

Building a fault-tolerant firewall system with virtual machines

TUYEN THANH PHAM



APRIL 15, 2025

Contents

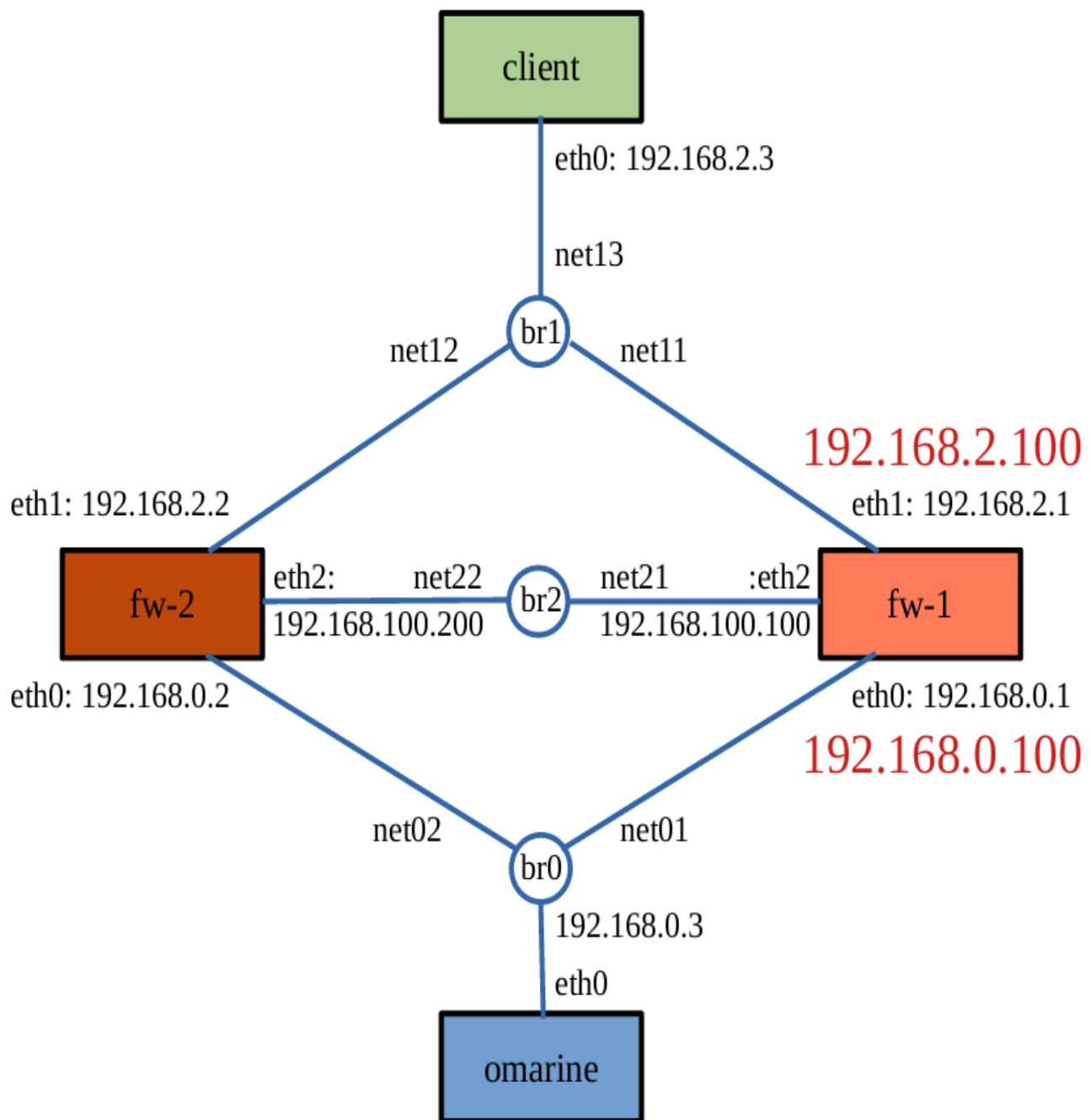
1. Introduction	3
2. Creating bridge devices and tap interfaces	6
3. Creating disk image	9
4. Creating virtual machine with qemu	13
5. Accessing remote virtual machine	17
6. Configuring X to use qxl video driver	17
7. Network configuration using systemd	19
8. Testing spice agent and USB redirection	22
9. Routing	23
10. Configuring HA and conntrackd	26
11. Load balancing	30
12. Expectation: part 1: helper	34
13. Expectation: part 2: expectation	45
14. Expectation: part 3: diagram	51
15. Writing a firewall ruleset	52
16. Kernel patch due to loss of helper	55
17. References	57

1. Introduction

Combining netfilter connection tracking tool conntrackd with HA (High Availability) service using keepalived we can build a fault tolerant firewall system. With qemu and spice, the system is built based on virtual machines that are almost as friendly as the physical ones. Users can use virtual machines remotely via the spice protocol and can perform copy and paste operations between virtual machines and the real machine and between virtual machines. We can also redirect USB from the real machine to the virtual machine and then use the USB devices on the virtual machine with the feeling of using them on a physical machine.

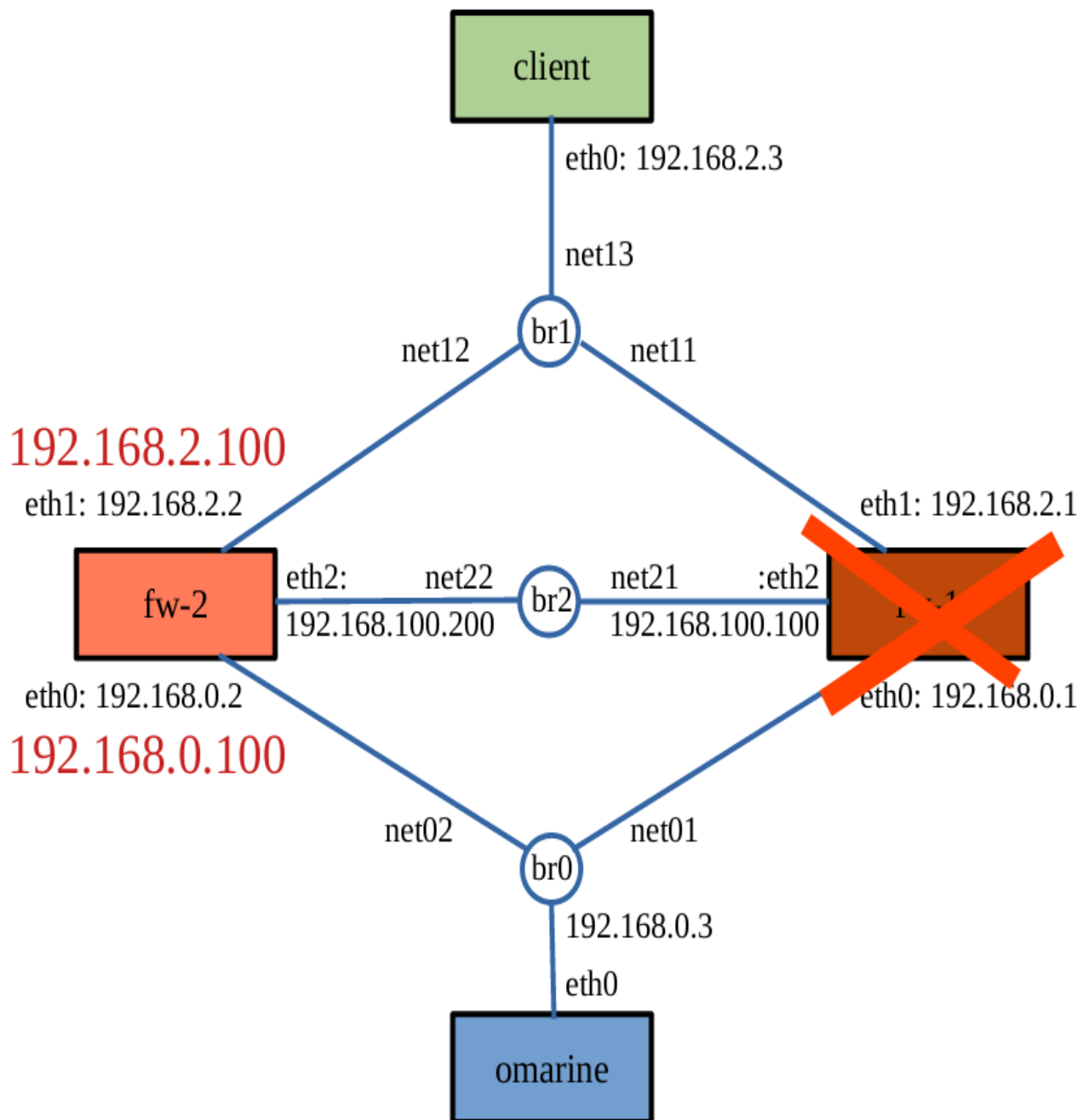
This series of articles introduces building and running a fault-tolerant firewall system through a real-world example. Among them are techniques for creating virtual network interfaces, designing virtual machines using qemu command input parameters, accessing and using remote virtual machines using spice, running spice agent on virtual machines to communicate with spice server on the host, configuring X to use the qxl video driver on the virtual machines, configuring network interfaces using systemd, routing, network address translation, configuring HA and running keepalived on the firewall machines, integrating conntrackd into the system and running conntrackd.service on firewall machines, etc. Finally, a set of stateful packet filtering rules are set up, precise to each specific network interface on the firewall machines to specify which services the client is allowed to access on the server.

In the figure below, when the machine client wants to access any service on the server omarine every packets must go through a firewall system consisting of two machines fw-1 and fw-2. This is the primary/backup model of the HA system. Only one firewall machine at a time is responsible for packet filtering. If fw-1 starts first it will be the primary firewall, machine fw-2 is the backup. When the client accesses omarine, a packet with a destination address of 192.168.0.3 will go out at the interface eth0. It goes through the bridge br1 and then to the firewall fw-1 at the interface eth1. The packet does not go inside the firewall machine but is forwarded from the eth1 interface to eth0 interface. The packet filtering rule set is active from the moment eth1 receives the packet. It will decide to allow or drop the packet on the floor. If accepted, the packet is sent from the firewall's eth0 interface, across the bridge br0, and then into omarine at the eth0 interface



Although the firewall fw-2 does not perform packet filtering, it is replicated conntrack states through the eth2 interfaces between two firewalls connected via the br2 bridge.

If the firewall fw-1 fails, the firewall fw-2 immediately becomes active. Because of its full packet filtering tracking states, the firewall fw-2 from the moment of handover is able to distinguish a connection as established or a new one, and becomes primary firewall effectively

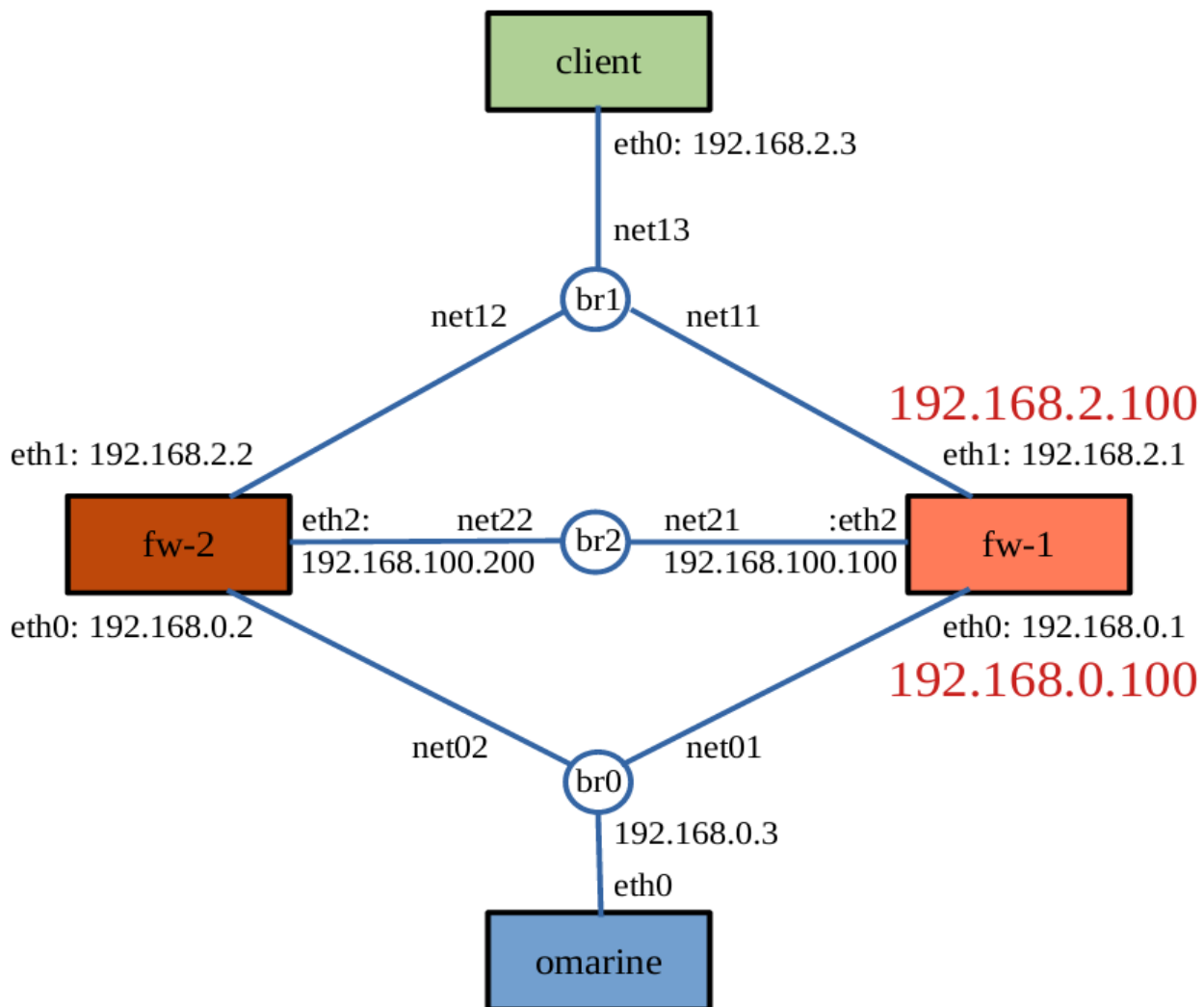


2. Creating bridge devices and tap interfaces

The tap network backend is the most appropriate configuration option in qemu for us to create network interfaces for virtual machines because virtual ethernet interfaces created in such a way are considered as normal ethernet devices without any restriction. We also need bridge devices to connect the interfaces in the networks.

By design, a bridge device is used to attach ethernet interfaces at its bridge ports. The bridge device then becomes a logically large ethernet interface consisting of a bunch of attached ethernet interfaces. The bridge is addressed instead of the ethernet interfaces participating in the bridge. In this case the bridge becomes a gateway to the network of the participating ethernet interfaces. But our firewall system does not use such functionality.

Non-grounding bridge solution



In the diagram above we see there are two networks. The client is in the 192.168.2.0/24 network and the omarine server is in the 192.168.0.0/24 network. Firewalls are also the gateway for traffic to flow from one network to another. The interesting thing is that they are automatic routers. If the active router fw-1 suddenly stops, the router fw-2 will automatically take its place.

Imagine the machine omarine together with the bridge br0 is on the 1st floor, i.e. on the ground. Two firewalls are on the 2nd floor and the client is on the 3rd. A packet that wants to go from the client, i.e. on the 3rd floor to the omarine on the 1st floor, must go through the 2nd layer, ie through the firewall system. The bridge br0, which is inherent in the host machine (omarine), is addressed to 192.168.0.3 as usual and is present in the main routing table. We call it a “grounding” bridge. The bridges br1 and br2 are overhead, unaddressed, and not present in any of the host’s routing tables. The packets at these bridges have no path to go directly to the ground, it is forced to follow the network traffic that we have designed. We call the bridges br1 and br2 “non-grounding” bridges.

Creating bridge devices and tap interfaces

First we create two virtual bridge devices br1 and br2, and turn them up:

```
sudo ip link add br1 up type bridge
sudo ip link add br2 up type bridge
```

Then create tap interfaces: net01, net02 for bridge br0; net11, net12, net13 for bridge br1; net21, net22 for bridge br2:

```
sudo ip tuntap add net01 mode tap
sudo ip tuntap add net02 mode tap
sudo ip tuntap add net11 mode tap
sudo ip tuntap add net12 mode tap
sudo ip tuntap add net13 mode tap
sudo ip tuntap add net21 mode tap
sudo ip tuntap add net22 mode tap
```

Set the tap interfaces to up:

```
sudo ip link set net01 up
sudo ip link set net02 up
sudo ip link set net11 up
sudo ip link set net12 up
sudo ip link set net13 up
sudo ip link set net21 up
sudo ip link set net22 up
```

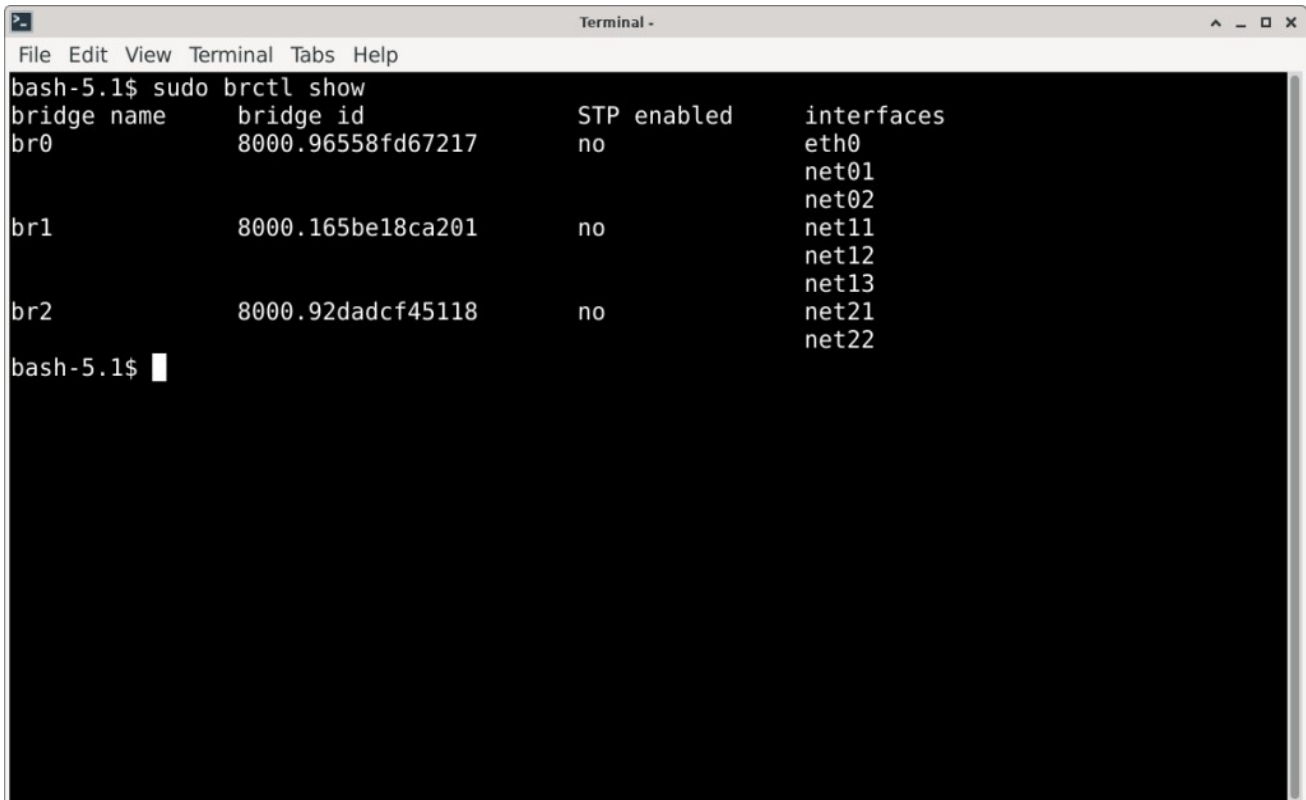
Attach net01, net02 to bridge br0; attach net11, net12, net13 to bridge br1; attach net21, net22 to bridge br2:

```
sudo brctl addif br0 net01
sudo brctl addif br0 net02
sudo brctl addif br1 net11
```

```
sudo brctl addif br1 net12
sudo brctl addif br1 net13
sudo brctl addif br2 net21
sudo brctl addif br2 net22
```

To view the bridges and the interfaces attached to them, run the command below:

```
sudo brctl show
```



```
File Edit View Terminal Tabs Help
bash-5.1$ sudo brctl show
bridge name      bridge id        STP enabled      interfaces
br0              8000.96558fd67217 no                eth0
                  net01
                  net02
br1              8000.165be18ca201 no                net11
                  net12
                  net13
br2              8000.92dadcf45118 no                net21
                  net22
bash-5.1$
```

Adding modules to the kernel

Once using the bridge devices, you need to add the module `br_netfilter` to the kernel if you want to filter packets at bridge ports and bridge interfaces. That is, use the module `'physdev'` in iptables and the family `'bridge'` in nftables. Also add the module `nf_conntrack_bridge` to the kernel to track packet filtering against the bridges if necessary. It is best to add these two modules at boot time (firewalld needs that). You add the following content to the kernel command line

```
rd.driver.pre=br_netfilter,nf_conntrack_bridge
```

You can edit the `/boot/grub/grub.cfg` file directly to modify the kernel command line, or run the two commands below:

```
echo \
GRUB_CMDLINE_LINUX=\"rd.driver.pre=br_netfilter,nf_conntrack_bridge\"
| sudo tee -a /etc/default/grub
sudo grub-mkconfig -o /boot/grub/grub.cfg
```


Remove bridges from influence zones of firewalld

If you're running firewalld, you'll need to keep the bridges out of firewalld's zones of influence. We will write a set of specific packet filtering rules for firewall machines later.

