc/net we first configure two ethernet devices port0 and port1:

```
# cat >> /etc/net/iftab
port0 mac 00:13:46:66:01:5e
port1 mac 00:13:46:66:01:5f
^D
# mkdir /etc/net/ifaces/port0
# cat > /etc/net/ifaces/port0/options
TYPE=eth
MODULE=via-rhine
# mkdir /etc/net/ifaces/port1
# cat > /etc/net/ifaces/port1/options
TYPE=eth
MODULE=via-rhine
^D
```

Then we describe the bridge:

```
# mkdir /etc/net/ifaces/mybridge
# cat > /etc/net/ifaces/mybridge/options
TYPE=bri
HOST='port0 port1'
^D
# cat > /etc/net/ifaces/mybridge/brctl
stp AUTO on
^D
```

Now we can use "ifup mybridge" to bring it up. port0 and port1 will be brought up automatically.

# FAQ

## What does a bridge do?

A bridge transparently relays traffic between multiple network interfaces. In plain English this means that a bridge connects two or more physical Ethernets together to form one bigger (logical) Ethernet.

## Is it protocol independent?

Yes. The bridge knows nothing about protocols, it only sees Ethernet frames. As such, the bridging functionality is protocol independent, and there should be no trouble relaying IPX, NetBEUI, IP, IPv6, etc.

## Why is this code better than a switch?

Please note that this code wasn't written with the intent of having Linux boxes take over from dedicated networking hardware. Don't see the Linux bridging code as a replacement for switches, but rather as an extension of the Linux networking capabilities. Just as there are situations where a Linux router is better than a dedicated router (and vice versa), there are situations where a Linux bridge is better than a dedicated bridge (and vice versa).

Most of the power of the Linux bridging code lies in its flexibility. There is a whole lot of bizarre stuff you can do with Linux already (read Linux Advanced Routing and Traffic Control document to see some of the possiblities), and the bridging code adds some more filter into the mix.

One of the most significant advantages of a Linux solution over a dedicated solution that come to mind is Linux' extensive firewalling capabilities. It is possible to use the full functionality of netfilter (iptables) in combination with bridging, which provides way more functionality than most proprietary offerings do.

## Why is this code worse than a switch?

In order to act a a bridge, the network device must be placed into promiscuous mode which means it receives all traffic on a network. On a really busy network, this can eat significant bandwidth out of the processor, memory slowing the system down. The answer is to setup either a separate dedicated Linux box as the bridge, or use a hardware switch.

## What is the performance of the bridge?

The performance is limited by the network cards used and the processor. A research paper was done by James Yu at Depaul University comparing Linux bridging with a Catalyst switchYu-Linux-TSM2004.pdf [http://facweb.cti.depaul.edu/jyu/Publications/Yu-Linux-TSM2004.pdf]

## My bridge does not show up in traceroute!

It's not supposed to. The operation of a bridge is (supposed to be) fully transparent to the network, the networks that a bridge connects together are actually to be viewed as one big network. That's why the bridge does not show up in traceroute; the packets do not feel like they are crossing a subnet boundary.

For more information on this, read a book about TCP/IP networking.

## It doesn't work!

It says: “br_add_bridge: bad address” when I try to add a bridge!

Either your kernel is old (2.2 or earlier), or you forgot to configure Ethernet bridging into your kernel.

## No traffic gets trough (except ARP and STP)

Your kernel might have ethernet filtering (ebtables, bridge-nf, arptables) enabled, and traffic gets filtered. The easiest way to disable this is to go to /proc/sys/net/bridge. Check if the bridge-nf-* entries in there are set to 1; in that case, set them to zero and try again.

```
# cd /proc/sys/net/bridge
# ls
bridge-nf-call-arptables  bridge-nf-call-iptables
bridge-nf-call-ip6tables  bridge-nf-filter-vlan-tagged
# for f in bridge-nf-*; do echo 0 > $f; done
```

## Does bridging work on 2.2?

The base kernel for 2.2, did not support the current bridging code. The original development was on 2.2, and there used to be patches available for it. But these patches are no longer maintained.

## Are there plans for RSTP (802.1w) support?

Yes, work is being done to integrate RSTP support in a future 2.6 release. The code was done for a version of 2.4 and needs to be cleaned up, tested and updated.

## What can be bridged?

Linux bridging is very flexible; the LAN's can be either traditional Ethernet device's, or pseudo-devices such as PPP, VPN's or VLAN's. The only restrictions are that the devices:

- All devices share the same maximum packet size (MTU). The bridge doesn't fragment packets.
- Devices must look like Ethernet. i.e have 6 byte source and destination address.
- Support promiscuous operation. The bridge needs to be able to receive all network traffic, not just traffic destined for its own address.
- Allow source address spoofing. The bridge must be able to send data over network as if it came from another host.

## Can I do bridging in combination with netfilter/iptables?

Yes. The code for this is available in most kernels. See ebtables project. Does it work with Token Ring [http://en.wikipedia.com/wiki/Token_Ring], FDDI [http://en.wikipedia.com/wiki/FDDI], or Firewire?

No, the addressing and frame sizes are different.

## I keep getting the message **retransmitting tcn bpdu**!

It means that your Linux bridge is retransmitting a Topology Change Notification Bridge Protocol Data Unit (so now you know what the letters are for  . Seriously, there is probably another switch (or Linux bridge) nearby that isn't complying to the rules of the spanning tree protocol (which is what bridges speak).

In each bridged local area network, there is one 'master bridge', which is also called the root bridge. You can find out which bridge this is using brctl.

