Troubleshooting

Table of contents

This section covers the following topics:

- BlueField-3 Jumbo MTU Not Working
- Virtio-net-controller.service Fails to Start
- Function Not Implemented Error When Creating VF
- Guest OS Hangs When Creating VF
- Hotplug Device Does Not Show Correctly in Guest OS
- Hot-unplug Devices with Heavy Self-traffic, Guest OS Gets Call Trace
- Ubuntu Guest OS Hangs with Kernel 5.15.0-88/89-generic

BlueField-3 Jumbo MTU Not Working

Problem

Ping failed with packet size greater than 1500/4000 after configuring jumbo MTU.

Solution

Jumbo MTU is supported starting from the following kernel version:

	Release
Upstream	VM kernel: 4.18.0-193.el8.x86_64 VM Linux version supports big MTU after 4.11.
Ubuntu	DOCA_2.5.0_BSP_4.5.0_Ubuntu_22.04
Virtnet	v1.7 or v1.6.26

The following steps configure jumbo MTU:

1. Change the MTU of uplink representor (or bond) from the BlueField Arm OS:

```
# echo 9216 > /sys/bus/pci/devices/0000:03:00.0/net/p0/mtu
```

2. Restart virtio-net-controller from the BlueField Arm OS:

```
# systemctl restart virtio-net-controlle
```

3. Change the corresponding device MTU on BlueField Arm OS. For example, for the first VF on the first PF, run:

```
# virtnet modify -p 0 -v 0 device -t 9216
```

4. Reload the virtio driver from the guest OS:

```
# modprobe -rv virtio-net && modprobe -v virtio-net
```

5. Verify the VQs' MTU configuration is correct on BlueField Arm OS:

```
# virtnet query -p 0 -v 0 --dbg_stats | grep jumbo_mtu
     "jumbo_mtu":  1
     "jumbo_mtu":  1
```

6. Change the MTU of the virtio-net interface from the guest OS:

```
# echo 9216 >
/sys/bus/pci/devices/0000:af:00.2/virtio0/net/enp175s0f2/mtu
```

Virtio-net-controller.service Fails to Start

Problem

The problem can be verified using the following commands:

```
# virtnet list
ERR: Can't connect to virtnet controller: [Errno 111] Connection
refused
     Check 'systemctl status virtio-net-controller'
     Or controller is not ready to accept commands
```

```
# systemctl status virtio-net-controller
  virtio-net-controller.service - Nvidia VirtIO Net Controller
Daemon
   Loaded: loaded (/etc/systemd/system/virtio-net-
controller.service; enabled; vendor preset: disabled)
   Active: inactive (dead) since Fri 2023-10-27 17:46:59 CDT; 2min
26s ago
     Docs: file:/opt/mellanox/mlnx_virtnet/README.md
  Process: 29652 ExecStart=/usr/sbin/virtio_net_manager
(code=exited, status=0/SUCCESS)
 Main PID: 29652 (code=exited, status=0/SUCCESS)
```

Solution

The problem may happen due to the following reasons.

Virtio-net Not Enabled

1. Check if mlxconfig has `VIRTIO_NET_EMULATION_ENABLE` enabled:

```
# mlxconfig -d 03:00.0 -e q | grep -i
VIRTIO_NET_EMULATION_ENABLE
*        VIRTIO_NET_EMULATION_ENABLE                    False(0)
True(1)          True(1)
```

   Both 2 and 3 columns should appear as `true`.

2. If they are not, perform the following from the BlueField Arm side:

```
# mlxconfig -d 03:00.0 s VIRTIO_NET_EMULATION_ENABLE=1
```

3. Perform a BlueField system-level reset as documented in the BlueField software documentation.

Not Enough SFs Reserved

This can happen when more `VIRTIO_NET_EMULATION_NUM_PF` are reserved than `PF_TOTAL_SF`, as each virtio-net PF/VF requires a corresponding SF created:

```
# mlxconfig -d 03:00.0 -e q | grep -iE
'PF_TOTAL_SF|VIRTIO_NET_EMULATION_NUM_PF'
*        VIRTIO_NET_EMULATION_NUM_PF                    0
    4                4
*        PF_TOTAL_SF                                    0
    8                8
```

Info

By default, the BlueField creates an SF for each PF. Take this into consideration when reserving `PF_TOTAL_SF`.

