

Virtio as a universal communication format

A study in interface design

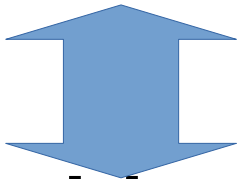
Michael S. Tsirkin

Consulting Engineer
Chair of Virtio TC



Terminology



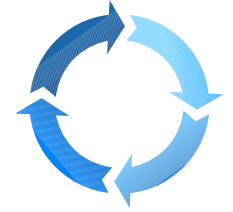
- Virtion: Asymmetrical two-party interface
 - Driver (AKA **virtio driver**)
submits requests by making them **available**
 - Kernel driver, userspace process, firmware ...
- 
- Device (AKA **vhost driver**)
uses (processes) requests
 - Hypervisor, kernel, another process, PCI device ...

The inexplicable popularity of virtio

- Started around 2007 by Rusty Russell
Guest/Hypervisor interface for VMs
- 2010 vhost: userspace/kernel interface
- 2012 virtio multimedia hardware offload
"cool and random" – Rusty
- 2014 vhost/virtio-user: userspace/userspace
- 2017 vdpa: hardware interface



Network effects



SOFTWARE



DPDK
DATA PLANE DEVELOPMENT KIT

SPDK

FIRMWARE



SLOF



SCSI

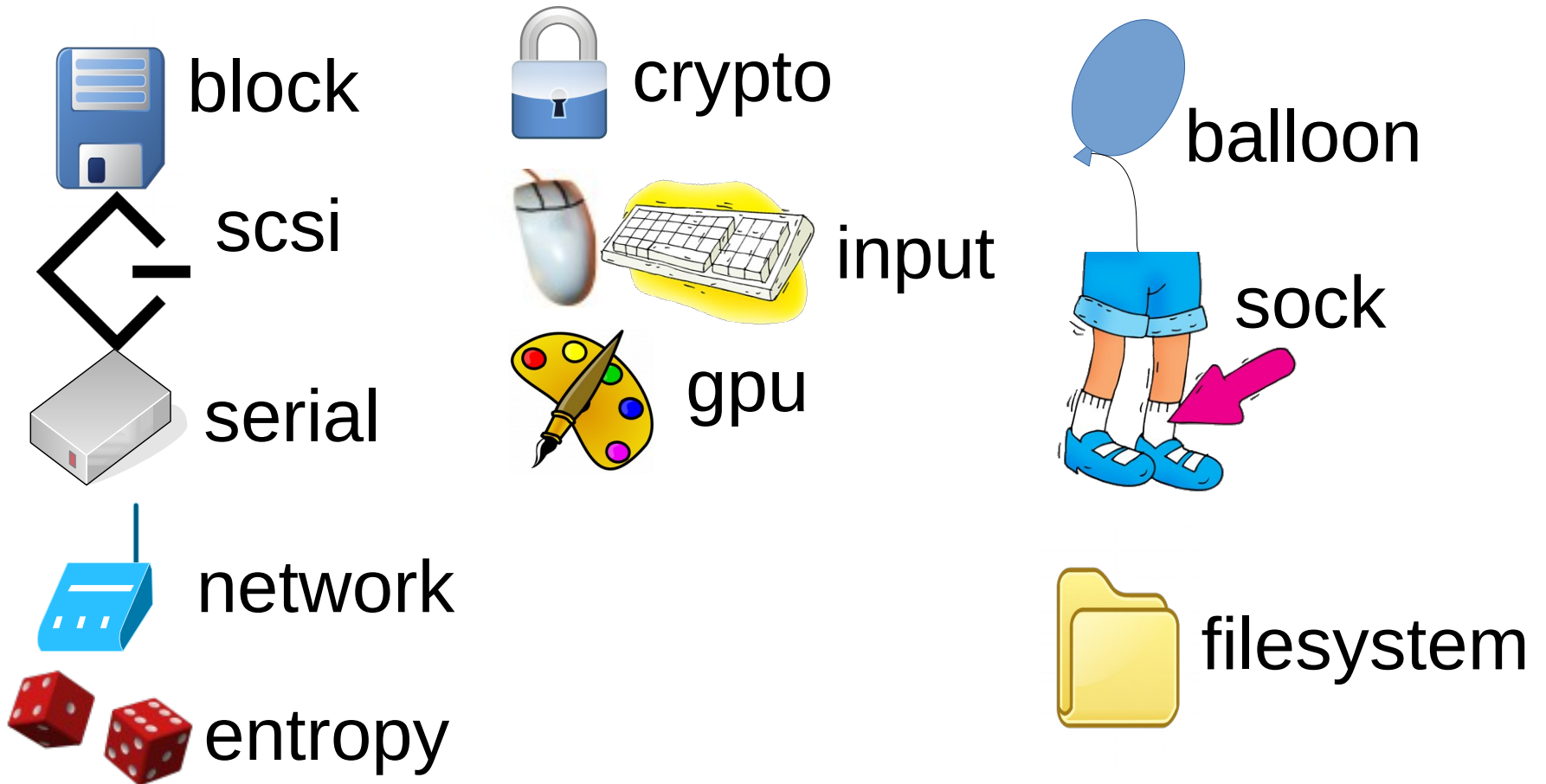


HARDWARE

VDPA

Mmedia

Virtio Interface Zoo



Standards are good!

Motivation: userspace drivers

- Drivers often packaged with application

Unlike kernel: New devices require app

- Kernel has no visibility into device state



- Link with a virtio library and forget
- Snapshot/restore can be made to work (WIP)

Motivation: VM guests

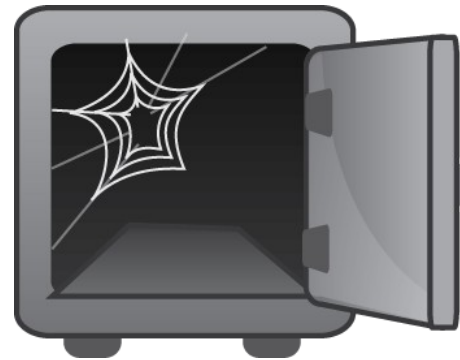
- Pass-through for performance
- Cross-host migration without guest changes
- Multi-vendor clusters supported
- Live migration also works

Hypervisor aware of guest visible



Motivation: overcommit

- Hardware
- Memory



- Switching to software
- Possibly live (WIP)

Motivation: bugs

- Who's to blame for a crash?
Buggy card or buggy driver?
- Swap in a different device and find out!
- Software implementations available
- Fix it in the right place