You run the commands below:

```
sudo firewall-cmd --zone=trusted --add-interface=br0
sudo firewall-cmd --zone=trusted --add-interface=br1
sudo firewall-cmd --zone=trusted --add-interface=br2
```

Since br0 is automatically configured at host startup, we make it permanent:

```
sudo firewall-cmd --permanent --zone=trusted --add-interface=br0
```

3. Creating disk image

Each virtual machine needs a disk image containing the operating system. We have three virtual machines so we need three disk images. However, just create one image and then make copies of the other two.

Creating partitions and filesystem on disk

An operating system generally needs one root partition and one swap partition. We practice on a USB stick or a USB hard disk. Assuming the device name is /dev/sdb. The commands below create a 10G root partition (exactly, minus the first 1M on disk) and a 2G swap partition, and create the root filesystem on the root partition

```
sudo wipefs -a /dev/sdb
sudo parted /dev/sdb mklabel msdos
sudo parted /dev/sdb mkpart primary ext4 1MiB 10240MiB
sudo parted /dev/sdb mkpart primary linux-swap 10240MiB 12288MiB
sudo mkfs.ext4 -L firewall /dev/sdb1
```

To view info of the partitions and the root filesystem, run the following commands:

```
sudo parted /dev/sdb unit MiB print
sudo fsck /dev/sdb1
```

```
Terminal -
File Edit View Terminal Tabs Help
bash-5.1$ sudo parted /dev/sdb unit MiB print
Model: Kingston DataTraveler 3.0 (scsi)
Disk /dev/sdb: 14755MiB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start      End          Size         Type         File system  Flags
  1      1.00MiB    10240MiB     10239MiB     primary      ext4
  2      10240MiB   12288MiB     2048MiB      primary

bash-5.1$ sudo fsck /dev/sdb1
fsck from util-linux 2.37.2
e2fsck 1.46.2 (28-Feb-2021)
firewall: clean, 11/655360 files, 66753/2621184 blocks
bash-5.1$
```

The size of the root partition is 10239M. The filesystem has 2621184 blocks. Each block 4K. So the size of the filesystem is $2621184 * 4 / 1024 = 10239$ (M), equal to the partition size. However, these two dimensions are not always equal.

We need to calculate the filesystem size exactly because the disk image needs to be compact. Next is to install the operating system onto the disk as usual.

Increasing the filesystem size

Maybe you have a need to increase the filesystem size to install more software or perform tasks that require more space. Remember that increasing the partition size does not increase the filesystem size. With the partitions and filesystem organized on the disk as above, assuming that the system is used and needs to preserve data, you need to delete partition 2 first, and then increase the size of partition 1, for example from 10G to 12G. Run the commands below:

```
sudo parted /dev/sdb rm 2
sudo parted /dev/sdb resizepart 1 12288MiB
```

Then recreate partition 2, i.e. swap partition:

```
sudo parted /dev/sdb mkpart primary linux-swap 12288MiB 14336MiB
```

Now the root partition size has increased but the filesystem size has not changed

```
Terminal -
File Edit View Terminal Tabs Help
bash-5.1$ sudo parted /dev/sdb unit MiB print
Model: Kingston DataTraveler 3.0 (scsi)
Disk /dev/sdb: 14755MiB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start      End          Size         Type         File system  Flags
  1      1.00MiB    12288MiB     12287MiB     primary      ext4
  2      12288MiB   14336MiB     2048MiB      primary

bash-5.1$ sudo fsck /dev/sdb1
fsck from util-linux 2.37.2
e2fsck 1.46.2 (28-Feb-2021)
firewall: clean, 11/655360 files, 66753/2621184 blocks
bash-5.1$
```

To increase the filesystem size to the same as the partition size, run this command:

```
sudo resize2fs /dev/sdb1
```

```
Terminal -
File Edit View Terminal Tabs Help
bash-5.1$ sudo fsck /dev/sdb1
fsck from util-linux 2.37.2
e2fsck 1.46.2 (28-Feb-2021)
firewall: clean, 11/655360 files, 66753/2621184 blocks
bash-5.1$ sudo resize2fs /dev/sdb1
resize2fs 1.46.2 (28-Feb-2021)
Resizing the filesystem on /dev/sdb1 to 3145472 (4k) blocks.
The filesystem on /dev/sdb1 is now 3145472 (4k) blocks long.

bash-5.1$ sudo fsck /dev/sdb1
fsck from util-linux 2.37.2
e2fsck 1.46.2 (28-Feb-2021)
firewall: clean, 11/786432 files, 76004/3145472 blocks
bash-5.1$ sudo parted /dev/sdb unit MiB print
Model: Kingston DataTraveler 3.0 (scsi)
Disk /dev/sdb: 14755MiB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start      End          Size         Type         File system  Flags
  1      1.00MiB    12288MiB     12287MiB     primary      ext4
  2      12288MiB   14336MiB     2048MiB      primary

bash-5.1$
```

The last thing is to run `sudo mkswap /dev/sdb2` to set up the swap area on the new swap partition. Then update `/etc/fstab` if necessary. If the system is running, you need to run `sudo swapoff /dev/sdb2` before deleting the old swap partition. To enable the swapping, run `sudo swapon -a`

Creating disk image

For the above example we use 14336M in disk, you run the command below to create a raw disk image file named `disk-omarine-fw.raw`

```
sudo dd if=/dev/sdb of=disk-omarine-fw.raw bs=4M count=3584 conv=spars
```

14336 / 1024 = 14. Thus, the disk image size is exactly 14G.

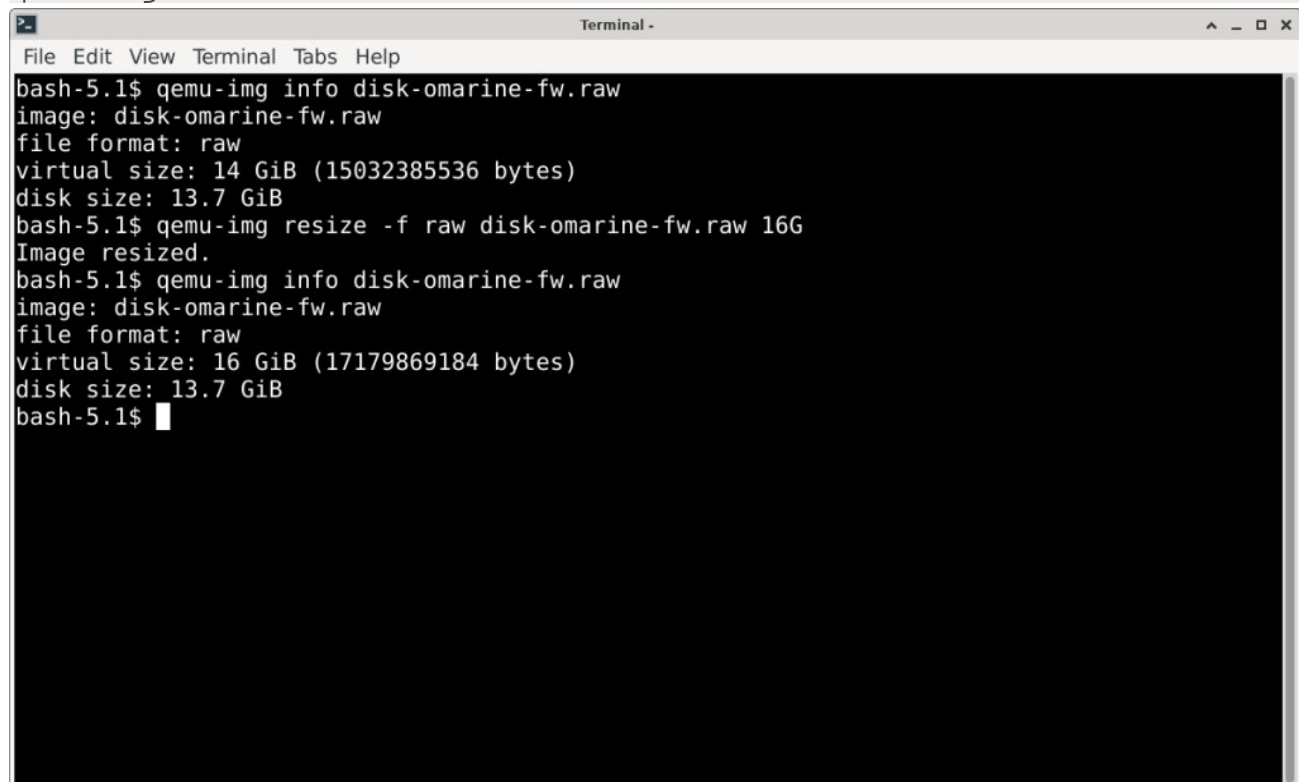
Increasing the disk image size

Once the disk image has been used, we can still resize it. However, this only makes sense when adding disk operations inside the virtual machine. To view info of the disk image `disk-omarine-fw.raw`, run this command:

```
qemu-img info disk-omarine-fw.raw
```

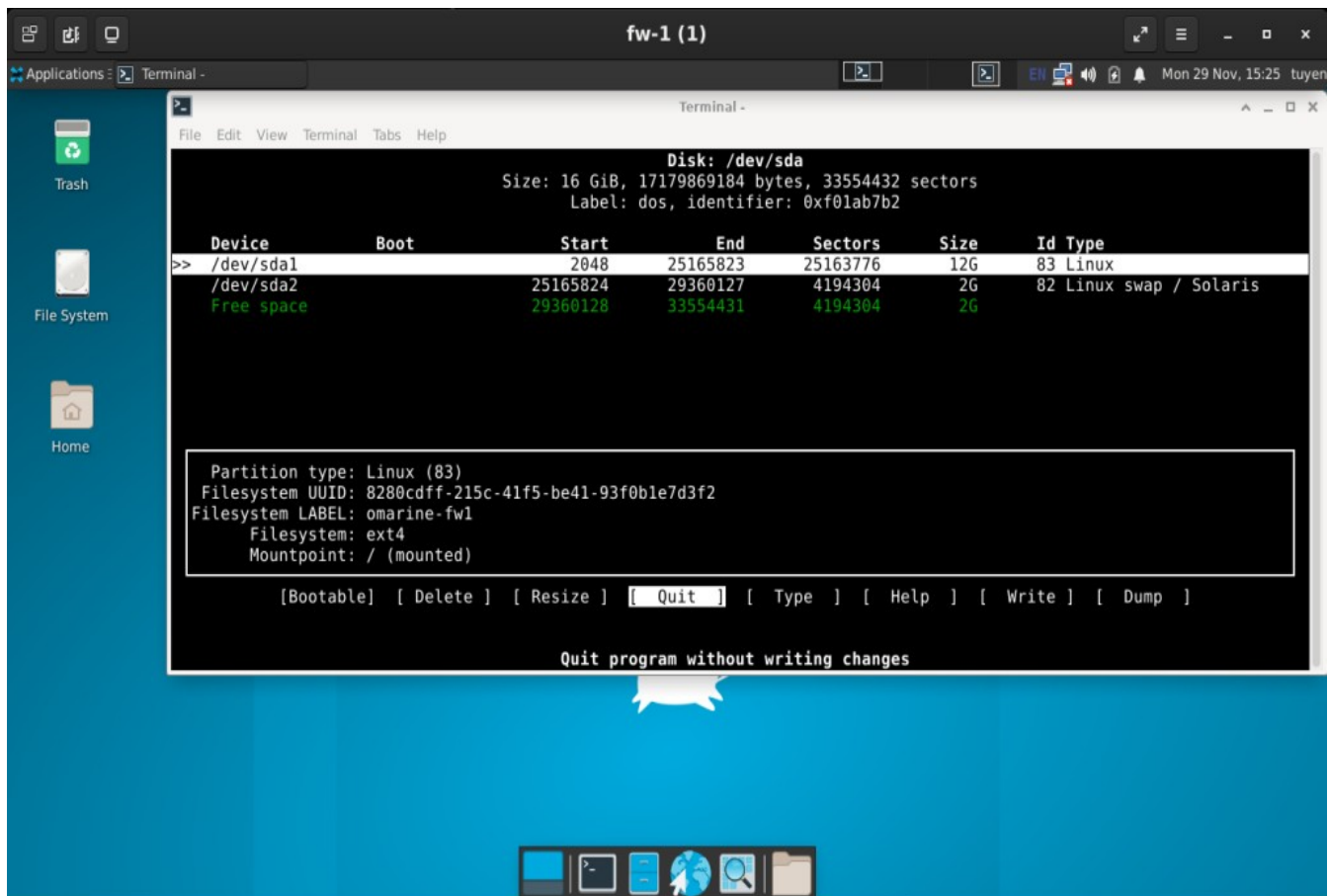
The command below increases the size of the disk image `disk-omarine-fw.raw` from 14G to 16G

```
qemu-img resize -f raw disk-omarine-fw.raw 16G
```

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
bash-5.1$ qemu-img info disk-omarine-fw.raw
image: disk-omarine-fw.raw
file format: raw
virtual size: 14 GiB (15032385536 bytes)
disk size: 13.7 GiB
bash-5.1$ qemu-img resize -f raw disk-omarine-fw.raw 16G
Image resized.
bash-5.1$ qemu-img info disk-omarine-fw.raw
image: disk-omarine-fw.raw
file format: raw
virtual size: 16 GiB (17179869184 bytes)
disk size: 13.7 GiB
bash-5.1$
```

The above command adds a 2G free space inside the virtual disk



You still have to reorganize the disk to take advantage of the free space, doing inside the virtual machine. Disk manipulation in a virtual machine is no different from that of a real machine.

4. Creating virtual machine with qemu

For secure purpose, do not run qemu as root. However, there are situations where the virtual machine creation process is forced to use root privileges. This should be controlled and considered empowering decision-making for the admin and the security policy.

When qemu runs the bridge helper program to create tap devices that require root privileges, which defaults to the binary /usr/libexec/qemu-bridge-helper, this binary can be run as root because it is set uid root

```
Terminal -
File Edit View Terminal Tabs Help
bash-5.1$ ls -l /usr/libexec/qemu-bridge-helper
-rwsr-x---. 1 root kvm 300360 Oct 26 10:24 /usr/libexec/qemu-bridge-helper*
bash-5.1$
bash-5.1$
bash-5.1$ ls -Z /usr/libexec/qemu-bridge-helper
system_u:object_r:virt_bridgehelper_exec_t:SystemLow /usr/libexec/qemu-bridge-helper*
bash-5.1$
bash-5.1$
bash-5.1$ cat /etc/qemu/bridge.conf
allow br0
bash-5.1$
bash-5.1$
bash-5.1$ sudo ls -Z /dev/net
system_u:object_r:tun_tap_device_t:SystemLow tun
bash-5.1$
```

The snapshot above shows that only users in the kvm group can use the helper program. Furthermore, the helper program can only run if the administrator configuring to allow access for the corresponding bridge devices in the /etc/qemu/bridge.conf configuration file.

It's not enough. In terms of security, the qemu-bridge-helper binary has the type virt_bridgehelper_exec_t. It will run in the virt_bridgehelper_t domain and is able to open /dev/net/tun only if the following rule is included in the security policy:

```
allow virt_bridgehelper_t tun_tap_device_t:chr_file rw_chr_file_perms;
```

The command below creates the virtual machine fw-1:

```
qemu -enable-kvm \
-m 4G \
-smp 2 \
-cpu host \
-machine q35,vmpport=off \
-netdev tap,id=n1,br=br0,ifname=net01,script=no,downscript=no \
-device virtio-net,netdev=n1,mac=12:34:56:78:9a:01 \
-netdev tap,id=n2,br=br1,ifname=net11,script=no,downscript=no \
-device virtio-net,netdev=n2,mac=12:34:56:78:9a:11 \
-netdev tap,id=n3,br=br2,ifname=net21,script=no,downscript=no \
-device virtio-net,netdev=n3,mac=12:34:56:78:9a:21 \
```