Function Not Implemented Error When Creating VF

Problem

Creating a virtio-net VF returns an error from the command line:

```
# echo 3 > /sys/bus/pci/drivers/virtio-pci/0000:41:00.2/sriov_numvfs
write error: Function not implemented
```

The host-side dmesg shows the following:

```
[  301.204661] virtio-pci 0000:41:00.2: Driver doesn't support SRIOV
configuration via sysfs
```

Solution

Virtio SR-IOV is only supported starting from the following kernel version:

	Release
Upstream	4.18 with commit cfecc2918d2b3
Ubuntu	Ubuntu-hwe-4.18.0-9.10_18.04.1
CentOS	3.10.0-957.el7 / 7.6.1810

Guest OS Hangs When Creating VF

Problem

The following command from the hypervisor hangs:

```
# echo 100 > /sys/bus/pci/drivers/virtio-
pci/0000:89:00.4/sriov_numvfs
```

Solution

This can happen when more `VIRTIO_NET_EMULATION_NUM_PF`/ `VIRTIO_NET_EMULATION_NUM_VF` are reserved than `PF_TOTAL_SF` (`VIRTIO_NET_EMULATION_NUM_PF` + `VIRTIO_NET_EMULATION_NUM_VF` > `PF_TOTAL_SF`) as each virtio-net PF/VF requires a corresponding SF created. Example:

```
# mlxconfig -d 03:00.0 -e q | grep -iE
'PF_TOTAL_SF|VIRTIO_NET_EMULATION_NUM_PF|VIRTIO_NET_EMULATION_NUM_VF'
*        VIRTIO_NET_EMULATION_NUM_VF                    0
126                 126
*        VIRTIO_NET_EMULATION_NUM_PF                    0
4                   4
*        PF_TOTAL_SF                                    0
508                 508
```

Info

By default, BlueField creates an SF for each PF. Take this into consideration when reserving `PF_TOTAL_SF`.

Note

BlueField supports a limited number of SFs. The SF reserved on the BlueField Arm side and host side are not shared. Make sure to remove the SFs reserved on the host side when reserving a large number on the BlueField Arm side.

Hotplug Device Does Not Show Correctly in Guest OS

Problem

After creating a hotplug device from the BlueField side, probing virtio drivers does not create the virtio-net device correctly.

Solution

The problem may happen due to the following reasons.

BAR 0

Possible failure on BAR 0. check dmesg from guest OS for corresponding hotplug BDF:

```
[10.874845] pci 0000:87:00.1: BAR 0: failed to assign [mem size
0x00100000]
```

> **Info**
>
> In this example, the hotplug PCIe BDF is 87:00.1. This value can be retrieved using "`lspci | grep -i virtio`" from the guest OS.

This can be normally resolved by adding "`pci=realloc`" in the Linux command line (grub).

BAR 14/15

Possible failure on other PCIe BAR. Check the dmesg from the guest OS for the corresponding hotplug BDF:

```
[ 2893.484281] pcieport 0000:10:01.0: bridge window [mem 0x00100000-
0x000fffff] to [bus 12] add_size 200000 add_align 100000
[ 2893.484285] pcieport 0000:10:01.0: BAR 14: no space for [mem size
0x00200000]
[ 2893.484287] pcieport 0000:10:01.0: BAR 14: failed to assign [mem
size 0x00200000]
[ 2893.484289] pcieport 0000:10:01.0: BAR 14: no space for [mem size
0x00200000]
[ 2893.484290] pcieport 0000:10:01.0: BAR 14: failed to assign [mem
size 0x00200000]
```

Info

In this example, the hotplug PCIe BDF is 10:01.0. This value can be retrieved using "`lspci | grep -i virtio`" from the guest OS.