When the topology of a bridged local area network changes (f.e. somebody unplugs a cable between two bridges), the bridge which detects this sends a topology change notification to the root bridge. The root bridge will respond to this by setting a 'topology changed' bit in the hello packets it sends out for the next X seconds (X usually being 30). This way, all bridges will learn of the topology change, so that they can take measures like timing out learned MAC entries faster for example.

After having sent out a topology change notification, if a bridge does not find the 'topology changed' bit set in the hello packets received (which in essence serves as the 'acknowledgment' of the topology change notification), it concludes that the topology change notification was lost. So it will retransmit it. However, some bridges run lobotomized implementations of the Spanning Tree Protocol which causes them not to acknowledge topology change notifications. If you have one of those bridges as your root bridge, all of the other bridges will keep retransmitting their topology changed notifications. Which will lead to these kinds of syslog messages.

There are a number of things you can do:

- Find out which bridge is the root bridge, find out where it is located, and what internetworking software it runs. Please report this info to the mailing list (or to me directly), so that I can keep a blacklist.

- Force the linux bridge to be the root bridge. See what the priority of the current root bridge is, and use the brctl 'setbridgeprio' command to set the priority of the linux bridge to 1 lower. (The bridge with the lowest priority always becomes the root bridge.)
- Disable the spanning tree protocol on your linux bridge altogether. In this case, watch out for bridging loops! If you have loops in your topology, and if no bridge in the loop is running the spanning tree protocol, mayhem will come your way, as packets will be forwarded forever. Don't Do This(TM).

## It doesn't work with my regular Ethernet card!

Unfortunately, some network cards have buggy drivers that fail under load. The situation is improving, so having a current kernel and network driver can help. Also try swapping with another brand.

Please report all problems to the Bridge mailing list: bridge@osdl.org. If your network card doesn't work (even without bridging) then try the Linux networking mailing list linux-net@vger.kernel.org

## It doesn't work with my Wireless card!

This is a known problem, and it is not caused by the bridge code. Many wireless cards don't allow spoofing of the source address. It is a firmware restriction with some chipsets. You might find some information in the bridge mailing list archives to help. Has anyone found a way to get around Wavelan not allowing anything but its own MAC address? (answer by Michael Renzmann (mrenzmann at compulan.de))

Well, for 99% of computer users there will never be a way to get rid of this. For this function a special firmware is needed. This firmware can be loaded into the RAM of any WaveLAN card, so it could do its job with bridging. But there is no documentation on the interface available to the public. The only way to achieve this is to have a full version of the hcf library which controls every function of the card and also allows accessing the card's RAM.

To get this full version Lucent wants to know that it will be a financial win for them, also you have to sign an NDA. So be sure that you won't most probably get access to this peace of software until Lucent does not change its mind in this (which I doubt never will happen).

## I still don't understand!!

Doing full bridging of wireless (802.11) requires supporting WDS [https://www.linuxfoundation.org/node/add/wiki?gids[]=5066]. The current implementation doesn't do it.

It is possible to do limited wireless to Ethernet functionality with some wireless drivers. This requires the device to be able to support a different sender address and source address. That is what WDS provides.

There are ways to make it work, but it is not always straightforward and you probably won't get it right without a pretty solid understanding of 802.11, it's modes, and the frame header format.

## I get the error 'too much work in interrupt'

This is because the network card is getting lots of packets. There are a few things you can try. First, build the driver with NAPI support (if it isn't on by default). NAPI means the driver will do the receive processing at soft IRQ, not at the low level interrupt.

If the driver doesn't support NAPI, you can try to increase the amount of work a driver will attempt to do in an interrupt. For 3c59x this is done with the option max_interrupt_work (so add something like 'options 3c59x max_interrupt_work=10000' to your /etc/modules.conf file), other cards might have similar options.

## Does DHCP work over/through a bridge?

The bridge will forward DHCP traffic (broadcasts) and responses. You can also use DHCP to set the local IP address of the bridge pseudo-interface.

One common mistake is that the default bridge forwarding delay setting is 30 seconds. This means that for the first 30 seconds after an interface joins a bridge, it won't send anything. This is because if the bridge is being used in a complex topology, it needs to discover other bridges and not create loops. This problem was one of the reasons for the creation of Rapid Spanning Tree Protocol (RSTP).

If the bridge is being used standalone (no other bridges near by). Then it is safe to turn the forwarding delay off (set it to zero), before adding interface to a bridge. Then you can run DHCP client right away.

```
# brctl setfd br0 0
# brctl addif br0 eth0
# dhclient eth0
```

# Contact Info

The code is currently maintained by Stephen Hemminger<shemminger@osdl.org> [mailto:mailto:shemminger@osdl.org] for both 2.4 and 2.6 Linux. Bridge bugs and enhancements are discussed on the Bridge mailing list<bridge@osdl.org> [mailto:mailto:bridge@osdl.org]. The list is open to anyone interested, use the web mailman interfacehttp://lists.osdl.org/mailman/listinfo/bridge [http://lists.osdl.org/mailman/listinfo/bridge] to subscribe.

# External Links

- Ethernet VPN bridging [http://openvpn.sourceforge.net/bridge.html]

- Ebtables firewalling [http://ebtables.sourceforge.net]
- Ethernet-bridge HOWTO [http://www.tldp.org/HOWTO/Ethernet-Bridge-netfilter-HOWTO.html]
- Linux-bridge STP HOWTO [http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO/index.html]
- wikipedia:Spanning_tree_protocol [http://en.wikipedia.com/wiki/Spanning_tree_protocol]
- Understanding Spanning-Tree Protocol (Cisco) [http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/sw_ntman/cwsimain/cwsi2/cwsiug2/vlan2/stpapp.htm]

---