```

-drive file=disk-omarine-fw.raw,format=raw \
-name fw-1 \
-device intel-hda -device hda-duplex \
-display none \
-vga qxl \
-spice port=3001,disable-ticketing=on \
-device virtio-serial \
-chardev spicevmc,id=vdagent,debug=0,name=vdagent \
-device virtserialport,chardev=vdagent,name=com.redhat.spice.0 \
-device qemu-xhci,multifunction=on \
-chardev spicevmc,name=usbredir,id=usbredirchardev1 \
-device usb-redir,chardev=usbredirchardev1,port=1 \
-chardev spicevmc,name=usbredir,id=usbredirchardev2 \
-device usb-redir,chardev=usbredirchardev2,port=2 \
-chardev spicevmc,name=usbredir,id=usbredirchardev3 \
-device usb-redir,chardev=usbredirchardev3,port=3 \

```

Explanation of options:

- `-enable-kvm`: Use KVM full virtualization support
- `-m 4G`: Set RAM size to 4G
- `-smp 2`: SMP system with 2 CPUs
- `-cpu host`: Use CPU model as host machine
- `-machine q35,vmport=off`: The emulated machine type is q35 – the newest machine type in QEMU. Disable VMWare IO port, this is necessary for the mouse to work properly with spice
- `-netdev tap,id=n1,br=br0,ifname=net01,script=no,downscript=no`: Configure the tap network backend with id n1, bridge br0. Don't use bridge helper because we have created the tap interface net01. Once the tap interface has been created, normal users can access `/dev/net/tun` without requiring root privileges.
- `-device virtio-net,netdev=n1,mac=12:34:56:78:9a:01`: Define a nic whose type virtio-net for the guest (virtual machine) connecting to the host's tap interface which corresponds to the tap network backend whose id n1. The mac address is needed for the DHCP server to automatically assign an IP address to the virtual machine. The next two nics are configured in a similar way
- `-drive file=disk-omarine-fw.raw,format=raw`: The virtual machine uses the disk image disk-omarine-fw.raw, raw format
- `-name fw-1`: The name of the guest is fw-1. This name is shown on the virtual machine window caption, not the machine name on the network.
- `-device intel-hda -device hda-duplex`: Configure Intel HD Audio sound device
- `-display none`: Do not display the virtual machine at the host machine. The user will connect to and display the remote virtual machine at the client later
- `-vga qxl`: Use the qxl graphics card. The virtual machine needs to equip the qxl video driver and the X

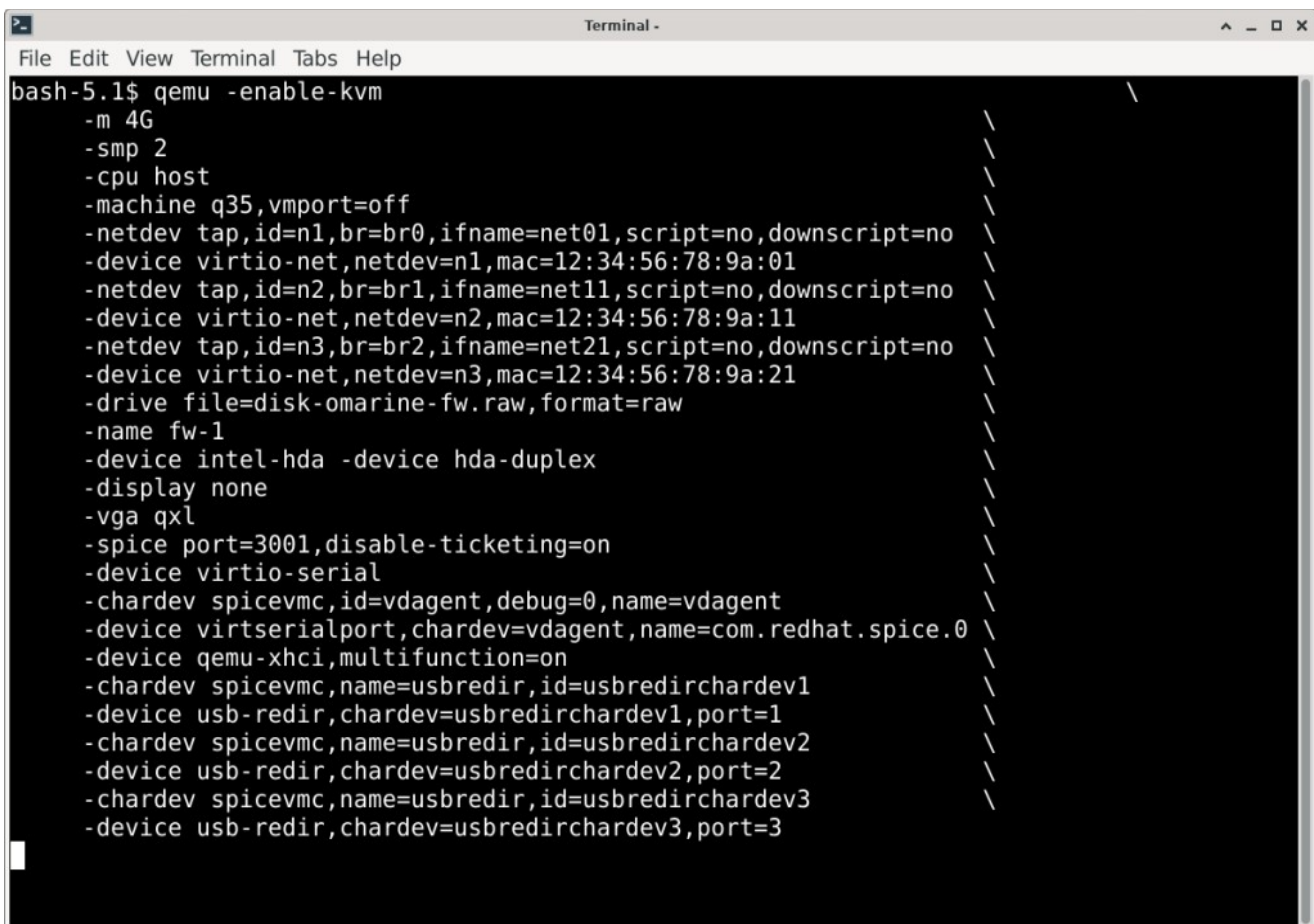
configuration to use it. This is the best graphics card when going with spice protocol

- -spice port=3001,disable-ticketing=on: Communicate with the client via port 3001. To simplify the example, do not authenticate the client.

- -device virtio-serial -chardev spicevmc,id=vdagent,debug=0,name=vdagent -device virtserialport,chardev=vdagent,name=com.redhat.spice.0: spicevmc character device backend is used by virtio serial port in the spice agent's communication on virtual machine. The default port is /dev/virtio-ports/com.redhat.spice.0. This port can be set when running spice agent. Usually, mouse's motion and response on virtual machines are not smooth. With spice agent, the mouse works in the client mode, its motion and response is completely smooth. There is no need to capture the mouse when hovering the mouse over the virtual machine window on the client. It also supports copy and paste between virtual machine and client, drag and drop to transfer files.

- -device qemu-xhci,multifunction=on -chardev spicevmc,name=usbredir,id=usbredirchardev1 -device usb-redir,chardev=usbredirchardev1,port=1...: Define three redirected USB ports, supports USB 3.0, thanks to the spicevmc character device backend

The command to create virtual machine above can be considered successful if nothing is output behind the command



```
bash-5.1$ qemu -enable-kvm
-m 4G
-smp 2
-cpu host
-machine q35,vmport=off
-netdev tap,id=n1,br=br0,ifname=net01,script=no,downscript=no
-device virtio-net,netdev=n1,mac=12:34:56:78:9a:01
-netdev tap,id=n2,br=br1,ifname=net11,script=no,downscript=no
-device virtio-net,netdev=n2,mac=12:34:56:78:9a:11
-netdev tap,id=n3,br=br2,ifname=net21,script=no,downscript=no
-device virtio-net,netdev=n3,mac=12:34:56:78:9a:21
-drive file=disk-omarine-fw.raw,format=raw
-name fw-1
-device intel-hda -device hda-duplex
-display none
-vga qxl
-spice port=3001,disable-ticketing=on
-device virtio-serial
-chardev spicevmc,id=vdagent,debug=0,name=vdagent
-device virtserialport,chardev=vdagent,name=com.redhat.spice.0
-device qemu-xhci,multifunction=on
-chardev spicevmc,name=usbredir,id=usbredirchardev1
-device usb-redir,chardev=usbredirchardev1,port=1
-chardev spicevmc,name=usbredir,id=usbredirchardev2
-device usb-redir,chardev=usbredirchardev2,port=2
-chardev spicevmc,name=usbredir,id=usbredirchardev3
-device usb-redir,chardev=usbredirchardev3,port=3
```


5. Accessing remote virtual machine

We are working on a remote physical machine named ngoc at 192.168.0.12. This machine acts as a client to access virtual machine fw-1 on host machine omarine at 192.168.0.3. The communication port is 3001, which we configured when setting up the fw-1 virtual machine in the previous chapter.

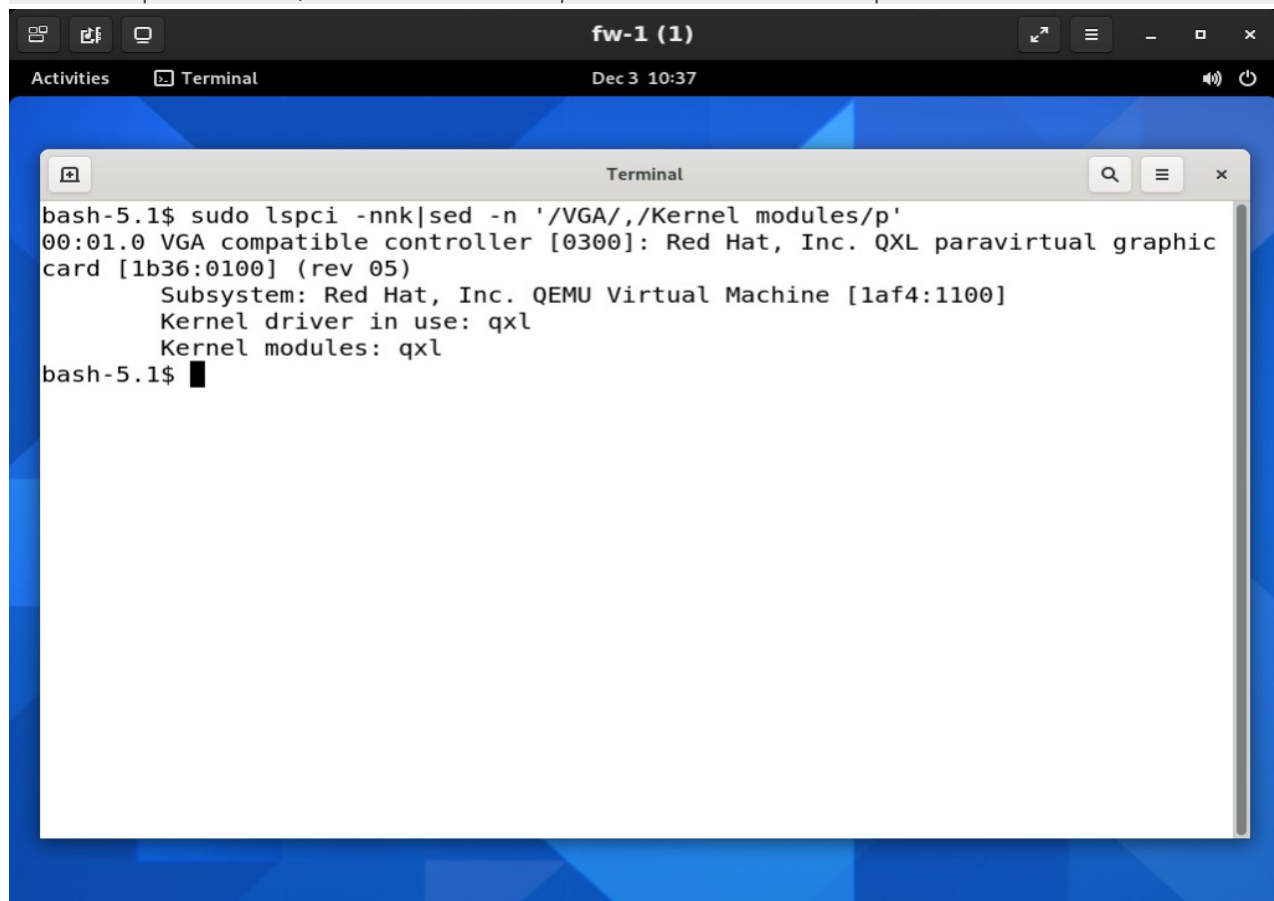
The access command is as follows:

```
remote-viewer spice://omarine.omarine.co:3001
```

6. Configuring X to use qxl video driver

Now we work on the fw-1 virtual machine. First let's try to see with the `-vga qxl` definition, which video driver qemu will give us in the kernel. You run this command:

```
sudo lspci -nnk|sed -n '/VGA/,/Kernel modules/p'
```



Oh so the qxl video driver is available. The remaining is X must handle kernel modesetting. This is done by X qxl driver. But we have to tell it. You configure as below:

```

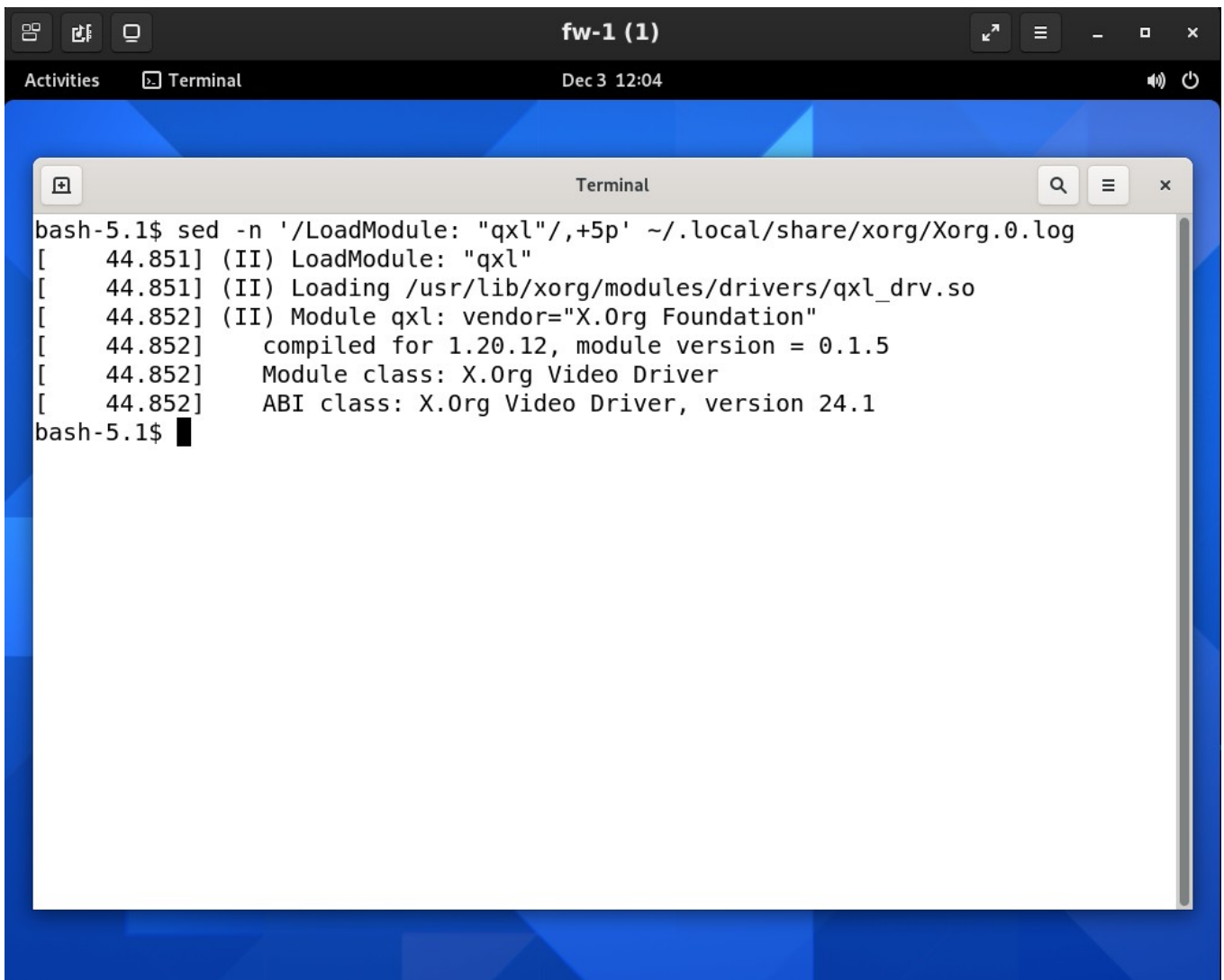
sudo tee /etc/X11/xorg.conf.d/10-qxl.conf << EOF
Section "ServerLayout"
    Identifier      "Default Layout"
    Screen          0  "Screen0" 0 0
EndSection
Section "Device"
    Identifier      "Videocard0"
    Driver          "qxl"
EndSection
Section "Screen"
    Identifier      "Screen0"
    Device          "Videocard0"
    DefaultDepth     24
    SubSection      "Display"
        Viewport     0 0
        Depth        24
        Modes         "1360x768"
    EndSubSection
EndSection
EOF

```

The default video mode is 1360×768. You can replace with other mode as needed in the list of supported video modes.

The following command extracts six lines in the X server's log at while load X qxl video driver:

```
sed -n '/LoadModule: "qxl"/,+5p' ~/.local/share/xorg/Xorg.0.log
```



The screenshot shows a terminal window titled "fw-1 (1)" with a top bar indicating "Activities", "Terminal", and the date/time "Dec 3 12:04". The terminal content shows a user running the command `sed -n '/LoadModule: "qxl"/,+5p' ~/.local/share/xorg/Xorg.0.log`. The output displays the loading details for the `qxl` module, including its path, vendor information, compilation details, and class/ABI information.

```
bash-5.1$ sed -n '/LoadModule: "qxl"/,+5p' ~/.local/share/xorg/Xorg.0.log
[  44.851] (II) LoadModule: "qxl"
[  44.851] (II) Loading /usr/lib/xorg/modules/drivers/qxl_drv.so
[  44.852] (II) Module qxl: vendor="X.Org Foundation"
[  44.852]      compiled for 1.20.12, module version = 0.1.5
[  44.852]      Module class: X.Org Video Driver
[  44.852]      ABI class: X.Org Video Driver, version 24.1
bash-5.1$
```

7. Network configuration using systemd

Almost any network configuration can be done using commands in the `iproute2` package. For example, you can run the command below to rename the `enp0s2` interface to `eth0`:

```
sudo ip link set enp0s2 name eth0
```

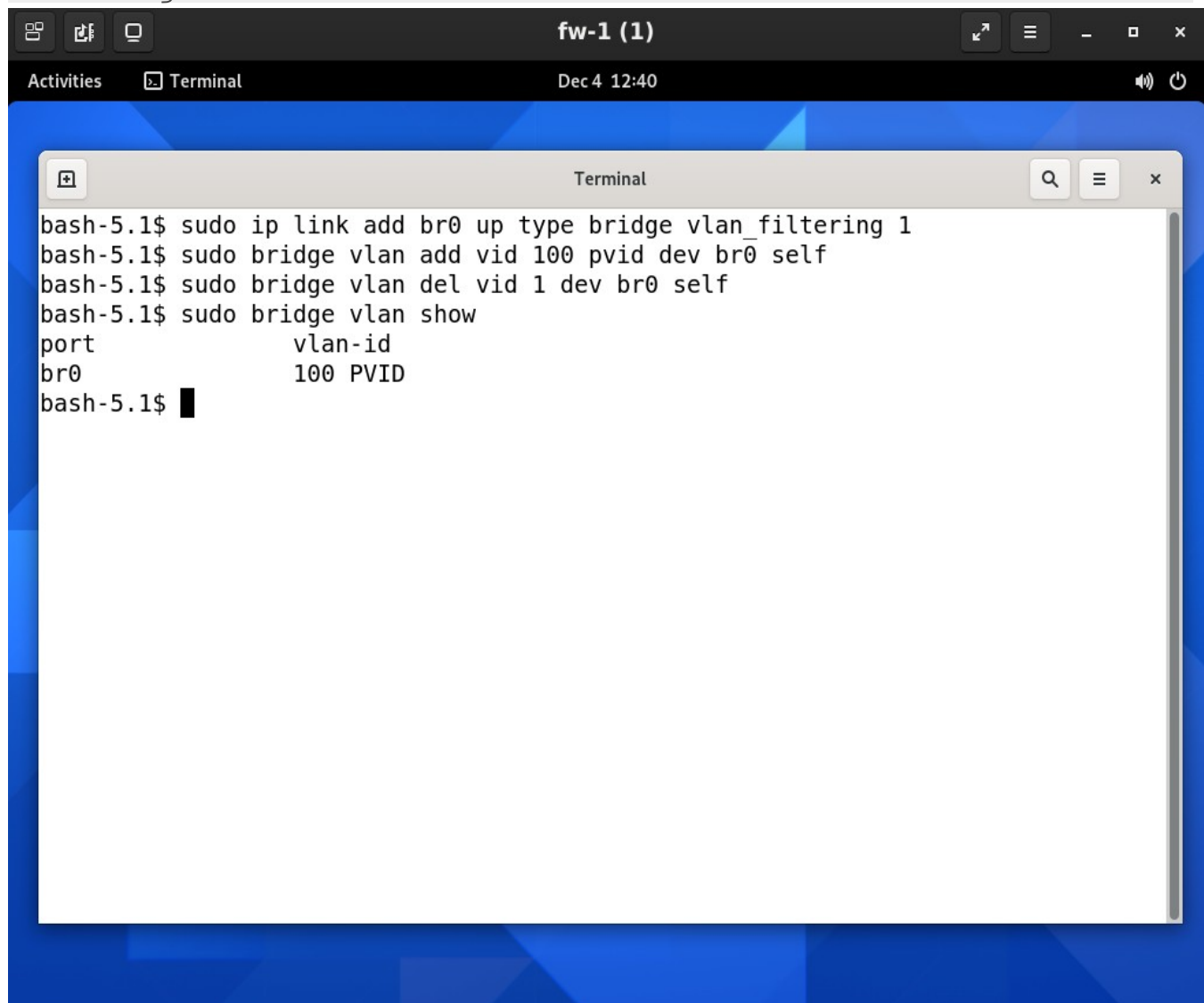
This can be done automatically at machine startup by running a service with file contents like this:

```
[Unit]
Description=Naming interface
After=network.target
```

```
[Service]
Type=oneshot
ExecStart=/usr/sbin/ip link set enp0s2 name eth0
RemainAfterExit=yes
[Install]
WantedBy=multi-user.target
```

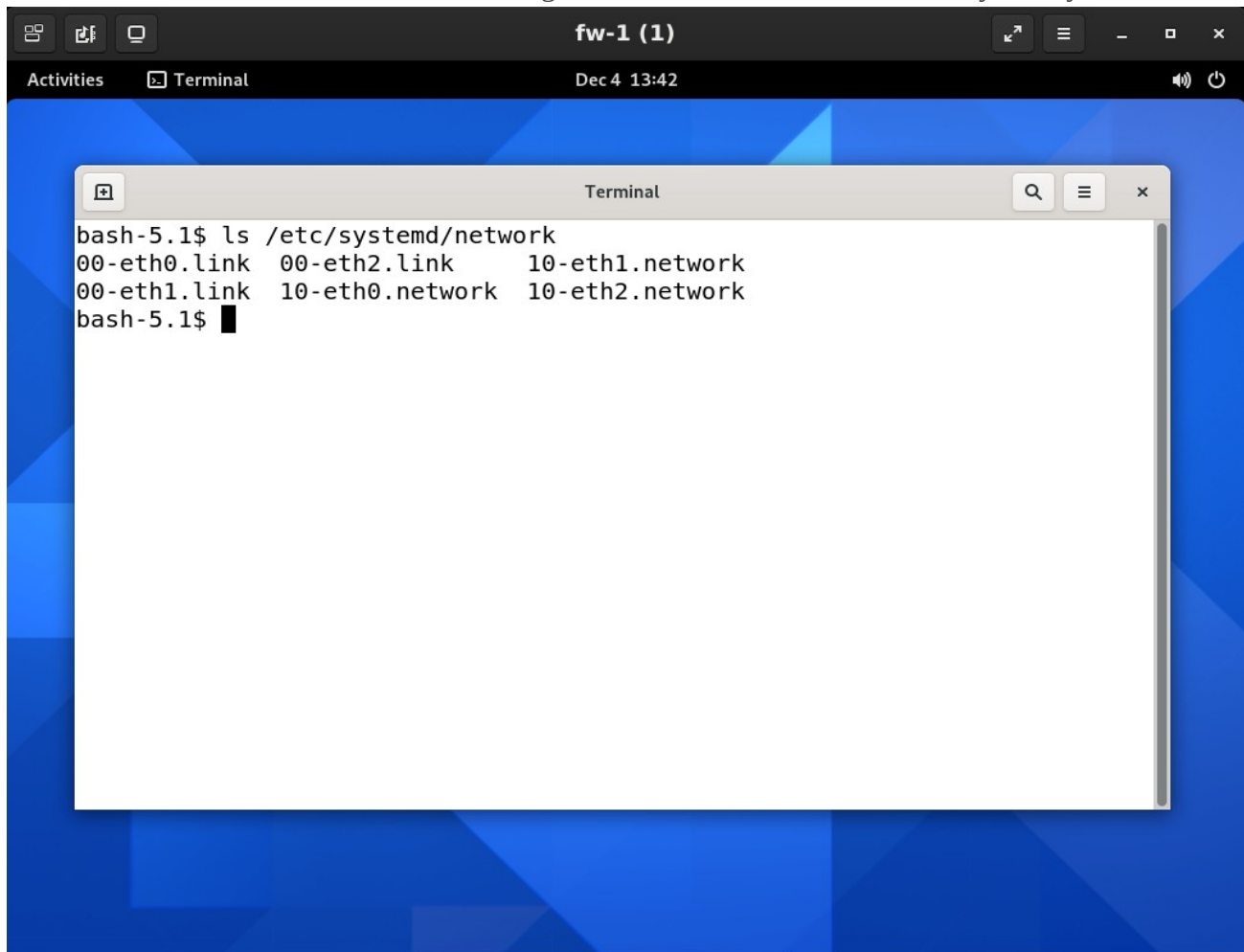
The commands below set up bridge br0 and display the vlan configuration information. This bridge can join vlan 100 with traffic including only tagged packets

```
sudo ip link add br0 up type bridge vlan_filtering 1
sudo bridge vlan add vid 100 pvid dev br0 self
sudo bridge vlan del vid 1 dev br0 self
sudo bridge vlan show
```

A screenshot of a Linux desktop environment. At the top, a terminal window titled 'fw-1 (1)' is visible. Below it, a larger terminal window titled 'Terminal' is open, showing a series of commands and their output. The commands are: 'sudo ip link add br0 up type bridge vlan_filtering 1', 'sudo bridge vlan add vid 100 pvid dev br0 self', 'sudo bridge vlan del vid 1 dev br0 self', and 'sudo bridge vlan show'. The output of the last command shows a table with two columns: 'port' and 'vlan-id'. The first row shows 'br0' and '100 PVID'. The terminal window has a search icon, a menu icon, and a close icon in the top right corner. The desktop background is blue with a geometric pattern.

```
bash-5.1$ sudo ip link add br0 up type bridge vlan_filtering 1
bash-5.1$ sudo bridge vlan add vid 100 pvid dev br0 self
bash-5.1$ sudo bridge vlan del vid 1 dev br0 self
bash-5.1$ sudo bridge vlan show
port          vlan-id
br0           100 PVID
bash-5.1$
```

However, a convenient way is to use systemd. The firewall machine needs three network interfaces with different functions. There are six configuration files located in the directory /etc/systemd/network

A screenshot of a Linux desktop environment. The background is a blue geometric pattern. At the top, there is a taskbar with icons for Activities, Terminal, and system status. The main window is a terminal titled "fw-1 (1)". Inside the terminal, a smaller window titled "Terminal" is open, showing the command `ls /etc/systemd/network` and its output: `00-eth0.link 00-eth2.link 10-eth1.network` and `00-eth1.link 10-eth0.network 10-eth2.network`. The prompt is `bash-5.1$`.

The contents of the file 00-eth0.link are as follows:

```
[Match]
MACAddress=12:34:56:78:9a:01
[Link]
Name=eth0
```

Normally ethernet interfaces have names such as `enp0s2`. The above configuration file talks to systemd that we want the first interface with the mac address `12:34:56:78:9a:01` to be assigned the name `eth0`. The same goes for the files `00-eth1.link` and `00-eth2.link`. As a result we have interfaces `eth0`, `eth1` and `eth2`. That makes it suitable for the design scheme and uniform in the tasks related to network interface names such as firewall work, HA system configuration, packet filtering tracking system configuration, network address translation. So all network interfaces need to define mac address when creating the virtual machine.

The contents of the file `10-eth0.network` are as follows:

```
Match]
Name=eth0
[Network]
DHCP=yes
```

The interface eth0 uses the DHCP service to assign IP address automatically.

The contents of the file 10-eth1.network are as follows:

```
[Match]
Name=eth1
[Network]
Address=192.168.2.1/24
```

The interface eth1 is assigned the static address 192.168.2.1. This interface is designed to be located at the default gateway (but not always) of the 192.168.2.0/24 network, ie that of the client. The term “client” here is in the context of the firewall system, not to be confused with client accessing remote virtual machine in the host-guest-client relationship.

The file 10-eth2.network is similar to the file 10-eth1.network:

```
[Match]
Name=eth2
[Network]
Address=192.168.100.100/24
```

The interface eth2 is used to synchronize information between two firewalls.

8. Testing spice agent and USB redirection

To see the effect of spice agent and USB redirection we practice the tasks like this:

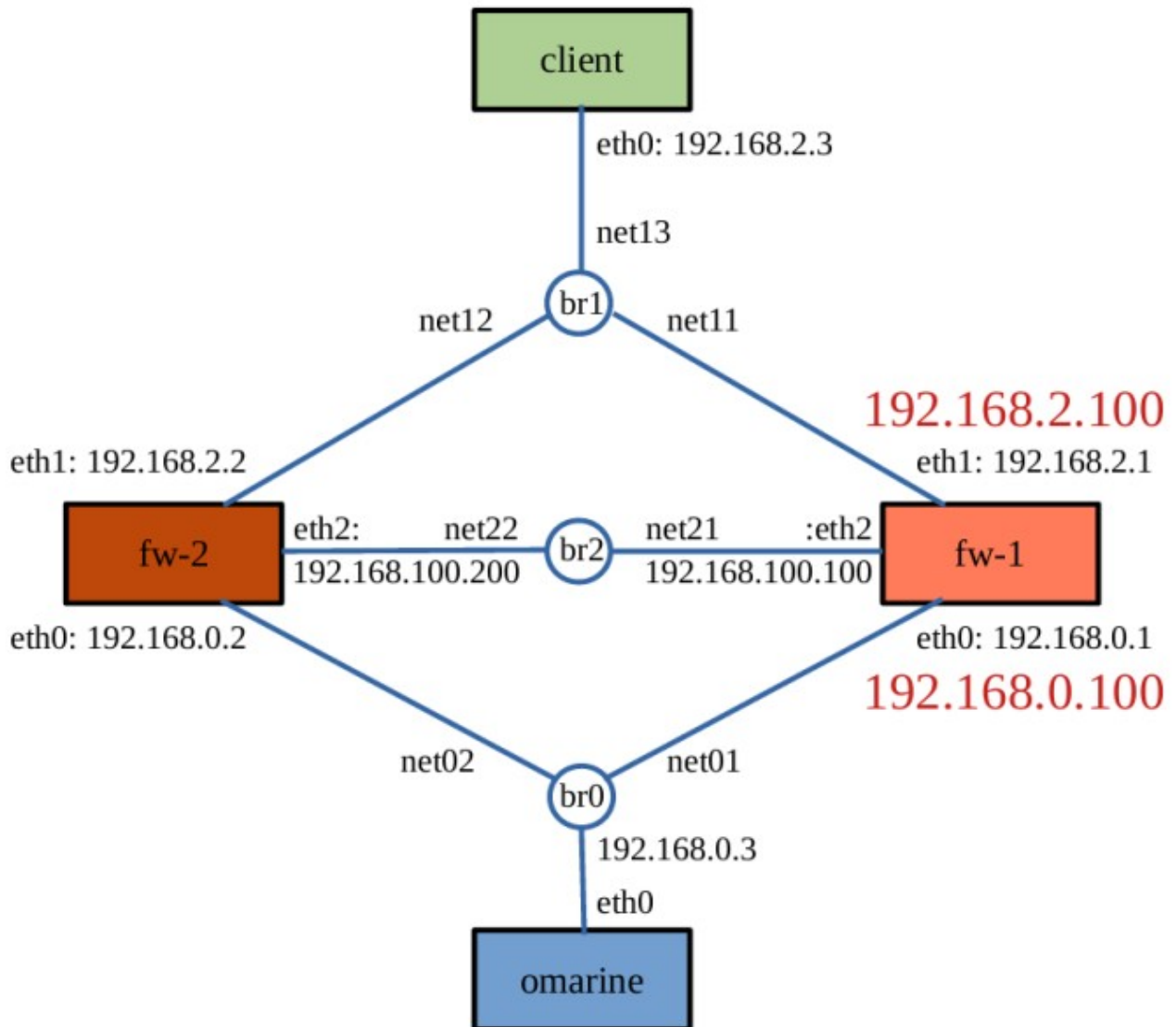
1. Perform drag and drop to transfer a file from the client to the virtual machine
2. Copy a text from the client and paste it into gedit's editing area on the virtual machine
3. While working on the virtual machine, plug a USB stick whose label DUMMY into the client machine. The label DUMMY then appears in the virtual machine's file browser as a new drive. Click it to open the USB drive

We also see no mouse capture and no need to release the mouse when switching between the client and the virtual machine.

9. Routing

Routing is an interesting and important issue. Not only smoothing traffic, routing work also determines the path of a packet so that the correct packet filtering can be performed.

We start from the client. There are two ways out from the client, so which way to go?



192.168.2.1 on the firewall machine fw-1 cannot be set as the client's default gateway, as such the firewall machine fw-2 is completely disabled. Likewise, the firewall machine fw-2's 192.168.2.2 cannot be used. The solution is to use the virtual IP address 192.168.2.100 generated by the HA system. From the point of view of the HA system, all four machines above are real machines. It generates virtual IP addresses for automatic routing. If the firewall machine fw-1 is the primary firewall, the address 192.168.2.100 will be added to fw-1's eth1 interface.

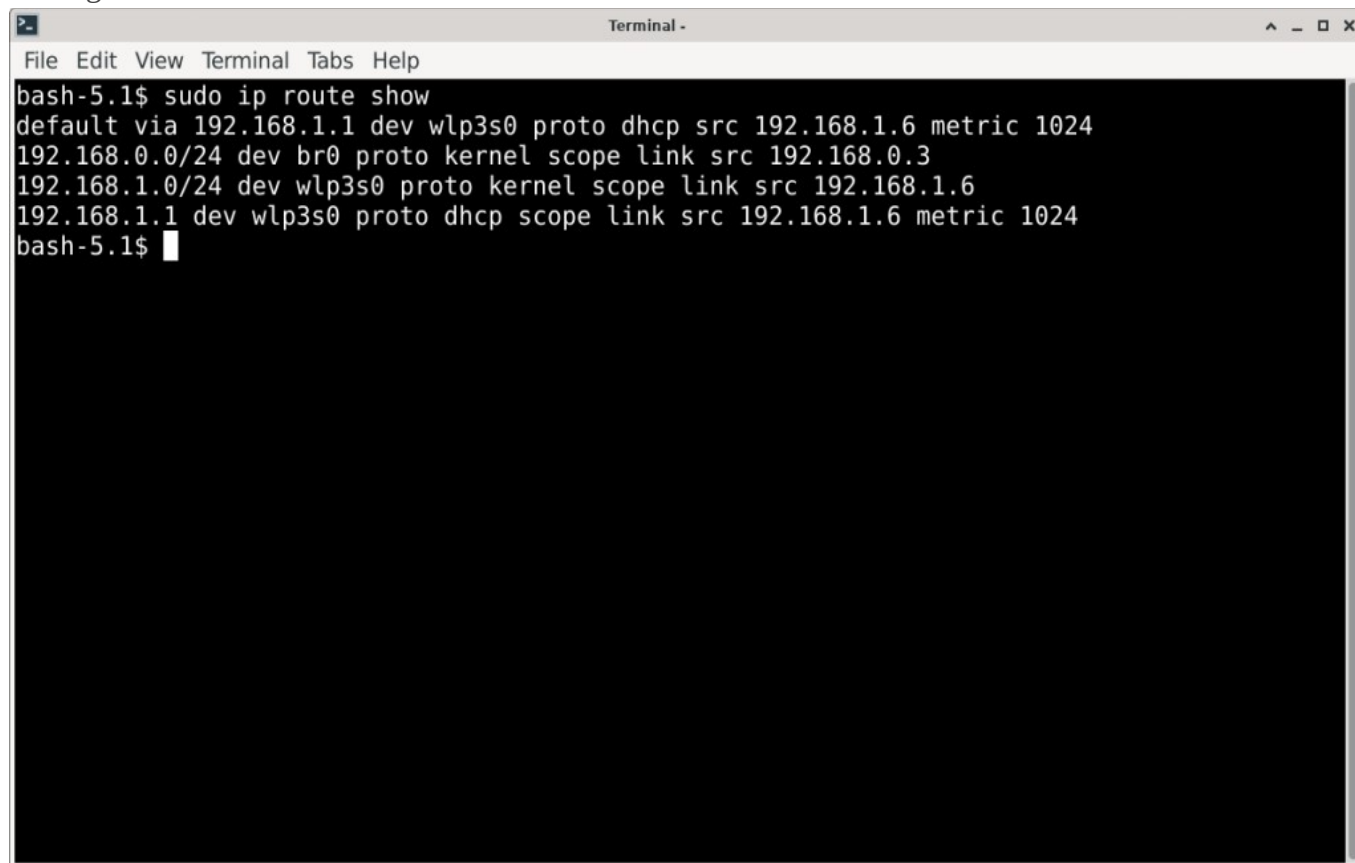
So the client needs to create a route through the default gateway 192.168.2.100. You run this command:

```
sudo ip route add default proto static via 192.168.2.100 dev eth0
```

Traffic was clear. If the firewall fw-1 is down, the address 192.168.2.100 is assigned to the firewall fw-2's eth1 interface and the client always has a single logical route to get to the service it needs.

Network address translation

But the above only addresses the route in the client network, which is 192.168.2.0/24. Consider the routing table of the omarine server

A terminal window titled "Terminal -" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the output of the command "sudo ip route show". The output lists four routes: a default route via 192.168.1.1 on wlp3s0, and three local routes for 192.168.0.0/24, 192.168.1.0/24, and 192.168.1.1 on wlp3s0. The prompt "bash-5.1\$" is visible at the end of the output.