- This is mostly due to there being insufficient BAR resources. Try to reduce the PF BAR size by performing the following from the BlueField side:

  ```
  # mlxconfig -d 03:00.0 s PF_LOG_BAR_SIZE=0
  ```

- This can also be caused by the BIOS provider not reserving enough memory. Check the guest OS's dmesg for similar messages for the PCIe bus of the BlueField device:

```
[3.979061] pci_bus 0000:a0: root bus resource [mem 0x41c0800000-
0x41c10fffff window] (9M)
[3.979062] pci_bus 0000:a0: root bus resource [bus a0-bf]
[4.017770] pci 0000:a4:00.0:   bridge window [mem 0x41c0800000-
0x41c0ffffff 64bit pref] (8M)
[4.018243] pci 0000:a4:00.0: BAR 15: no space for [mem size
0x05800000 64bit pref] (88M)
[4.018245] pci 0000:a4:00.0: BAR 15: failed to assign [mem size
0x05800000 64bit pref]
```

- On the host, the prefetchable memory limit of the root bus (`a0`) is only 9 M. This means that all the devices under this bus (including BlueField) can only be allocated 9M prefetchable memory in total.
- The BAR 15 is the total prefetchable memory limit on the bridge (`a4`) of the device. The PCI bridge window of the BlueField for prefetchable memory is 8M, but the bridge requires 88M for its child device (BlueField). A fter several attempts, the PCIe bridge did not find sufficient IO memory to allocate for BlueField BARs. This can be solved by contacting the BIOS provider to provide enough memory to the PCI root.

Rescan

If the the hotplug operation from the BlueField Arm side is performed before the guest OS is up, and the virtio device is not found by the command " `lspci | grep -i virtio` ". Try to rescan from guest OS:

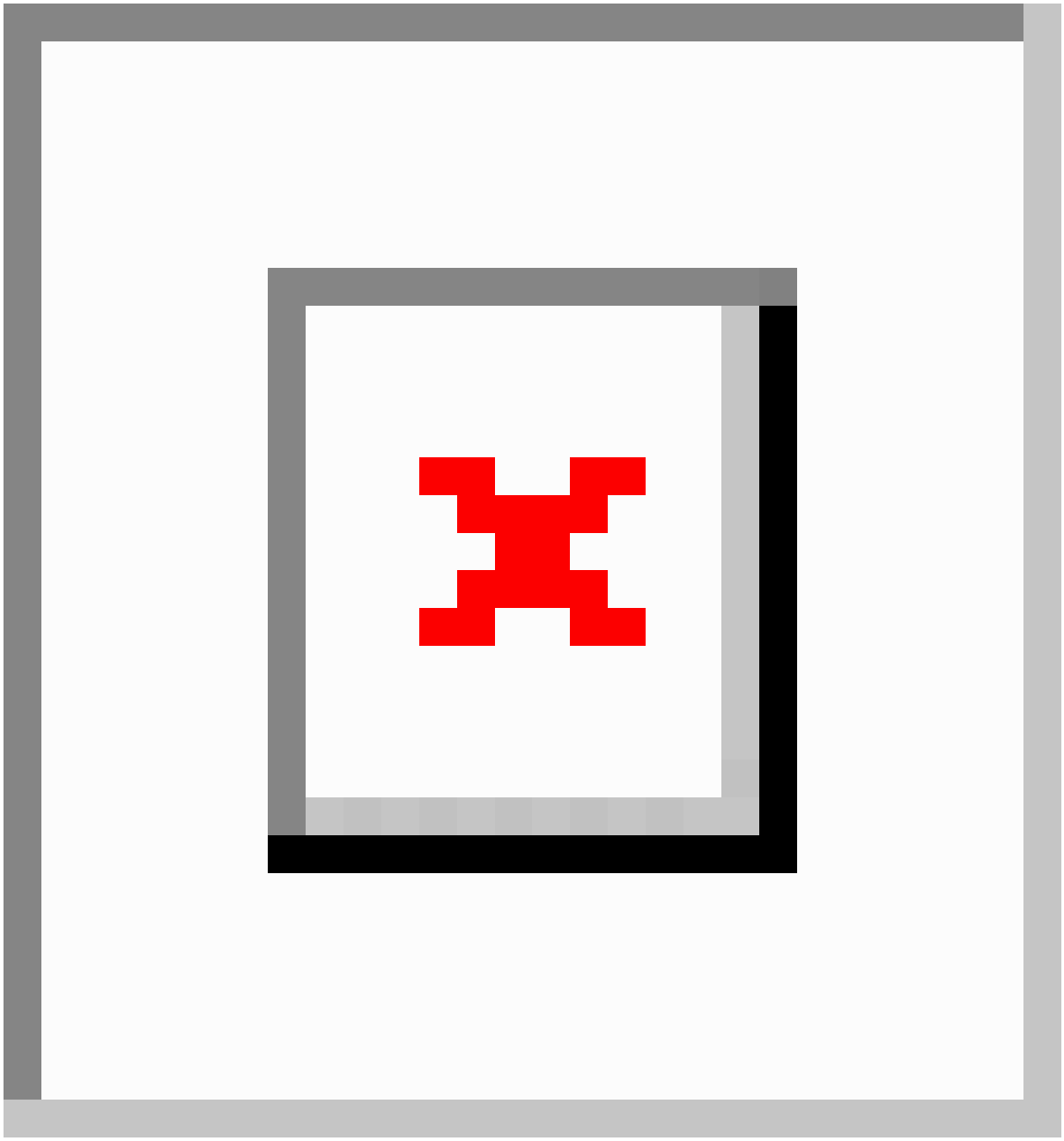
```
# echo 1>/sys/bus/pci/rescan
```

No Hotplug from BIOS

The server BIOS may not support hotplug device. This can be confirmed by looking at guest OS dmesg:

```
[8.209406] acpi PNP0A08:03: _OSC: platform does not support
[PCIeHotplug PME]
```

Try to enable hotplug from the BIOS:



Force Hotplug

Guest OS may be running a kernel older than 4.19, the virtio device is not found by "`lspci | grep -i virtio`". Add the entry `pciehp.pciehp_force=1` to the grub command line.

Hot-unplug Devices with Heavy Self-traffic, Guest OS Gets Call Trace

Problem

When the guest OS is running heavy traffic (e.g., iperf/iperf3) on a hotplug virtio-net device, unplugging those devices from BlueField side at the same time may results in the guest OS hanging.

The guest OS would print a call traffic similar like the following:

Unexpected TXQ (x) queue failure and **sched: RT throttling activated** are usually observed.

```
kernel: pcieport 0000:e2:01.0: pciehp: Slot(1): Card not present
kernel: net ens1f0: Unexpected TXQ (0) queue failure: -5
kernel: sched: RT throttling activated
kernel: rcu: INFO: rcu_sched self-detected stall on CPU
kernel: rcu:         42-....: (1 GPs behind)
idle=39f/1/0x4000000000000000 softirq=388460/388461 fqs=7491
kernel:          (t=15000 jiffies g=2309357 q=7886)
kernel: Sending NMI from CPU 42 to CPUs 25:
kernel: NMI backtrace for cpu 25
kernel: CPU: 25 PID: 491 Comm: irq/71-pciehp Tainted: G
OE     5.15.48 #10
kernel: Hardware name: Dell Inc. PowerEdge R7525/0590KW, BIOS
2.2.5 04/08/2021
kernel: RIP: 0010:native_queued_spin_lock_slowpath+0x74/0x230
kernel: Code: 0f ba 2b 08 0f 92 c2 8b 03 0f b6 d2 c1 e2 08 30 e4 09
d0 a9 00 01 ff ff 0f 85 0f 01 00 00 85 c0 74 0e 8b 03 84 c0 74 08 f3 90
```

```
<8b> 03 84 c0 75 f8 b8 01 00 00 00 66 89 03 5b 41 5c 41 5d 41 5e 41
kernel: RSP: 0018:ffffb94988f97988 EFLAGS: 00000202
kernel: RAX: 0000000000000101 RBX: ffff8c2bc56bba80 RCX:
ffff8c2bd8192800
kernel: RDX: 0000000000000000 RSI: 0000000000000000 RDI:
ffff8c2bc56bba80
kernel: RBP: ffffb94988f979b0 R08: ffff8c2bc7be3000 R09:
ffffffff9fa89ef8
kernel: R10: ffffb94988f979b8 R11: 000000000000001f R12:
0000000000000019
kernel: R13: ffff8c2bc56bba80 R14: 0000000000000000 R15:
ffff8c2bc7be3000
kernel: FS:  0000000000000000(0000) GS:ffff8c339f640000(0000)
knlGS:0000000000000000
kernel: CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
kernel: CR2: 00007ff9c58a8a50 CR3: 0000000dc5010000 CR4:
0000000000350ee0
kernel: Call Trace:
kernel:  <TASK>
kernel:  _raw_spin_lock+0x1f/0x30
kernel:  dev_deactivate_many+0xf3/0x2e0
kernel:  __dev_close_many+0x7d/0x120
kernel:  dev_close_many+0x7f/0x120
kernel:  ? kernfs_put.part.0+0xe2/0x1a0
kernel:  unregister_netdevice_many+0x13a/0x790
kernel:  ? idr_find+0xf/0x20
kernel:  unregister_netdevice_queue+0x91/0xe0
kernel:  unregister_netdev+0x1d/0x30
kernel:  virtnet_remove+0x4d/0x80 [virtio_net]
kernel:  virtio_dev_remove+0x4b/0xa0
kernel:  __device_release_driver+0x1a8/0x290
kernel:  device_release_driver+0x29/0x40
kernel:  bus_remove_device+0xde/0x150
kernel:  device_del+0x19c/0x3f0
kernel:  ? __cond_resched+0x1a/0x50
kernel:  device_unregister+0x18/0x60
```

```
kernel:  unregister_virtio_device+0x18/0x30
kernel:  virtio_pci_remove+0x41/0x80 [virtio_pci]
kernel:  pci_device_remove+0x3e/0xb0
kernel:  __device_release_driver+0x1a8/0x290
kernel:  device_release_driver+0x29/0x40
kernel:  pci_stop_bus_device+0x71/0xa0
kernel:  pci_stop_and_remove_bus_device+0x13/0x30
kernel:  pciehp_unconfigure_device+0x7e/0x130
kernel:  pciehp_disable_slot+0x6c/0x100
kernel:  pciehp_handle_presence_or_link_change+0xde/0x2f0
kernel:  pciehp_ist+0x197/0x1a0
kernel:  ? irq_forced_thread_fn+0x90/0x90
kernel:  irq_thread_fn+0x28/0x60
kernel:  irq_thread+0xde/0x1b0
kernel:  ? irq_thread_fn+0x60/0x60
kernel:  ? irq_thread_check_affinity+0xf0/0xf0
kernel:  kthread+0x12a/0x150
kernel:  ? set_kthread_struct+0x50/0x50
kernel:  ret_from_fork+0x22/0x30
kernel:  </TASK>
```