```
bash-5.1$ sudo ip route show
default via 192.168.1.1 dev wlp3s0 proto dhcp src 192.168.1.6 metric 1024
192.168.0.0/24 dev br0 proto kernel scope link src 192.168.0.3
192.168.1.0/24 dev wlp3s0 proto kernel scope link src 192.168.1.6
192.168.1.1 dev wlp3s0 proto dhcp scope link src 192.168.1.6 metric 1024
bash-5.1$
```

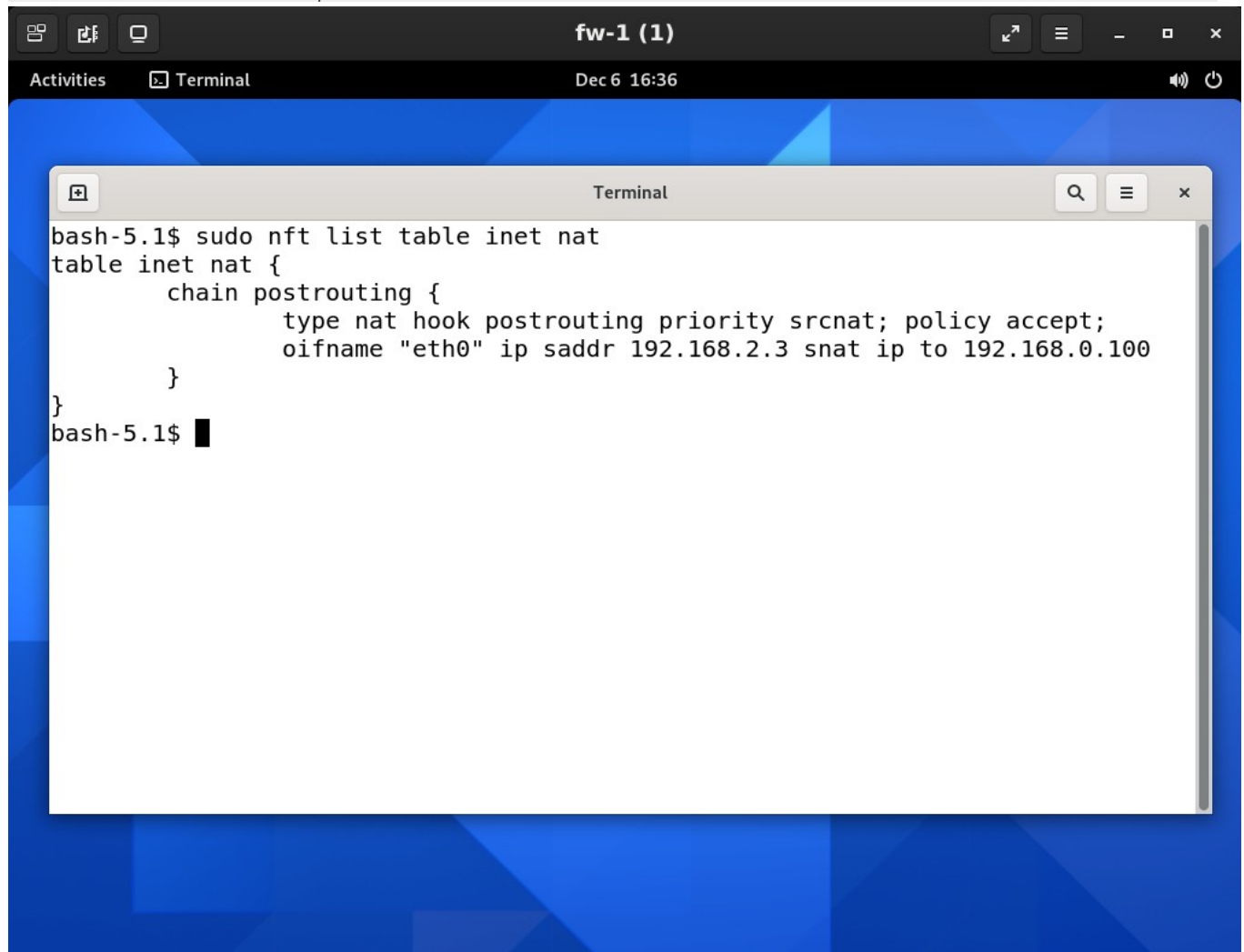
Apparently the client network 192.168.2.0/24 is not present in the table. A connection originates from the client, when it reaches the omarine server it has no way back in bidirectional client/server negotiation. To fix it, we add a route as follows:

```
sudo ip route add 192.168.2.0/24 via 192.168.0.100 dev br0
```

Another approach is network address translation. Network address translation or NAT is a familiar concept. However, its application is situation-specific. NAT's job is simply to handle layer 3/4 protocol headers to modify the source/destination address usually for the purpose of making it possible for the packet to reach its destination. The commands below create the table nat on the firewall machine, add the chain postrouting to the table, and then handle packet going from the client to the omarine with a

rule that changes the source address 192.168.2.3 to 192.168.0.100 as it goes out of the firewall from the eth0 interface

```
sudo nft add table inet nat
sudo nft add chain inet nat postrouting { type nat hook postrouting
priority srcnat \; }
sudo nft add rule inet nat postrouting oifname eth0 ip saddr
192.168.2.3 snat ip to 192.168.0.100
```

A screenshot of a Linux desktop environment with a blue geometric background. A terminal window titled "fw-1 (1)" is open, showing the command prompt "bash-5.1\$". The user has entered the command "sudo nft list table inet nat". The terminal output shows the configuration of the "inet nat" table, including a "chain postrouting" with a "type nat hook postrouting priority srcnat; policy accept;" and a rule "oifname 'eth0' ip saddr 192.168.2.3 snat ip to 192.168.0.100". The prompt "bash-5.1\$" is followed by a cursor. The terminal window has a search icon, a menu icon, and a close button in the top right corner. The desktop taskbar at the top shows "Activities", "Terminal", and the date/time "Dec 6 16:36".

The corresponding packet on the way back will return 192.168.0.100 instead of 192.168.2.3. The new destination 192.168.0.100 is obviously transparent to omarine's routing table. At the firewall's eth0 station, the NAT translates this destination again to 192.168.2.3 for the packet to return to the client. Again, the entrance to the firewall system of a packet on the way back is the virtual IP address 192.168.0.100 and not the actual IP address of the eth0 interface of the firewall machines. It is a single, logical path through a road-fork firewall system that automatically chooses the route.

10. Configuring HA and conntrackd

Once the network topology has been established, configuring the HA and conntrackd becomes simple.

HA Configuration

Keepalived uses VRRP (Virtual Router Redundancy Protocol) protocol to provide HA system. We use the sample configuration file keepalived.conf in the doc/sync directory of the conntrack-tools package, copy it to the /etc/keepalived directory and modify the parameters accordingly. The actual configuration file is as follows:

```
#
# Simple script for primary-backup setups
#
vrrp_sync_group G1 {    # must be before vrrp_instance declaration
    group {
        VI_1
        VI_2
    }
    notify_master "/etc/conntrackd/primary-backup.sh primary"
    notify_backup "/etc/conntrackd/primary-backup.sh backup"
    notify_fault "/etc/conntrackd/primary-backup.sh fault"
}
vrrp_instance VI_1 {
    interface eth1
    state SLAVE
    virtual_router_id 61
    priority 80
    advert_int 3
    authentication {
        auth_type PASS
        auth_pass papas_con_tomate
    }
    virtual_ipaddress {
        192.168.2.100    # default CIDR mask is /32
    }
}
vrrp_instance VI_2 {
    interface eth0
    state SLAVE
    virtual_router_id 62
```

```

priority 80
advert_int 3
authentication {
    auth_type PASS
    auth_pass papas_con_tomate
}
virtual_ipaddress {
    192.168.0.100
}
}

```

First define a VRRP synchronization group named G1. This group has two members, VI_1 and VI_2, which are two VRRP instances. The instance VI_1 runs on the interface eth1 with virtual IP address 192.168.2.100 and the instance VI_2 runs on the interface eth0 with the virtual IP address 192.168.0.100 as shown in the detailed instance configurations. primary-backup.sh is a script file also located in the doc/sync directory, we copy it to the /etc/conntrackd directory. The notify_master, notify_backup, notify_fault declarations are for the keepalived to notify conntrackd to take actions in the script when the firewalls transition to primary, backup or fault, respectively.

When a firewall becomes primary, it recovers the connection in terms of packet filtering based on its previous backup property. For the backup situation, the backup firewall will require re-syncing with the primary firewall. And when a firewall fails, it can also clean up garbage when receiving notification. That is the capability of Keepalived. As a result, the firewall system works seamlessly with the HA system to perform the role of a fault-tolerant firewall system.

Configuring conntrackd

The configuration file is conntrackd.conf located in the /etc/conntrackd directory. The file contents are as follows:

```

Sync {
    Mode FTFW {
        ResendQueueSize 131072
        PurgeTimeout 60
        ACKWindowSize 300
        DisableExternalCache Off
    }
    UDP {
        IPv4_address 192.168.100.100
        IPv4_Destination_Address 192.168.100.200
        Port 3780
        Interface eth2
        SndSocketBuffer 1249280
    }
}

```

```

RcvSocketBuffer 1249280
    Checksum on
}
Options {
    TCPWindowTracking Off
    # ExpectationSync On
}
}
General {
    Systemd on
    HashSize 32768
    HashLimit 131072
    # The default logfile is /var/log/contrackd.log
    LogFile on
    LockFile /var/lock/contrack.lock
    UNIX {
        Path /var/run/contrackdctl
    }
    NetlinkBufferSize 2097152
    NetlinkBufferSizeMaxGrowth 8388608
    NetlinkOverrunResync On
    NetlinkEventsReliable Off
    # PollSecs 15
    EventIterationLimit 100
    Filter From Kernelspace {
        Protocol Accept {
            TCP
            SCTP
            DCCP
            # UDP
            # ICMP
            # IPv6-ICMP
        }
        Address Ignore {
            IPv4_address 127.0.0.1 # loopback
            IPv4_address 192.168.2.100 # virtual IP 1
            IPv4_address 192.168.0.100 # virtual IP 2
            IPv4_address 192.168.2.1
            IPv4_address 192.168.0.1

```

```

    IPv4_address 192.168.100.100 # dedicated link ip
    IPv6_address ::1
}
# State Accept {
#     ESTABLISHED CLOSED TIME_WAIT CLOSE_WAIT for TCP
# }
}
}

```

Some parameters use default values. There are a few points worth noting:

- conntrackd synchronizes firewalls in FTFW (Fault Tolerant Firewall) mode. In this mode conntrackd performs message tracking so this is a reliable synchronization mode.
- Two firewalls use UDP protocol for synchronous communication, using interface eth2 as dedicated link with address 192.168.100.100 for the firewall fw-1 and 192.168.100.200 for the firewall fw-2.
- Conntrackd has user-configurable event filtering function to monitor and synchronize tracking states for certain traffic flows. There are three types of filtering: by protocol, by IP address, and by flow state.
- The above configuration file selects to filter the event messages from kernel space instead of user space. This reduces CPU consumption as there is no need to copy event messages from kernel space to user space.
- Use only three layer 4 protocols: TCP, SCTP and DCCP. That doesn't mean other protocols are left floating. conntrackd is a userspace tool that provides a connection recovery utility from failure, it does not degrade the functionality of the kernel that comes with the ruleset in the firewall role to protect network.
- Ignore local addresses because packets are only forwarded through the firewall.

Integrating conntrackd into the system

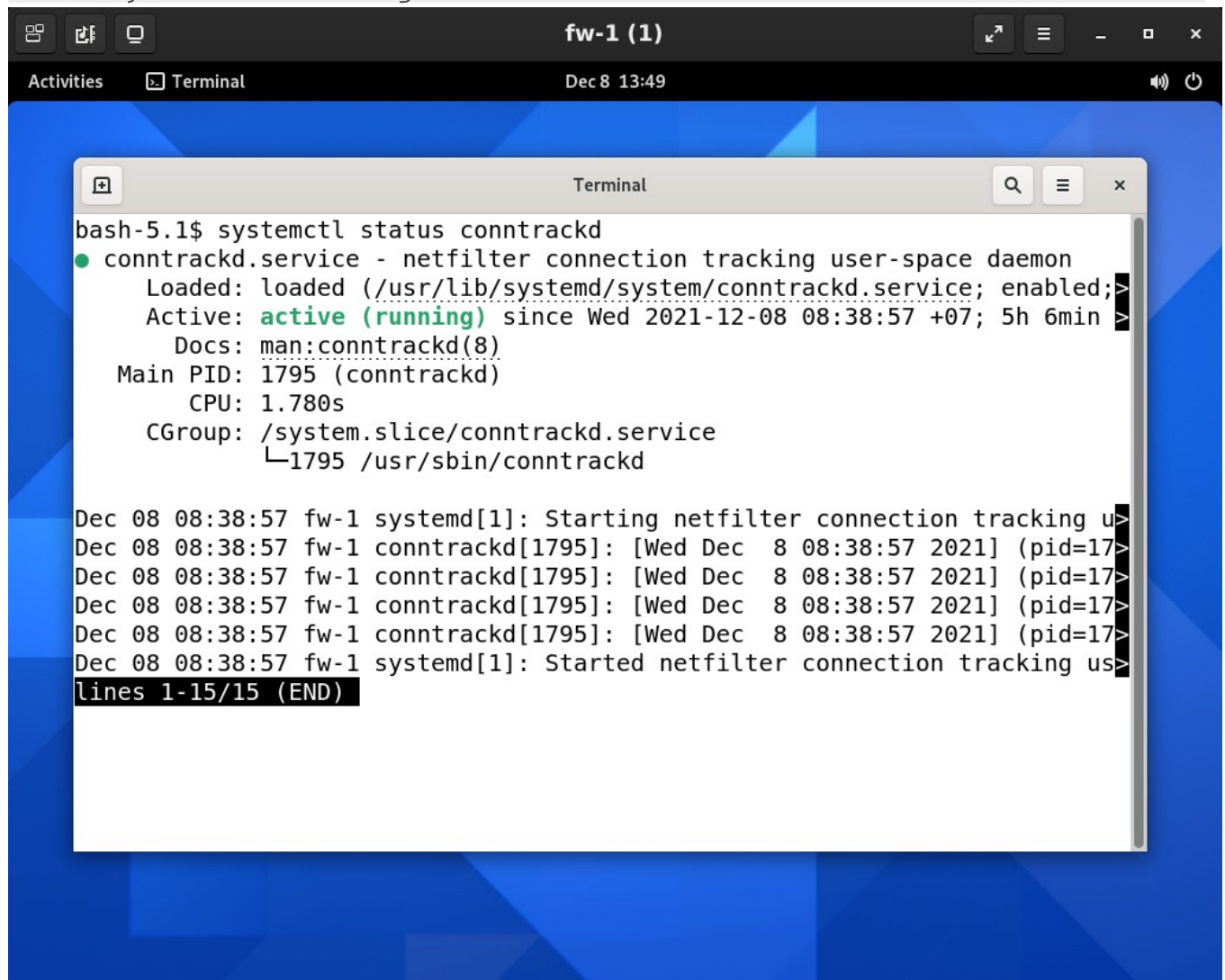
The integration of conntrackd into the system is done by running the conntrackd.service which has the following file contents:

```

[Unit]
Description=netfilter connection tracking user-space daemon
After=network-online.target
Wants=network-online.target
Documentation=man:conntrackd(8)
[Service]
Type=notify
NotifyAccess=all
KillMode=control-group
# ExecStartPre=/usr/sbin/nfct add helper ftp inet tcp
ExecStart=/usr/sbin/conntrackd

```

```
ExecStop=/usr/sbin/contrackd -k  
[Install]  
WantedBy=multi-user.target
```



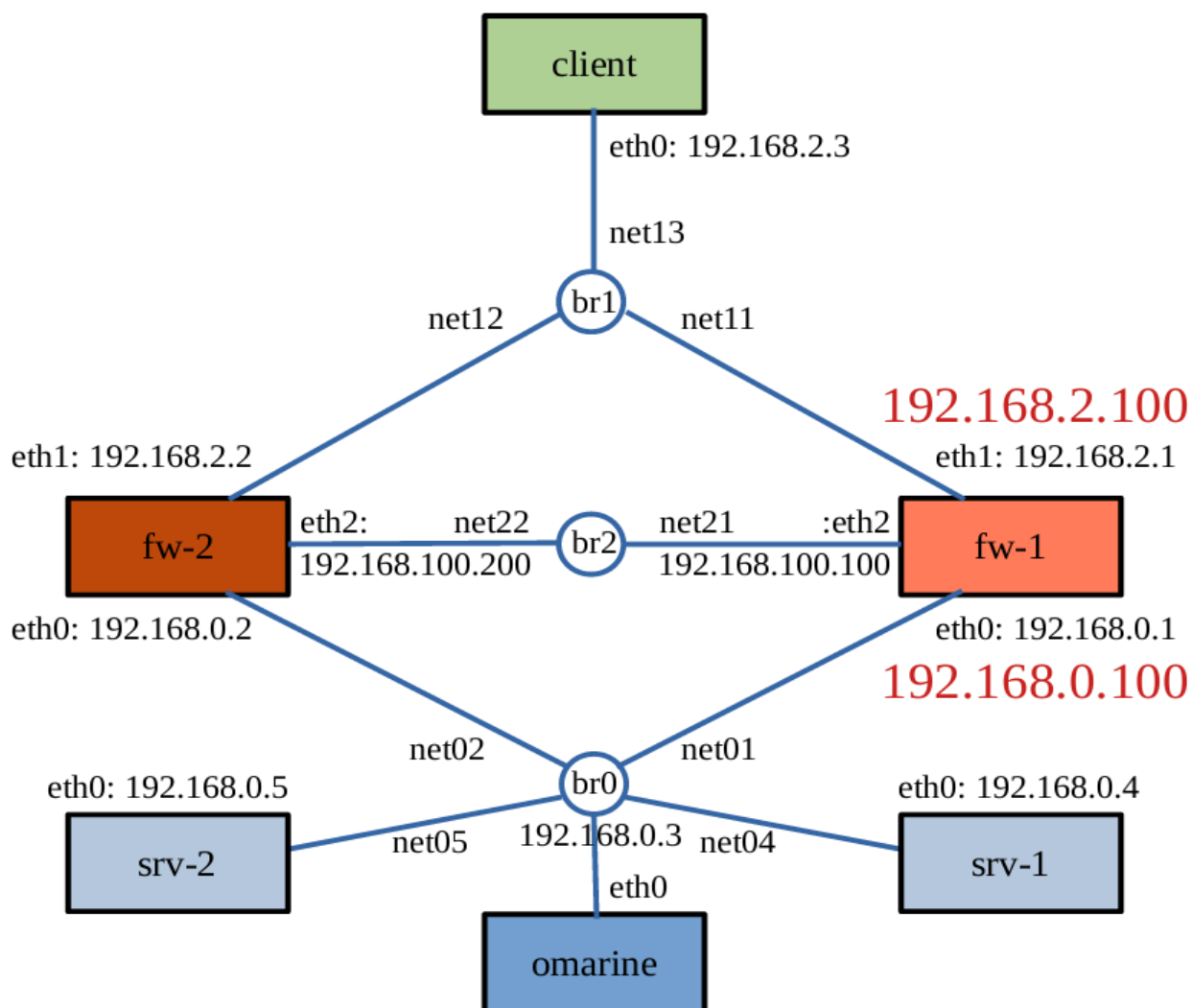
The screenshot shows a terminal window titled "fw-1 (1)" with a system menu and window control buttons. The terminal displays the output of the command `systemctl status conntrackd`. The output indicates that the `conntrackd.service` is loaded and active (running). It provides details such as the service file path, activation time, documentation, main PID (1795), CPU usage (1.780s), and CGroup. Below this, a series of log messages show the service starting and tracking connections. The terminal window is overlaid on a desktop background with a blue geometric pattern.

```
bash-5.1$ systemctl status conntrackd  
● conntrackd.service - netfilter connection tracking user-space daemon  
   Loaded: loaded (/usr/lib/systemd/system/conntrackd.service; enabled;>  
   Active: active (running) since Wed 2021-12-08 08:38:57 +07; 5h 6min >  
     Docs: man:conntrackd(8)  
    Main PID: 1795 (conntrackd)  
         CPU: 1.780s  
    CGroup: /system.slice/conntrackd.service  
            └─1795 /usr/sbin/conntrackd  
  
Dec 08 08:38:57 fw-1 systemd[1]: Starting netfilter connection tracking u>  
Dec 08 08:38:57 fw-1 conntrackd[1795]: [Wed Dec  8 08:38:57 2021] (pid=17>  
Dec 08 08:38:57 fw-1 conntrackd[1795]: [Wed Dec  8 08:38:57 2021] (pid=17>  
Dec 08 08:38:57 fw-1 conntrackd[1795]: [Wed Dec  8 08:38:57 2021] (pid=17>  
Dec 08 08:38:57 fw-1 conntrackd[1795]: [Wed Dec  8 08:38:57 2021] (pid=17>  
Dec 08 08:38:57 fw-1 systemd[1]: Started netfilter connection tracking us>  
lines 1-15/15 (END)
```

Right after it is run by the service at machine boot, the conntrackd is ready for its client commands in the HA system's the transition script.

11. Load balancing

Going hand in hand with high availability (HA) is the load balancing technique. Two servers `srv-1` and `srv-2` to be added to the network topology



The omarine server running keepalived acts as a virtual server that distributes connections equally to the real servers srv-1 and srv-2. All service access to the virtual server is routed to real servers. Real servers are health checked to monitor the health of the network. A quorum is set (required minimum total weight of all live servers in the pool). If a real server has problem and the quorum is below the minimum, then access goes to a sorry server. In this example we create a virtual Web service. The contents of the server's homepage are as follows:

- Real server srv-1: Hello, I am server 1.
- Real server srv-2: Hello, I am server 2.
- The sorry server: Sorry, the quorum was not achieved!

The quorum is set to 2. At startup both real servers are healthy and the quorum is sufficient. From the client we go to <http://omarine.omarine.co> several times, the connections go to srv-1 and srv-2 in turn equally. Then we stop the service on the srv-1. At this point the quorum is not reached and we are redirected to the home page of the sorry server at omarine

The configuration file is as follows:

```
global_defs {
    notification_email {
        tuyen
    }
    smtp_server 192.168.0.3 25
    smtp_alert yes
    smtp_alert_checker yes
    max_auto_priority 99
    lvs_flush
}
virtual_server_group G1 {
    192.168.0.3 80
}
virtual_server group G1 {
    lvs_sched rr
    lvs_method NAT
    protocol TCP
}
```



```

virtualhost omarine
  quorum 2
  sorry_server 192.168.0.3 80
  sorry_server_lvs_method DR
  inhibit_on_failure
  real_server 192.168.0.4 80 {
    weight 1
    HTTP_GET {
      url {
        path /
        digest 22f5bb897b25dabd36e19c2f20439ff7
      }
    }
  }
  real_server 192.168.0.5 80 {
    weight 1
    HTTP_GET {
      url {
        path /manual/
        digest e073a410a4ef34b0adc28d280b019383
      }
    }
  }
}

```

Notifications about the virtual service and the health of real servers are emailed to tuyen@omarine.omarine.co

```
Terminal -
File Edit View Terminal Tabs Help
Return-Path: <root@omarine.omarine.co>
Date: Mon, 13 Dec 2021 11:12:23 +0700
From: root@omarine.omarine.co
Subject: [omarine.omarine.co] Realserver [192.168.0.4]:tcp:80 of virtual server [G1]:0 - DOWN
X-Mailer: Keepalived
To: tuyen
Status: R0

=> CHECK failed on service : HTTP request failed <=

? 4
Message 4:
From root@omarine.omarine.co Mon Dec 13 11:14:23 2021
Return-Path: <root@omarine.omarine.co>
Date: Mon, 13 Dec 2021 11:14:23 +0700
From: root@omarine.omarine.co
Subject: [omarine.omarine.co] Realserver [192.168.0.4]:tcp:80 of virtual server [G1]:0 - UP
X-Mailer: Keepalived
To: tuyen
Status: R0

=> CHECK succeed on service <=

? 5
Message 5:
From root@omarine.omarine.co Mon Dec 13 11:14:23 2021
Return-Path: <root@omarine.omarine.co>
Date: Mon, 13 Dec 2021 11:14:23 +0700
From: root@omarine.omarine.co
Subject: [omarine.omarine.co] Virtualserver [G1]:0 - UP
X-Mailer: Keepalived
To: tuyen
Status: R0

=> Gained quorum 2+0=2 <= 2 <=

? █
```

In message 3, the real server srv-1 was stopped, so the message reported the failure status of srv-1 (192.168.0.4). In message 4, the server srv-1 was reactivated and the HTTP_GET checking was successful. The last message, number 5, which is a report about the virtual server, it had sufficient quorum and returned to the normal service status.

12. Expectation: part 1: helper

Some application protocols such as FTP, H.323 and SIP divide transaction into two flows with two separate connections. The first connection is the control connection followed by data connection. In the FTP case with passive mode, the first connection to port 21 of the file server is the control connection.

After the user has logged in and runs an ftp command eg 'ls' is at the beginning of the data connection in a very large port range 1024 – 65535. The firewall has to allow these two connections to operate, but not more than that, it should not allow so many ports. First the server and client will negotiate destination port number that the server will open to listen. But this information belongs to the application protocol and is not obtained by normal connection tracking that only get the protocol header deepest to layer 4. The data analysis behind the TCP header claim a protocol help called helper. This helper read the destination port number and creates a so-called expectation. That's the expectation of data connection to become 'related' to the control connection that acts as a master connection. We then only need to write 'ct state related' rule without knowing what the specific destination port is. The helper registration is made by the `nf_conntrack_helper_register()` function. The argument entering the function is a `nf_conntrack_helper` structure pointer. This structure is the representative of the helper, including the following fields, we explain to the FTP case:

- Helper's name, "ftp".
- A tuple is a `nf_conntrack_tuple` structure, information that it needs to hold is the layer 3 protocol number, layer 4 protocol number and working port number of master connection. The layer 3 protocol is IPv4 or IPv6. The layer 4 protocol is TCP, the working port is 21.
- Function pointer help to handle the application protocol. This is the main job of helper.
- `hnode` is a `hlist_node` structure to insert this helper at the head of a list in a bucket in the helper hash table.
- Function pointer `from_nlattnr`, the function is called when the master conntrack is injected from the user space, typically when the firewall recovers the connection. This function intends to handle netlink attributes but FTP only sets `NF_CT_FTP_SEQ_PICKUP` flag to ignore the sequence number checking of the data because the previous backup firewall does not know the sequence number (only the active firewall can update the sequence numbers).
- Flags, expectation policy and some other things.

The registration function takes the index based on the helper's tuple to access the array of the hash table and insert the helper at the head of the list at that index. Once the node is in the hash table, the helper is its container so can be retrieved.

When the first packet of the master connection arrives, like any normal connection, the connection tracking system (hereafter called the conntrack system) will create an entry called conntrack. A conntrack is a `nf_conn` structure that holds connection information, including:

- The status field holds the connection state, is the packet has been seen both ways, has left the box (confirmed and conntrack has been inserted into the official hash table in the last hook postrouting), is the expected connection, is the new connection or dying...
- `tuplehash` is an array of `nf_conntrack_tuple_hash` structs, there are two structures for the original and reply directions. Each structure has two fields: a tuple holding connection information and a `hnode`. Each structure is inserted into the hash table through its `hnode`. When it is need to access a conntrack that has a node in the hash table, the system computes the index to get the bucket containing the node, and then finds the node in the bucket. Once a node is present, the pointer is moved back by data offset

to the beginning of the `nf_conn` structure and `conntrack` is obtained. Two tuples are the most important members of a `conntrack`. Unlike helper tuple that need only three types of information to access its hash table, `conntrack`'s tuples contain full information: source address, destination address, source port, destination port, layer 3 protocol, layer protocol 4 and direction. One tuple is for a packet's original direction and another for reply direction.

- master `conntrack`, if it is a data connection, this element points to the `conntrack` of the control connection.
- `timeout` defines the life time of a `conntrack`. When it expires, the `conntrack` is destroyed.
- `ct_general` is a `nf_conntrack` structure. This structure has only one member, `use`, which is used to manage the reference count of the `conntrack` object. When the reference count decreases to 0, it is safe to release the object. In practice, the `nf_ct_put()` function is used to reduce the reference count by 1 and if the reference count is zero, the object is released. The function `nf_ct_expect_put()` has the same function as the `nf_ct_put()` function but applies to the expectation object.
- The `ext` pointer points to the `nf_ct_ext` structure. This structure has a `data` field which holds some extension structures to be added as needed. The `offset` field is an array containing the offsets of each extended structure from the beginning of the container structure with the array index of their id. `offset[NF_CT_EXT_HELPER]` is the offset of the `nf_conn_help` structure. The `nf_conn_help` structure helps the master `conntrack` manage its expectations and communicate with the helper. The `nf_conn_help` structure has four members:
 - The helper pointer points to the helper.
 - `expectations` is a structure `hlist_head`. It is the head of the list of expectations of the master `conntrack`. A newly created expectation will be inserted at the head of this list. This is how the master `conntrack` manages its expectations independently of the general management of expectations in the expectation hash table.
 - `expecting` is an array of integers, holding the number of current expectations by class. Currently FTP only uses one class, 0.
 - `data` is a 32 byte field for helper-specific information. FTP uses this data for the `nf_ct_ftp_master` structure which holds the `NF_CT_FTP_SEQ_PICKUP` flag and sequence number information as described above.

Another extension structure is `nf_conntrack_ecache`. This structure has a `cache` field that holds reporting events such as `IPCT_NEW`, `IPCT_DESTROY`, `IPCT_HELPER`.

- Some other things.

During `conntrack` initialization, helper assignment is performed if automatic helper assignment is configured. This includes finding the helper, adding the `nf_conn_help` structure and assigning its helper pointer to the helper. At the stage of the last hook, postrouting, before the packet comes out of the box, `conntrack` is confirmed with the `nf_conntrack_confirm()` function. If the packet is accepted this function inserts `conntrack` into the hash table. Then it checks the helper with the `nfct_help()` function. The `nfct_help()` function returns a pointer to the `nf_conn_help` structure. Because the helper was assigned, this went smoothly. So it sets the event with the `nf_conntrack_event_cache()` function, setting

bit 1 << IPCT_HELPER. Finally, the `nf_conntrack_confirm()` function delivers the event with the `nf_ct_deliver_cached_events()` function (in the `nf_conntrack_core.h` source file). This function first looks for cached events using the `nf_ct_ecache_find()` function. Since the event is set, it finds this one together the other events. Thus the cached events are delivered once, and they are deleted immediately after delivery (this is done by the statement: `events = xchg(&e->cache, 0);`).

In the `nf_ct_deliver_cached_events()` function there is a notify pointer of the `nf_ct_event_notifier` structure. This construct has a `fcn` field which is a function pointer to handle event messages.

Meanwhile in the source file `nf_conntrack_netlink.c` there is a struct `nf_ct_event_notifier` that was initialized when declared with its `fcn` pointer assigned the `ctnetlink_conntrack_event()` function. Now let's briefly analyze network activity to understand the event handover process.

The net structure holds the network operations. This structure has a `gen` pointer field that points to a `net_generic` structure (`generic.h` source file) which has a `ptr` field to be an array whose indexes are the ids of the net's network operations. Each network operation registered with the `register_pernet_subsys()` function which takes an argument of a `pernet_operations` structure pointer (the `net_namespace.c` source file). The `pernet_operations` structure is a representation of network activity. It has a `list` field to insert into the list of operations, an `id` pointer, an `init` function pointer, and several others. The `register_pernet_subsys()` function calls the `register_pernet_operations()` function. This function calls the `ida_alloc_min()` function to generate an id for the activity, and then it calls the `__register_pernet_operations()` function for the specific registration. In turn, the `__register_pernet_operations()` function adds the operation to the list and calls the `ops_init()` function to initialize the operation with an initialized net structure named `init_net`. It is the default net and also the only net in the system if we do not create additional network namespaces.

The `ops_init()` function calls the `net_assign_generic()` function to assign the new `net_generic` structure (if necessary) to the net's `gen` pointer, and then calls the `pernet_operations` structure's `init` function to initialize the network operation. The registration of network activity here has been completed.

We return to the event delivery part of the `ctnetlink` activity. The `ctnetlink` subsystem registers activity with the `register_pernet_subsys()` function where the active structure is `ctnetlink_net_ops`. The `ctnetlink_net_ops` has an `init` function of `ctnetlink_net_init()`, so registration leads to a call to `ctnetlink_net_init()`. The function `ctnetlink_net_init()` again calls the function `nf_conntrack_register_notifier()` with the argument of the above `nf_ct_event_notifier` structure pointer. Talking more about the net structure, it has a `ct` field which is a `netns_ct` structure that manages conntracks. The `netns_ct` structure has a field `nf_conntrack_event_cb` which is a pointer to the `nf_ct_event_notifier` structure with the goal of holding event notification callback function. So the `nf_conntrack_register_notifier()` function assigns address of the `nf_ct_event_notifier` structure above to the net's `nf_conntrack_event_cb` pointer.

Back to the `nf_ct_deliver_cached_events()` function, which uses the `rcu_dereference()` function to obtain `net->ct.nf_conntrack_event_cb` and assigns to the notify pointer. Pointer notify then run `fcn` ie call `ctnetlink_conntrack_event()` function with arguments to be the events and address of a `nf_ct_event` structure which contains the conntrack pointer.

nf_ct_deliver_cached_events() only delivers the events, while the ctnetlink_contrack_event() function actually broadcasts the events.

What we are interested in here is IPCT_HELPER ie event creating helper for the master conntrack.

```
static struct pernet_operations ctnetlink_net_ops = {
    .init      = ctnetlink_net_init,
    .exit_batch = ctnetlink_net_exit_batch,
};

static struct nf_ct_event_notifier ctntl_notifier = {
    .fcn = ctnetlink_contrack_event,
};
```

```
ctnetlink_init();
```

```
register_pernet_subsys(&ctnetlink_net_ops);
```

```
(&ctnetlink_net_ops)->init(net);
```



```
ctnetlink_net_init(net);
```

```
nf_conntrack_register_notifier(net, &ctntl_notifier);
```

```
rcu_assign_pointer(net->ct.nf_conntrack_event_cb,
    &ctntl_notifier);
```

The ctnetlink_contrack_event() function generates an event message including the header and payload in a socket buffer and fills the information based on the events and conntrack.

There are many types of events, but messages fall into three categories:

1. DESTROY: if the event is IPCT_DESTROY, then the message type is DESTROY (IPCTNL_MSG_CT_DELETE bit).
2. NEW: if the event is IPCT_NEW or IPCT_RELATED, the message type is NEW (IPCTNL_MSG_CT_NEW bit, with the NLM_F_CREATE | NLM_F_EXCL flags are set).
3. UPDATE: in the remaining cases, the message type is UPDATE (bit IPCTNL_MSG_CT_NEW but do not set the flag). The UPDATE message means that conntrack was created in the previous events and this is its updated status.

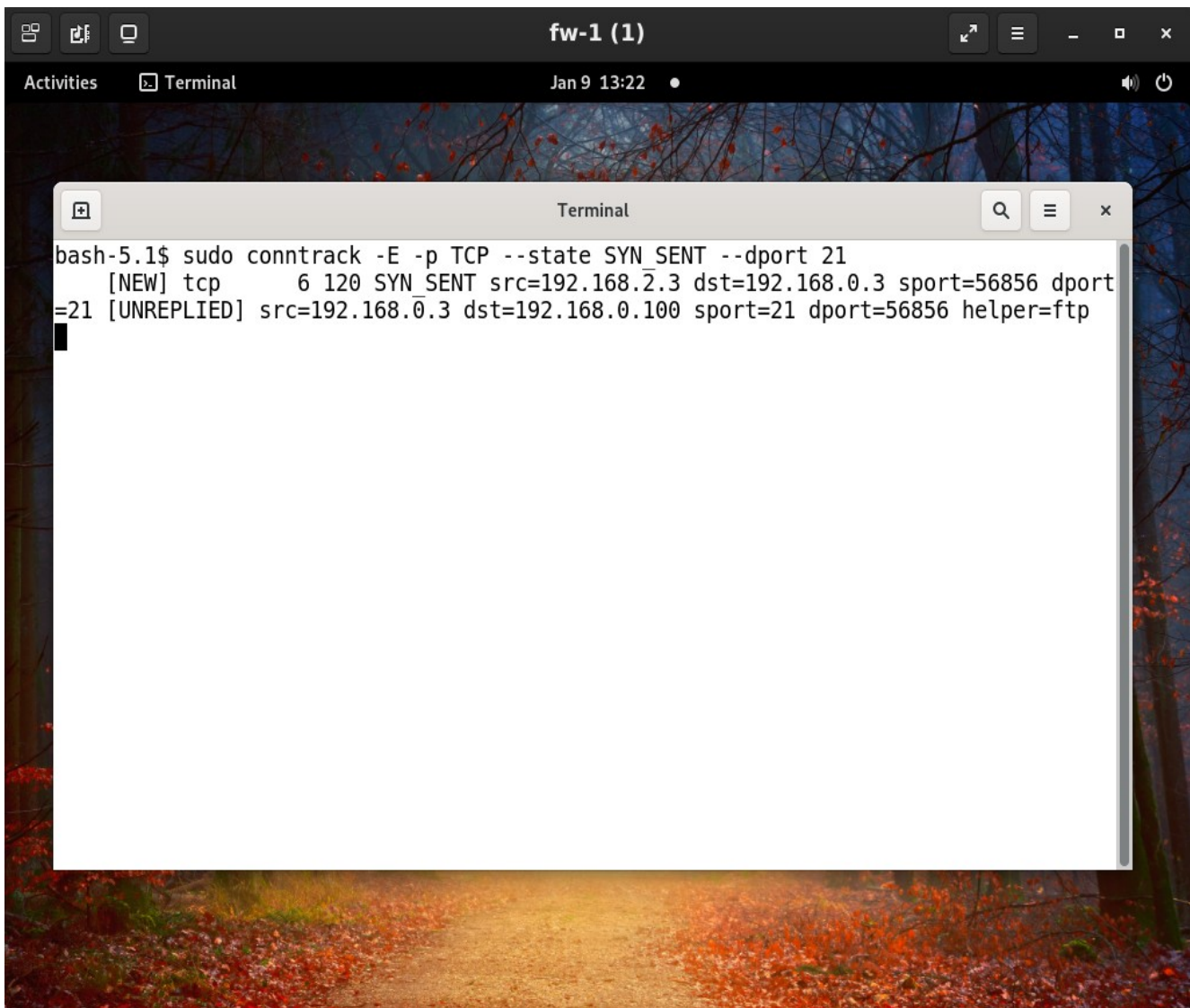
The helper creation happens in the new connection, the IPCT_HELPER event goes together with the IPCT_NEW event so the message type is NEW.

There are two message subsystems: the message subsystem for conntrack is NFNL_SUBSYS_CTNETLINK and the message subsystem for expectation is NFNL_SUBSYS_CTNETLINK_EXP. In this case it is conntrack so the function puts the type NFNL_SUBSYS_CTNETLINK in the message header.

With the IPCT_HELPER event it calls the ctnetlink_dump_helpinfo() function. The ctnetlink_dump_helpinfo() function asserts that it must obtain the helper pointer of the nf_conn_help structure of conntrack, otherwise the helper is considered non-existent . This happens when the firewall recovers the connection, it injects the conntrack whose inherent helper into the kernel table but loses the helper shortly after (we'll fix this soon).

The ctnetlink_dump_helpinfo() function then sets the netlink attribute CTA_HELP and the nested attribute CTA_HELP_NAME with the helper's name to the message. Finally, the nfnetlink_send() function sends the message to the opening netlink sockets.

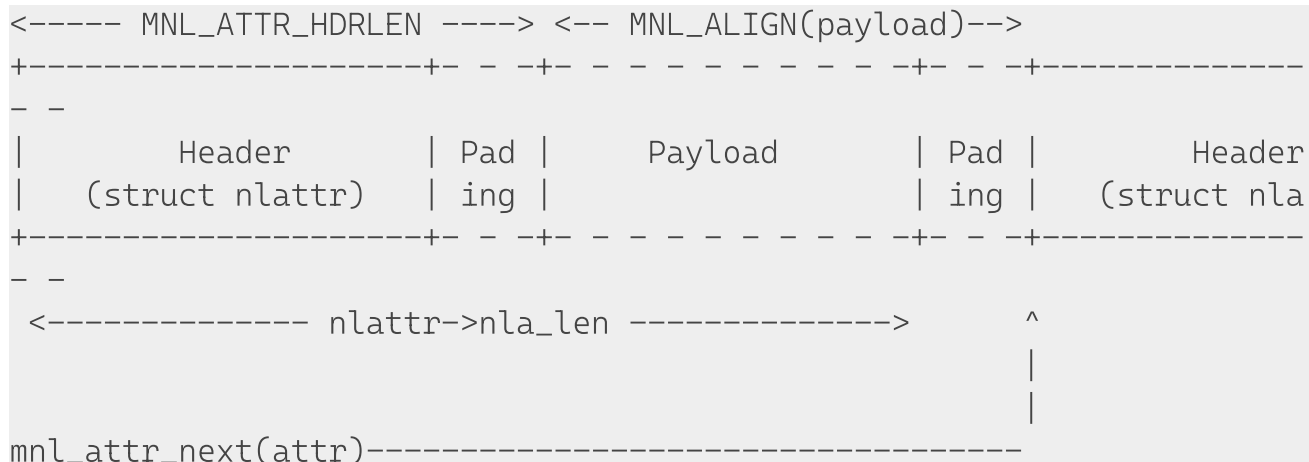
On the userspace side, the conntrack tool (command conntrack -E) opens a nfnetlink socket. Whenever events are delivered, it will collect the message and output the information

A screenshot of a Linux desktop environment. The background is a scenic image of a path covered in autumn leaves. In the foreground, there is a terminal window titled "fw-1 (1)" with a dark theme. The terminal shows the command `sudo conntrack -E -p TCP --state SYN SENT --dport 21` being executed. The output is: `[NEW] tcp 6 120 SYN SENT src=192.168.2.3 dst=192.168.0.3 sport=56856 dport=21 [UNREPLIED] src=192.168.0.3 dst=192.168.0.100 sport=21 dport=56856 helper=ftp`. Below the terminal window, there is another smaller terminal window titled "Terminal" with a light theme, which is currently empty.

As we can see, the message is NEW and the helper is created at the very first step in the three-way TCP connection establishment.

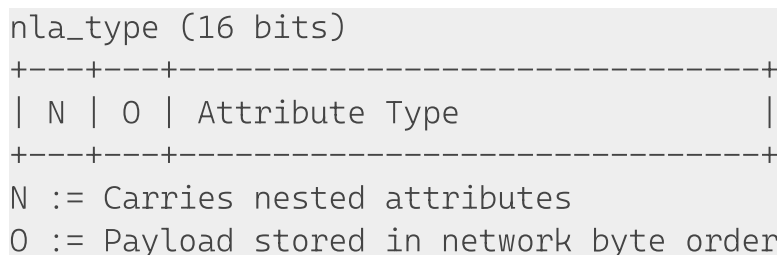
For the `conntrack -L` command, it queries the `conntrack` hash table and receives `conntrack` messages. The messages are then fed to a callback function named `__callback` (the `callback.c` source file of the `libnetfilter_conntrack` library). This function handles both `conntrack` and expectation messages. It analyzes the `nlmsg_hdr` message header structure and finds that the `nlmsg_type` field matches `NFNL_SUBSYS_CTNETLINK`, so it knows this is a `conntrack` message. The function therefore creates an `nf_conntrack` struct pointed by the `ct` pointer, which holds the user-space `conntrack` information, and uses the `nfct_nlmsg_parse()` function (`parse_mnl.c` source file) to parse the message and populate the `conntrack`. This function again calls the `nfct_payload_parse()` function to parse the payload. The `nfct_payload_parse()` function declares an array of pointers of `nlattr` structure (`netlink`

attribute structure), the array name is tb. The nlattr structure has two fields: nla_len is the length of the attribute data including the header and payload, and nla_type is the attribute type.



Padings are added so that the header and payload extend to multiples of 4.

nla_type consists of 16 bits: the last 14 bits are the actual attribute type; if the first bit is 1, then this attribute has nested attributes inside its payload; if the second bit is 1 then the payload is stored in network byte order, otherwise host byte order. However, the event messaging subsystem does not use the second bit because the netlink attributes are known in advance of the byte order of the payload and therefore need not be checked for this bit.



Next, the nfct_payload_parse() function calls the mnl_attr_parse_payload() function to populate the tb array, in the arguments nfct_parse_conntrack_attr_cb is a callback function called to process the data. The function mnl_attr_parse_payload() (libmnl library's attr.c source file) uses the mnl_attr_for_each_payload macro to iterate through the attributes in the message payload. For each attribute the callback function nfct_parse_conntrack_attr_cb() is called to process and tb contains attribute pointers with the array indexes to be the attribute types. For helper event the attribute pointer is tb[CTA_HELP].

Back to the nfct_payload_parse() function, there is now the tb[CTA_HELP] pointer, so it calls the nfct_parse_helper() function. The nfct_parse_helper() function again parses the CTA_HELP attribute to get the nested attributes in its payload. The nfct_parse_helper() function again uses a different tb array, and it obtains pointer tb[CTA_HELP_NAME] which points to the nested attribute, CTA_HELP_NAME.

At this point, we can easily get the character string that is the name of the helper by taking the payload of CTA_HELP_NAME, that is, adding to the pointer tb[CTA_HELP_NAME] an amount of

MNL_ATTR_HDRLEN. The `nfct_parse_helper()` function copies the name of the helper to the `helper_name` field of `conntrack` via the `ct` pointer. It then sets the `ATTR_HELPER_NAME` bit for `ct->head.set`. The statement in the source code is as follows:

```
set_bit(ATTR_HELPER_NAME, ct->head.set);
```

Message retrieving is complete, the `__callback` function passes the work to another callback function that was registered by `nfct_callback_register()`. Now it is the job of the `conntrack` tool to receive the `conntrack` handover from the library to render the information. That callback function is `dump_cb()`. After a bit of filtering by user demand on the command line, it calls the `nfct_snprintf_labels()` function to output the information. The `nfct_snprintf_labels()` function goes through a chain of intermediate processing functions, and the final output function is `__snprintf_conntrack_default()` (`snprintf_default.c` source file of the `libnetfilter_conntrack` library). The `__snprintf_conntrack_default()` function checks the `ATTR_HELPER_NAME` bit of `ct->head.set` and finds out so it outputs the name of the helper, here “ftp”.

If the master `conntrack` fails to create or loses helper, then `conntrack -L` or `conntrack -E` cannot display the name of the helper. Otherwise, once they see the `ATTR_HELPER_NAME` bit they will display `helper=ftp`



```
fw-1 (1)
Activities Terminal Jan 12 15:39
Terminal
bash-5.1$ sudo conntrack -L -p TCP --dport 21
tcp      6 431811 ESTABLISHED src=192.168.2.3 dst=192.168.0.3 sport=56906 dport=
21 src=192.168.0.3 dst=192.168.0.100 sport=21 dport=56906 [ASSURED] mark=0 secct
x=system_u:object_r:unlabeled_t:s0 helper=ftp use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
bash-5.1$
```

conntrack -E and conntrack -L both handle conntrack messages, so why doesn't conntrack -L present the message type?

There are some notable points in the output of `conntrack -L` :

- The `dump_cb()` function does not handle the type of the message, so it does not display the message type.
- The kernel doesn't set appropriate flag for the message header so the rendering doesn't show the correct message type.
- `conntrack -L` only lists the conntracks currently in the kernel's conntrack hash table (hereafter called the conntrack table) so obviously there is no DESTROY type message. Conntrack has thus been removed from the list. Expired conntracks are withdrawn from the conntrack table and placed on the dying table, but usually immediately after that it is withdrawn from the dying table and freed from memory. When the system is working properly, the dying table is always empty.
- The hash of a conntrack depends only on net and tuple. During the three-way TCP connection establishment, the net is the same (and is the unique net when not using the network namespace). The protocol parameters of layers 3 and 4 of the steps are the same so they have the same tuple (second step – SYN_RECV uses the reply tuple). Therefore conntrack is created only once in the first step (SYN_SENT). The next steps have the same hash to get into the conntrack bucket, and with tuple, net (and zone alike) they get the pre-generated conntrack. Thus, all three TCP connection steps have only one conntrack in the table, which is updated to the last step (LAST_ACK), and `conntrack -L` only outputs that single entry (conntrack -E shows all three entries).
- The helper information is filled in the message using the `ctnetlink_dump_helpinfo()` function as for the event.
- Each conntrack has two table entries, one for the original tuple and one for the reply tuple. But dumping the table requires only one entry, of the original tuple.
- A new conntrack is normally set three bits of status until it is inserted into the table, if NAT is not in use: `IPS_DST_NAT_DONE`, after the packet exits the `nf_conntrack_in()` function and the setup of the destination NAT is considered complete (even with use NAT or not) before routing; `IPS_SRC_NAT_DONE`, source NAT setup is considered complete, before conntrack is confirmed by `nf_conntrack_confirm()`; `IPS_CONFIRMED`, conntrack is confirmed and inserted into the hash table, has nothing more to do with the packet and let it go out of the box (add the `IPS_EXPECTED` bit if the conntrack matches expectation).

If you want `conntrack -L` to display the full message type you can modify the `dump_cb()` function and the kernel. The modified code to set message flag as follows:

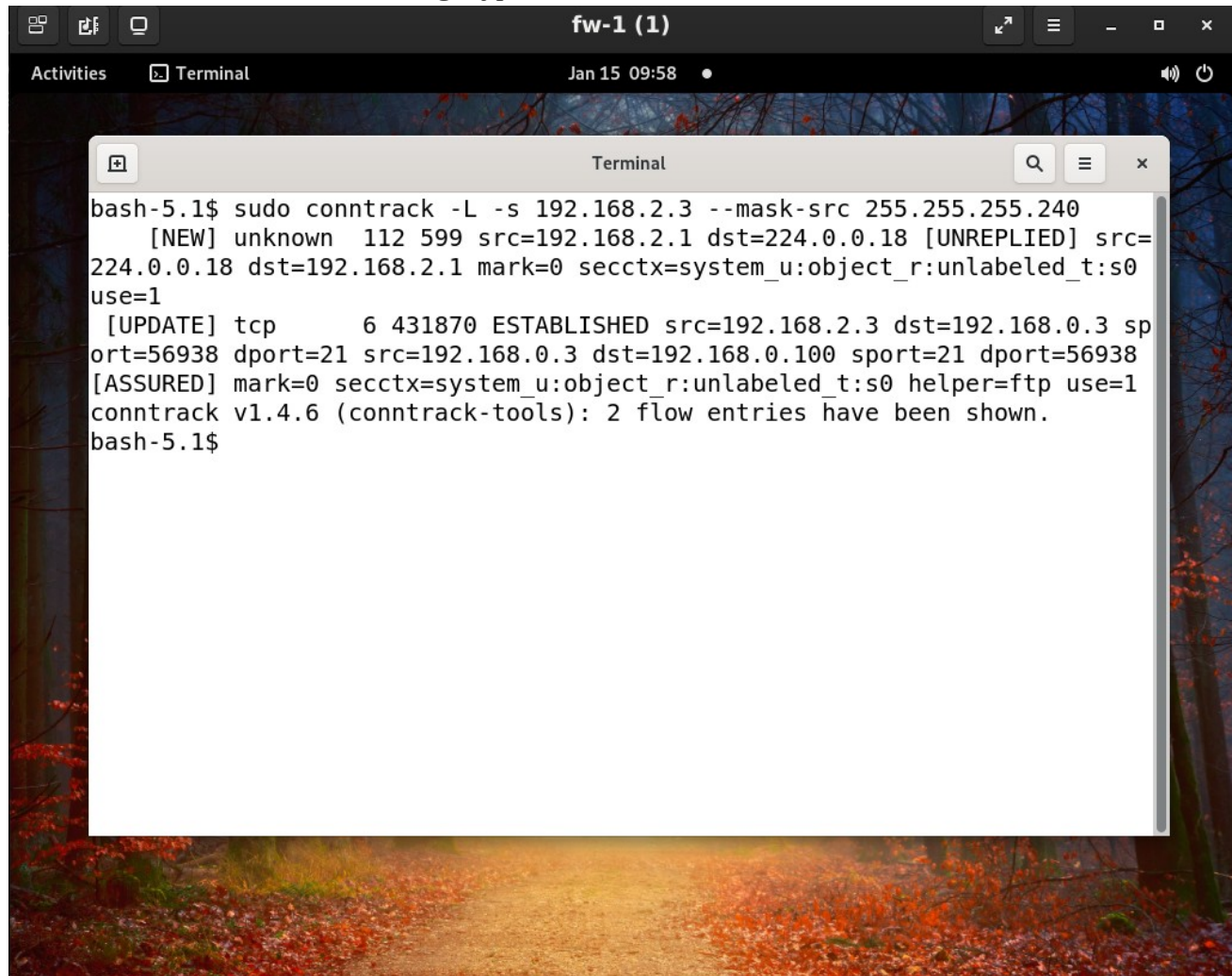
```
if (ct->status == (IPS_CONFIRMED | IPS_NAT_DONE_MASK))
    flags |= NLM_F_CREATE;
/* fill message */
...
flags &= ~NLM_F_CREATE;
```

Where `IPS_NAT_DONE_MASK := (IPS_DST_NAT_DONE | IPS_SRC_NAT_DONE)`.

A better way is to use the `IPS_SEEN_REPLY` bit to distinguish a new conntrack:

```
if (!test_bit(IPS_SEEN_REPLY_BIT, &ct->status))
    flags |= NLM_F_CREATE;
/* fill message */
...
flags &= ~NLM_F_CREATE;
```

Now conntrack -L shows the message types



13. Expectation: part 2: expectation

Recovering helper

The feature of fault-tolerant firewall is the ability to recover connections. But if it loses the helper, the connection recovering fails. When conntrackd injects a conntrack whose inherent helper into the kernel conntrack table, the netlink subsystem creates helper for it. Unfortunately, the work of NAT later took the helper away (don't use automatic helper assignment, now the safe way is to explicitly define helper using the firewall rule ct helper set). To fix it, we add the following code to the

`__nf_ct_try_assign_helper()` function:

```
/* tmpl has no helper but injected conntrack may have */
if (help && helper == NULL)
    helper = rcu_dereference(help->helper);
```

expectation

An expectation is a `nf_conntrack_expect` structure. Notable fields are:

- `lnode`: A `hlist_node` to insert the expectation into the list in the master conntrack.
- `hnode`: A `hlist_node` to insert the expectation into the expectation hash table (expect table).
- `tuple`: Contains layer 3 and layer 4 protocol information to match the expected conntrack.
- `mask`: Tuple mask to compare expectations.
- `helper`: The helper pointer to assign to the expected conntrack.
- `master`: conntrack master.
- `timeout`: When the expectation expires, the system destroys the expectation and broadcasts the

`IPEXP_DESTROY` message.

- `use`: When use is equal to 1 the function `nf_ct_expect_put()` will destroy expectation.
- `flags`: If the flag is not set the expectation is used only once, it is destroyed immediately after that.
- `class`: The class of expectation. FTP uses only one class, 0.

When the user is at the ftp command prompt at the client and runs a command such as the 'ls' command, client/server negotiation about the destination port number begins. The client requests a port in the origin direction and the server returns the port number in the reply direction. The negotiation uses the master connection, i.e. uses the same source port and destination port 21 so it reuses the master conntrack, this conntrack is old so the negotiation is not seen by the user, ie no more events are seen. The conntrack's internal status is changeless, it is still "established". However the external state, `ctinfo` is updated in the direction of the package. When the server returns the destination port number for the expected connection, i.e. in the reply direction, now `ctinfo` equals `IP_CT_ESTABLISHED_REPLY` and the helper's `help()` function begins to create expectation (in the postrouting stage).

It first takes the TCP header in the packet to calculate the data offset and gain the data. There is an important piece of information in the TCP header, ie the sequence number of the first byte in the data. The `help()` function always updates the sequence numbers so it knows that this first byte is behind a newline ('\n') in the data series. And so it analyzes the port pattern to find the destination port. Layer 3

protocol information including source address, destination address, ip protocol is already available in master conntrack, class is 0, layer 4 protocol is TCP, along with the destination port number just found is enough to create expectation for the expected connection.

The expected connection is exactly data connection, it happens right after the connection creating expectation. It is a new connection and a new conntrack is created. During initialization, conntrack looks for expectation, obviously it finds out the expectation just created for it. So it sets the IPS_EXPECTED bit to status, and so ctinfo is IP_CT_RELATED. The data connection is related to the master connection and is allowed to go through the firewall.

During the finding process, once found the expectation is destroyed because there is no flag by default. That shows that the expect table is always empty and the commands conntrackd -i exp, conntrackd -e exp and conntrack -L exp always appear nothing. Thus the user never sees expectations in the table. Furthermore, conntrackd's expectation synchronization function is meaningless.

To observe the expectation we need to modify the kernel a bit. First set the expectation flag to NF_CT_EXPECT_PERMANENT, so it will persist in the expect table and in the master conntrack's list until it expires.

But just setting the flag is not very good because every time a data connection is made a new expectation is created even though the old expectation has not expired yet. To reuse the old expectation we add the following code to the help() function:

```
/* We only update sequence number and create expectation
 * for old dest port. Old one is then renewed shortly
 * after that in the hash table. */
hlist_for_each_entry(exp, &help->expectations, lnode) {
    if (exp->tuple.dst.u.all == cmd.u.tcp.port)
        break;
    ret = NF_ACCEPT;
    goto out_update_nl;
}
```

Something is not right here! Expected connections cannot use other expectation. We need to define a port mask to allow a port number. In this practice we allow 4 ports. A mask is passed to the nf_ct_find_expectation() function:

```
__be16 port_mask = htons(0xffffc);
```

And the expectation finding condition is modified:

```
int found = 0;
...
for (h = 0; h < nf_ct_expect_hsize; h++) {
    hlist_for_each_entry(i, &nf_ct_expect_hash[h], hnode) {
        if (!(i->flags & NF_CT_EXPECT_INACTIVE) &&
            net_eq(net, nf_ct_net(i->master)) &&
            nf_ct_zone_equal_any(i->master, zone) &&
```

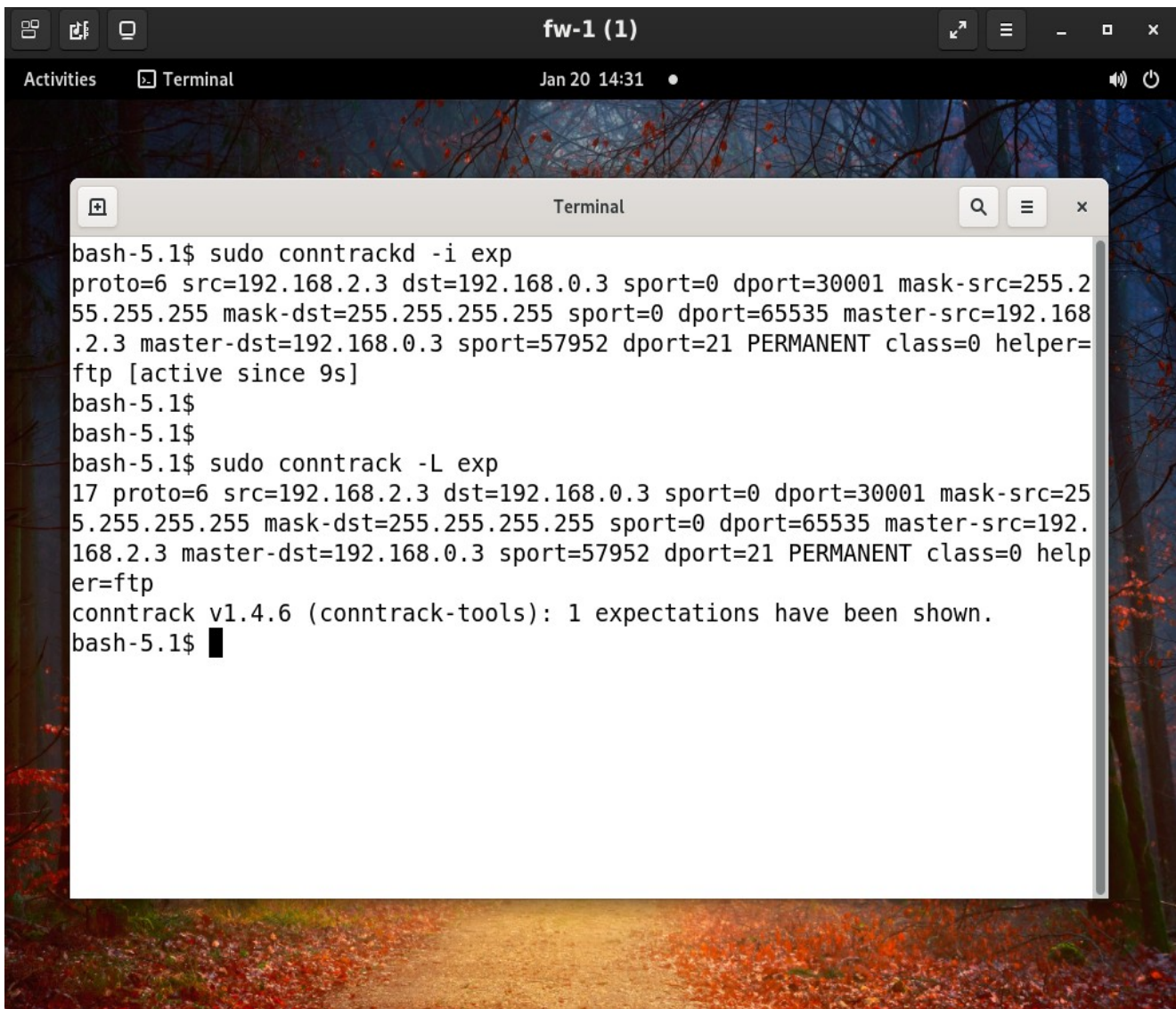
```

        nf_ct_tuple_src_mask_cmp(tuple, &i->tuple, &i->mask) &&
        nf_inet_addr_cmp(&tuple->dst.u3, &i->tuple.dst.u3) &&
        tuple->dst.protonum == IPPROTO_TCP &&
        !((tuple->dst.u.all ^ i->tuple.dst.u.all) & port_mask)) {
            exp = i;
            found = 1;
            break;
        }
    }
    if (found)
        break;
}

```

Note: The above patch file is only used in practice to observe expectations, do not apply in fact!

Now we will see expectations