Root Cause

From kernel 5.14, the following patch introduced a while loop for the virtio-net TX path which may enter infinite when VQ is broken (e.g., device is removed) under heavy traffic:

```
commit a7766ef18b33674fa164e2e2916cef16d4e17f43
Author: Michael S. Tsirkin <mst@redhat.com>
Date:   Tue Apr 13 01:30:45 2021 -0400

    virtio_net: disable cb aggressively

    There are currently two cases where we poll TX vq not in
response to a
```

```
    callback: start xmit and rx napi.  We currently do this with
callbacks
    enabled which can cause extra interrupts from the card.  Used
not to be
    a big issue as we run with interrupts disabled but that is no
longer the
    case, and in some cases the rate of spurious interrupts is so
high
    linux detects this and actually kills the interrupt.

    Fix up by disabling the callbacks before polling the tx vq.

    Signed-off-by: Michael S. Tsirkin <mst@redhat.com>
```

Solution

Currently, there is no official fix from the kernel side, some The following workarounds may be employed:

- Bring down the network interface corresponds to the hotplug device before unplugging it
- Use kernel without the offending kernel patches

Ubuntu Guest OS Hangs with Kernel 5.15.0-88/89-generic

Problem

When probing the virtio-pci and virtio-net kernel modules while running Ubuntu 22.04 with kernel 5.15.0-88/89-generic with any virtio function (i.e, PF or VF), the guest OS hangs and prints call traces as follows:

```
[ 2052.109566] CPU: 0 PID: 1183 Comm: systemd-udevd Tainted: P
O L    5.15.0-88-generic #98-Ubuntu
[ 2052.109568] Hardware name: Red Hat KVM, BIOS 1.15.0-
2.module+el8.6.0+14757+c25ee005 04/01/2014
[ 2052.109570] RIP: 0010:virtqueue_is_broken+0x9/0x20
[ 2052.109579] RSP: 0018:ffffc206423a79c0 EFLAGS: 00000246
[ 2052.109581] RAX: 0000000000000000 RBX: ffff9e8980bfa980 RCX:
0000000000000a20
[ 2052.109582] RDX: 0000000000000000 RSI: ffffc206423a79cc RDI:
ffff9e89847b9000
[ 2052.109583] RBP: ffffc206423a7a60 R08: 0000000000000000 R09:
0000000000000003
[ 2052.109584] R10: 0000000000000003 R11: 0000000000000002 R12:
ffffc206423a79f0
[ 2052.109585] R13: 0000000000000002 R14: 0000000000000004 R15:
ffff9e8984667400
[ 2052.109586] FS:  00007f3e295388c0(0000) GS:ffff9e89bbc00000(0000)
knlGS:0000000000000000
[ 2052.109588] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 2052.109590] CR2: 0000555613432be0 CR3: 0000000116af0002 CR4:
0000000000170ef0
[ 2052.109593] Call Trace:
```

```
[ 2052.109595]  <IRQ>
[ 2052.109598]  ? show_trace_log_lvl+0x1d6/0x2ea
[ 2052.109605]  ? show_trace_log_lvl+0x1d6/0x2ea
[ 2052.109609]  ? _virtnet_set_queues+0xbb/0x100 [virtio_net]
[ 2052.109615]  ? show_regs.part.0+0x23/0x29
[ 2052.109618]  ? show_regs.cold+0x8/0xd
[ 2052.109621]  ? watchdog_timer_fn+0x1be/0x220
[ 2052.109625]  ? lockup_detector_update_enable+0x60/0x60
[ 2052.109627]  ? __hrtimer_run_queues+0x107/0x230
[ 2052.109631]  ? kvm_clock_get_cycles+0x11/0x20
[ 2052.109637]  ? hrtimer_interrupt+0x101/0x220
[ 2052.109640]  ? __sysvec_apic_timer_interrupt+0x61/0xe0
[ 2052.109644]  ? sysvec_apic_timer_interrupt+0x7b/0x90
[ 2052.109650]  </IRQ>
[ 2052.109650]  <TASK>
[ 2052.109651]  ? asm_sysvec_apic_timer_interrupt+0x1b/0x20
[ 2052.109655]  ? virtqueue_is_broken+0x9/0x20
[ 2052.109656]  ? virtnet_send_command+0x105/0x170 [virtio_net]
[ 2052.109660]  _virtnet_set_queues+0xbb/0x100 [virtio_net]
[ 2052.109670]  virtnet_probe+0x4ca/0xa10 [virtio_net]
[ 2052.109674]  virtio_dev_probe+0x1ae/0x260
[ 2052.109676]  really_probe+0x222/0x420
[ 2052.109679]  __driver_probe_device+0xe8/0x140
[ 2052.109681]  driver_probe_device+0x23/0xc0
[ 2052.109683]  __driver_attach+0xf7/0x1f0
[ 2052.109685]  ? __device_attach_driver+0x140/0x140
[ 2052.109687]  bus_for_each_dev+0x7f/0xd0
[ 2052.109691]  driver_attach+0x1e/0x30
[ 2052.109693]  bus_add_driver+0x148/0x220
[ 2052.109695]  driver_register+0x95/0x100
[ 2052.109697]  register_virtio_driver+0x20/0x40
[ 2052.109698]  virtio_net_driver_init+0x74/0x1000 [virtio_net]
[ 2052.109702]  ? 0xffffffffc0d6f000
[ 2052.109704]  do_one_initcall+0x49/0x1e0
[ 2052.109709]  ? kmem_cache_alloc_trace+0x19e/0x2e0
[ 2052.109713]  do_init_module+0x52/0x260
```

```
[ 2052.109716]  load_module+0xb2b/0xbc0
[ 2052.109718]  __do_sys_finit_module+0xbf/0x120
[ 2052.109721]  __x64_sys_finit_module+0x18/0x20
[ 2052.109722]  do_syscall_64+0x5c/0xc0
[ 2052.109725]  ? do_syscall_64+0x69/0xc0
[ 2052.109726]  ? syscall_exit_to_user_mode+0x35/0x50
[ 2052.109729]  ? __x64_sys_newfstatat+0x1c/0x30
[ 2052.109733]  ? do_syscall_64+0x69/0xc0
[ 2052.109735]  entry_SYSCALL_64_after_hwframe+0x62/0xcc
```

Solution

There is a bug in upstream version v6.5-rc4, which is fixed in v6.5-rc7. Canonical backported the problematic patch to Ubuntu 5.15.0-88/89.generic, which triggers this Virtio-net deadlock issue:

```
commit 51b813176f098ff61bd2833f627f5319ead098a5
Author: Jason Wang <jasowang@redhat.com>
Date:   Wed Aug 9 23:12:56 2023 -0400

    virtio-net: set queues after driver_ok

    Commit 25266128fe16 ("virtio-net: fix race between set queues
and
    probe") tries to fix the race between set queues and probe by
calling
    _virtnet_set_queues() before DRIVER_OK is set. This violates
virtio
    spec. Fixing this by setting queues after
virtio_device_ready().

    Note that rtnl needs to be held for userspace requests to
change the
    number of queues. So we are serialized in this way.
```

```
Fixes: 25266128fe16 ("virtio-net: fix race between set queues and probe")
Reported-by: Dragos Tatulea <dtatulea@nvidia.com>
Acked-by: Michael S. Tsirkin <mst@redhat.com>
Signed-off-by: Jason Wang <jasowang@redhat.com>
Signed-off-by: David S. Miller <davem@davemloft.net>
```

Switch default kernel back to another version (e.g., 5.15.0-79-generic).

> **Note**
>
> From 5.15.0-90-generic, the Ubuntu official kernel has the issue fixed.

There are multiple ways to switch the default kernel. The following is only one example:

> **Note**
>
> Users must have root permission before proceeding.

1. Open `/etc/default/grub` and change `GRUB_DEFAULT` as follows:

   ```
   GRUB_DEFAULT=saved
   ```

2. Save file.

3. Run the following to get the number of the kernel you want

```
# grep "menuentry 'Ubuntu," /boot/grub/grub.cfg
```

> **Info**
>
> Numbering starts from 0 (i.e., first entry is 0)

4. Run the following to set the default kernel:

```
# grub-set-default num_from_last_step
```

5. Reboot.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF

ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.