```
bash-5.1$ sudo conntrackd -i exp
proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30001 mask-src=255.255.255.255 mask-dst=255.255.255.255 sport=0 dport=65535 master-src=192.168.2.3 master-dst=192.168.0.3 sport=57952 dport=21 PERMANENT class=0 helper=ftp [active since 9s]
bash-5.1$
bash-5.1$
bash-5.1$ sudo conntrack -L exp
17 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30001 mask-src=255.255.255.255 mask-dst=255.255.255.255 sport=0 dport=65535 master-src=192.168.2.3 master-dst=192.168.0.3 sport=57952 dport=21 PERMANENT class=0 helper=ftp
conntrack v1.4.6 (conntrack-tools): 1 expectations have been shown.
bash-5.1$
```

Once an expectation is created, it will last until it expires. In a special case, if the new expecting connection has the same destination port as an existing expectation, a new expectation is added. These two expectations will be in the same bucket in the hash table and have the same tuple so the old expectation is destroyed and the new expectation replaces it


```
bash-5.1$ sudo conntrack -E exp
[NEW] 30 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30003 m
ask-src=255.255.255.255 mask-dst=255.255.255.255 sport=0 dport=65535 maste
r-src=192.168.2.3 master-dst=192.168.0.3 sport=57952 dport=21 PERMANENT cl
ass=0 helper=ftp
[DESTROY] 19 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30003 m
ask-src=255.255.255.255 mask-dst=255.255.255.255 sport=0 dport=65535 maste
r-src=192.168.2.3 master-dst=192.168.0.3 sport=57952 dport=21 PERMANENT cl
ass=0 helper=ftp
[NEW] 30 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30003 m
ask-src=255.255.255.255 mask-dst=255.255.255.255 sport=0 dport=65535 maste
r-src=192.168.2.3 master-dst=192.168.0.3 sport=57952 dport=21 PERMANENT cl
ass=0 helper=ftp
[DESTROY] 0 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30003 ma
sk-src=255.255.255.255 mask-dst=255.255.255.255 sport=0 dport=65535 master
-src=192.168.2.3 master-dst=192.168.0.3 sport=57952 dport=21 PERMANENT cla
ss=0 helper=ftp
[NEW] 30 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30002 m
ask-src=255.255.255.255 mask-dst=255.255.255.255 sport=0 dport=65535 maste
r-src=192.168.2.3 master-dst=192.168.0.3 sport=57952 dport=21 PERMANENT cl
ass=0 helper=ftp
[DESTROY] 0 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30002 ma
```

In the first message, after a 'ls' command, an expectation is created for port 30003. After the second 'ls' command, the port is the same as before, 30003 so a new expectation is created and the old expectation in the same bucket is destroyed despite the time limit, 19 seconds left. In the fourth message, that new expectation is destroyed because of its expiration (a newly created expectation has a life time of 30 seconds). Now the expect table becomes empty so a new expectation is created for port 30002 by the next 'ls' command, in the fifth message. It ended up being destroyed also because of expiration. To see more, watch the video below

Sync expectations

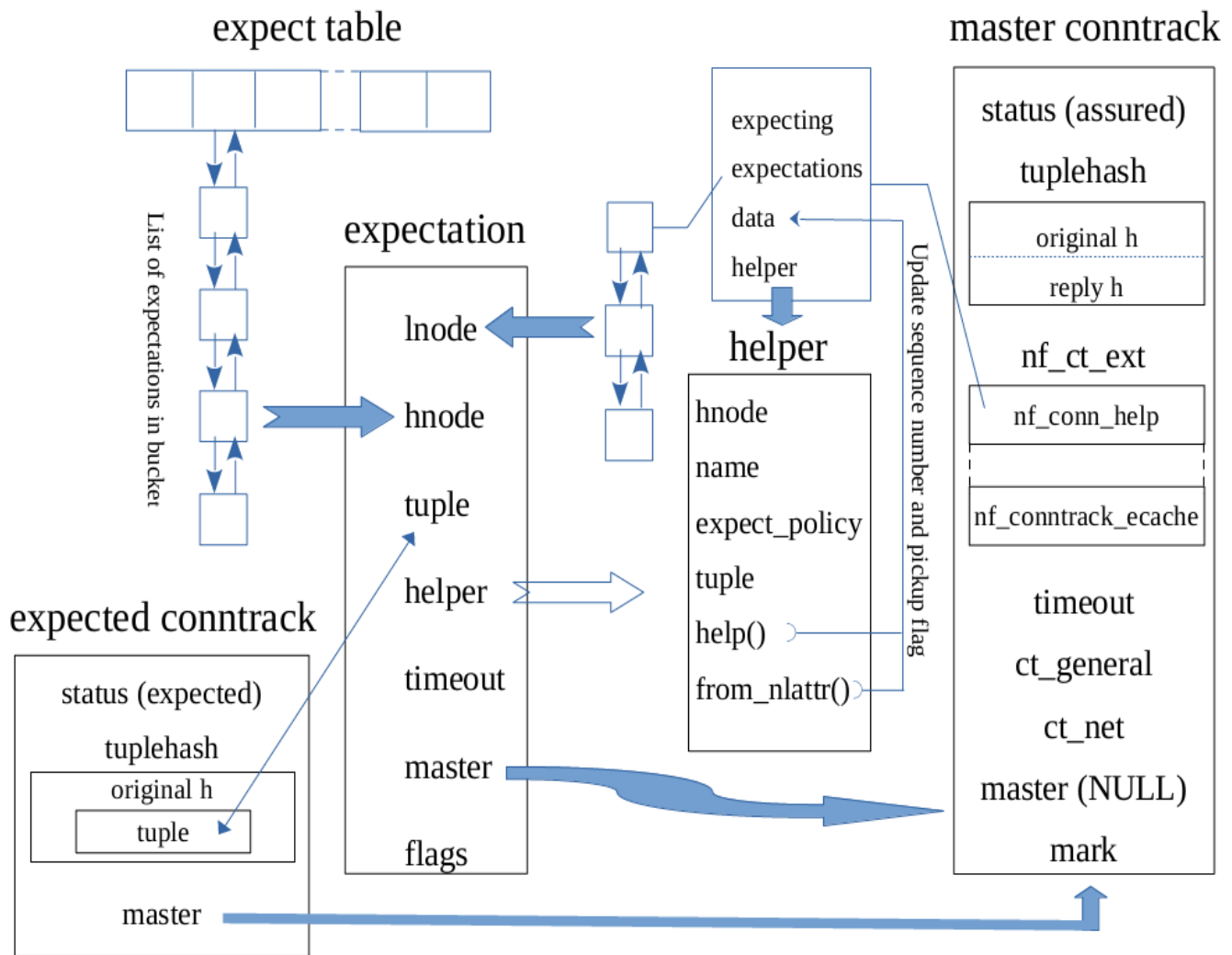
Now let's switch to the backup firewall fw-2. Its expect table is initially empty. The `conntrackd -e exp` command shows that fw-2 has obtained an expectation from the active firewall fw-1 and cached it externally. We use the `conntrackd -c` command to inject that expectation into the kernel expect table. So fw-2 now has an entry in the expect table

The screenshot shows a terminal window titled "fw-2 (1)" with a date and time of "Jan 20 15:34". The terminal output is as follows:

```
bash-5.1$ sudo conntrack -L exp
conntrack v1.4.6 (conntrack-tools): 0 expectations have been shown.
bash-5.1$ sudo conntrackd -e exp
proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30002 mask-src=255.255.255
.255 mask-dst=255.255.255.255 sport=0 dport=65535 master-src=192.168.2.3 master-
dst=192.168.0.3 sport=57998 dport=21 PERMANENT class=0 helper=ftp [active since
17s]
bash-5.1$ sudo conntrackd -c
bash-5.1$ sudo conntrack -L exp
297 proto=6 src=192.168.2.3 dst=192.168.0.3 sport=0 dport=30002 mask-src=255.255
.255 mask-dst=255.255.255.255 sport=0 dport=65535 master-src=192.168.2.3 mas
ter-dst=192.168.0.3 sport=57998 dport=21 PERMANENT class=0 helper=ftp
conntrack v1.4.6 (conntrack-tools): 1 expectations have been shown.
bash-5.1$
```

14. Expectation: part 3: diagram

When the dust of time covers the long lines of code, this image is easy to remember



15. Writing a firewall ruleset

Rules are the building material of the firewall. A firewall without a rule set is an empty firewall, like an air wall. Meaning it allows all including unwanted packets.

It's called rule-writing because rule-setting is flexible. There are no hard rules and for the same purpose we can write in many ways. For example, for the conntrack state we can rely on the internal state, the status of conntrack; or rely on external state, ctinfo. ctinfo is conntrack's state information but is updated according to the context. They are not completely identical. ctinfo shows the direction of the flow, status does not. Conversely, status indicates that a conntrack is expected while ctinfo no longer holds this information once the connection has been established in the conntrack semantics. However, in some situations they mean the same thing. For example ct state established is equivalent to ct status

seen-reply. ct state related is equivalent to ct status expected tcp flags syn. ct state established ct direction original is equivalent to ct status assured.

So how to write it depends on the person building the firewall. You can use goto or jump to break down the rule set to make it brighter.

Not a requirement, but you can create a flow table (flowtable) to accelerate packet forwarding and offload flow. Once conntrack is established, you can choose to place the flow entry into the table using flow add rule. Each entry is represented by a seven-element tuple: source address, destination address, source port, destination port, layer 3 protocol, layer 4 protocol, and input interface. In addition, it also caches the output interface. At the ingress hook, if the flow entry is found in the table, the packet will bypass the classic forwarding path i.e. not go through the netfilter hooks behind ingress but go directly to the output interface via the neigh_xmit() function in the hook nf_flow_offload_inet_hook().

In our example the firewall allows only the following services:

1. DNS (UDP, port 53)
2. www (TCP, port 80, 443)
3. File Transfer (TCP, port 21 with helper)
4. Secure remote login – ssh (TCP, port 22)
5. ping (ICMP, echo-request and echo-reply types)

The simple ruleset is as follows:

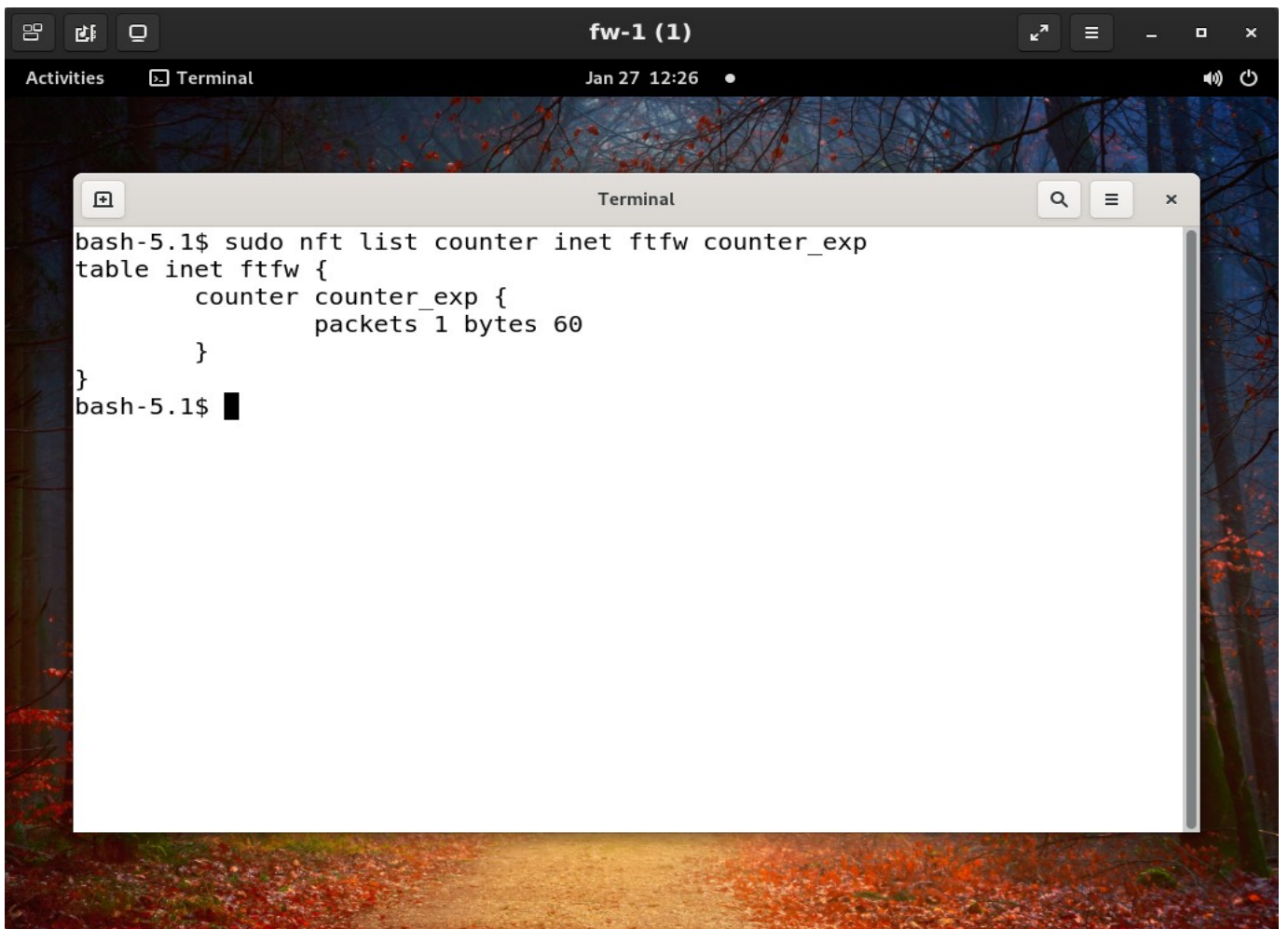
```
table inet ftfw {
    ct helper ftp-std {
        type "ftp" protocol tcp
        l3proto inet
    }
    counter counter_exp {
        packets 0 bytes 0
    }
    flowtable flow_exp {
        hook ingress priority filter
        devices = { eth0 }
    }
    chain prerouting {
        type filter hook prerouting priority filter; policy accept;
        tcp dport 21 ip daddr 192.168.0.3 ct state new \
            ct helper set "ftp-std"
    }
    chain forward_reply {
        ct status expected flow add @flow_exp \
            counter name "counter_exp" accept
        accept
    }
}
```

```

}
chain forward {
    type filter hook forward priority filter; policy drop;
    iifname "eth0" ct state established ct direction reply \
        goto forward_reply
    iifname "eth1" tcp dport { 21, 22, 80, 443 } tcp flags syn \
        ct state new accept
    iifname "eth1" udp dport 53 ct state new accept
    iifname "eth1" ct state related ct helper "ftp" accept
    iifname "eth1" ct state established ct direction original
accept
    icmp type echo-request limit rate 1/second accept
    ct state invalid log prefix "FORWARD-INVALID: "
}
}
table inet nat {
    chain postrouting {
        type nat hook postrouting priority srcnat; policy accept;
        oifname "eth0" ip saddr 192.168.2.3 snat ip to 192.168.0.100
    }
}

```

We create flowtable `flow_exp` with input interface `eth0` to offload large file download flow from file server. Offload only applies to the expected connections. The packets then bypass the classic forwarding path and go straight to the `eth1` interface. We also add counter `counter_exp` to check the offload effect. After downloading a large file the `counter_exp` counter shows that the traffic going through the classic forwarding path is only the first packet with 60 bytes, all the remaining traffic has gone through the forwarding fastpath

A screenshot of a Linux desktop environment. The background is a scenic image of a path covered in autumn leaves. In the foreground, there is a terminal window titled 'fw-1 (1)'. The terminal shows the command 'sudo nft list counter inet ftfw counter_exp' being executed. The output is a table definition for 'inet ftfw' containing a 'counter' rule named 'counter_exp' with 'packets 1' and 'bytes 60'. The prompt 'bash-5.1\$' is visible at the bottom of the terminal.

```
bash-5.1$ sudo nft list counter inet ftfw counter_exp
table inet ftfw {
    counter counter_exp {
        packets 1 bytes 60
    }
}
bash-5.1$
```

16. Kernel patch due to loss of helper

The issue of a conntrack injection from user space lost helper that was mentioned in the chapter “13. Expectation: part 2: expectation”. That’s one way to fix it.

A patch has been added to the Linux kernel as follows:

```
[ Upstream commit d1ca60efc53d665cf89ed847a14a510a81770b81 ]
```

When userspace, e.g. conntrackd, inserts an entry with a specified helper,

its possible that the helper is lost immediately after its added:

```
ctnetlink_create_conntrack
```

```
-> nf_ct_helper_ext_add + assign helper
```



```

-> ctnetlink_setup_nat
-> ctnetlink_parse_nat_setup
-> parse_nat_setup -> nfnetlink_parse_nat_setup
-> nf_nat_setup_info
-> nf_conntrack_alter_reply
-> __nf_ct_try_assign_helper
... and __nf_ct_try_assign_helper will zero the helper again.
Set IPS_HELPER bit to bypass auto-assign logic, its unwanted, just
like
when helper is assigned via ruleset.
Dropped old 'not strictly necessary' comment, it referred to use of
rcu_assign_pointer() before it got replaced by RCU_INIT_POINTER().
NB: Fixes tag intentionally incorrect, this extends the referenced
commit,
but this change won't build without IPS_HELPER introduced there.
Fixes: 6714cf5465d280 ("netfilter: nf_conntrack: fix explicit helper
attachment and NAT")
Reported-by: Pham Thanh Tuyen <phamtyn@gmail.com>
Signed-off-by: Florian Westphal <fw@strlen.de>
Signed-off-by: Pablo Neira Ayuso <pablo@netfilter.org>
Signed-off-by: Sasha Levin <sashal@kernel.org>
---
include/uapi/linux/netfilter/nf_conntrack_common.h | 2 +-
net/netfilter/nf_conntrack_netlink.c                | 3 ++-
2 files changed, 3 insertions(+), 2 deletions(-)
diff --git a/include/uapi/linux/netfilter/nf_conntrack_common.h
b/include/uapi/linux/netfilter/nf_conntrack_common.h
index 4b3395082d15c..26071021e986f 100644
--- a/include/uapi/linux/netfilter/nf_conntrack_common.h
+++ b/include/uapi/linux/netfilter/nf_conntrack_common.h
@@ -106,7 +106,7 @@ enum ip_conntrack_status {
        IPS_NAT_CLASH = IPS_UNTRACKED,
    #endif

-    /* Conntrack got a helper explicitly attached via CT target.
+    /* Conntrack got a helper explicitly attached (ruleset,
    ctnetlink). */
        IPS_HELPER_BIT = 13,

```



```

        IPS_HELPER = (1 << IPS_HELPER_BIT),

diff --git a/net/netfilter/nf_conntrack_netlink.c
b/net/netfilter/nf_conntrack_netlink.c
index 81d03acf68d4d..1c02be04aaf5c 100644
--- a/net/netfilter/nf_conntrack_netlink.c
+++ b/net/netfilter/nf_conntrack_netlink.c
@@ -2310,7 +2310,8 @@ ctnetlink_create_conntrack(struct net *net,
                        if (helper->from_nlattnr)
                            helper->from_nlattnr(helperinfo, ct);

-                        /* not in hash table yet so not strictly
necessary */
+                        /* disable helper auto-assignment for this
entry */
+                        ct->status |= IPS_HELPER;
                        RCU_INIT_POINTER(helper->helper, helper);
                    }
                } else {

```

The above patch has been released since versions linux-5.10.101, linux-5.15.24, linux-5.16.10.

17. References

- [The conntrack-tools user manual](#)
- [Keepalived User Guide](#)
- [systemd](#)

----- OMARINE.ORG -----