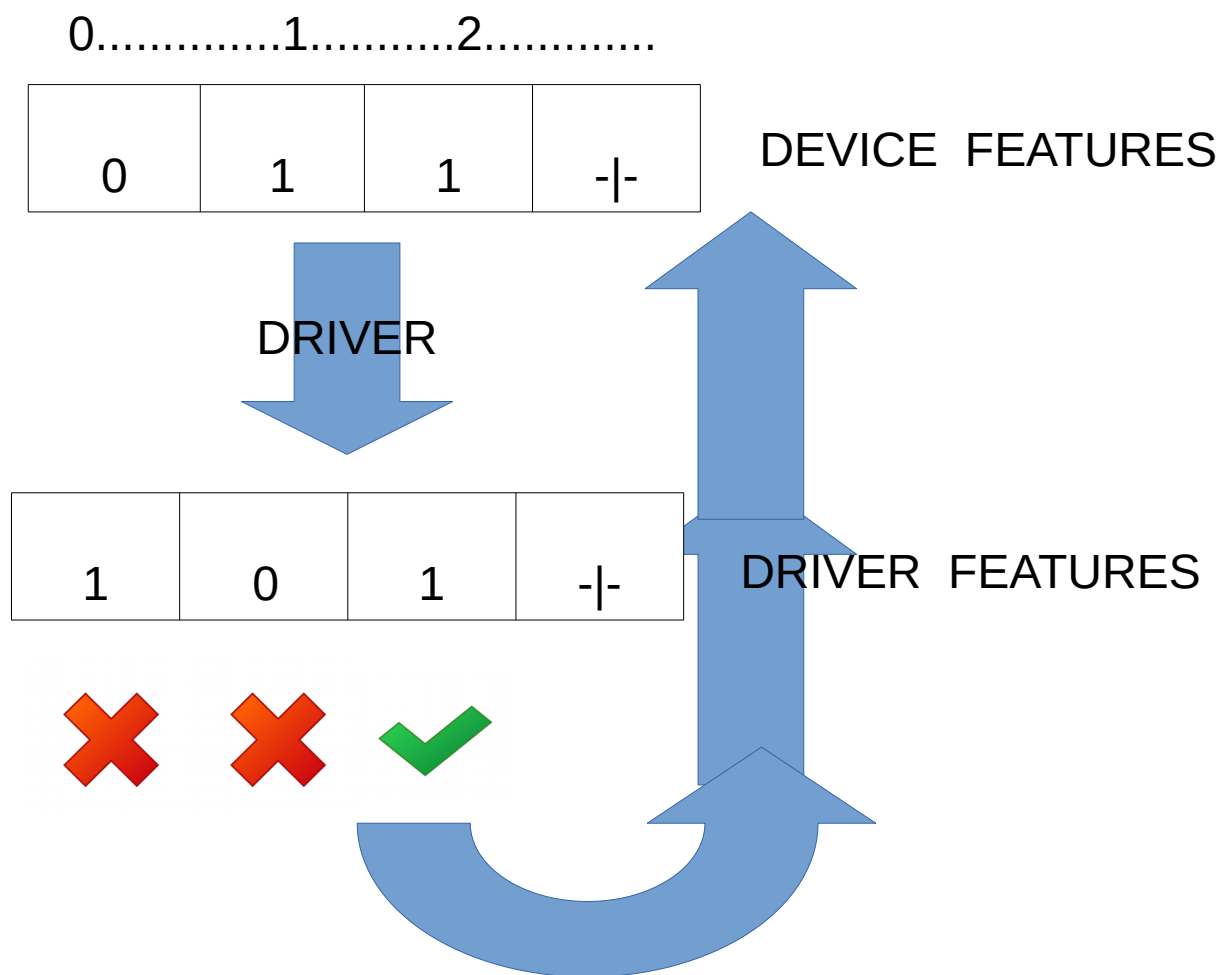
Virtio Properties



- Forward and Backward Compatibility
- PCI for Device Discovery
- Virtqueue Communication
- Reasonable Specification Process

Let's drill down ...

Virtio feature negotiation



Virtio net: add failover support

- Feature bit: `VIRTIO_NET_F_STANDBY = 0`
- New (failover aware) device: `device features = 0x1`
- New driver: `supported features = 0x1`
- Driver features: `0x1 & 0x1 = 0x1`
- Device and driver:

```
if (driver_features &  
    (1 << VIRTIO_NET_F_STANDBY))  
    enable failover support
```

- Updated device & driver: failover enabled!

Compatibility: existing drivers

- Device features = 0x1
- Driver supported = 0x0
- Driver features = 0x0
- $0x0 \ \& \ (1 \ll \text{VIRTIO_NET_F_FAILOVER}) == 0$
- Device: option 1: disable failover: compatible!
- Device: option 2: set status = fail
Not worse than building a new device!
Can suggest upgrading a driver.



Compatibility: existing devices

- Device features: 0x0
- Driver supported: 0x1
- Driver features: 0x0
- $0x0 \& (1 \ll \text{VIRTIO_NET_F_FAILOVER}) == 0$
- Driver: option 1: disable failover
- Driver: option 2: set status = fail
Can suggest upgrading a device.



Compatibility: virtio 0.9 versus 1.0

- virtio 1.0 – made default Jul 2016
- Switched devices to a different register layout
- Gated by a feature bit:

```
/* v1.0 compliant. */
```

```
#define VIRTIO_F_VERSION_1          32
```

- No one noticed!



PCI based discovery

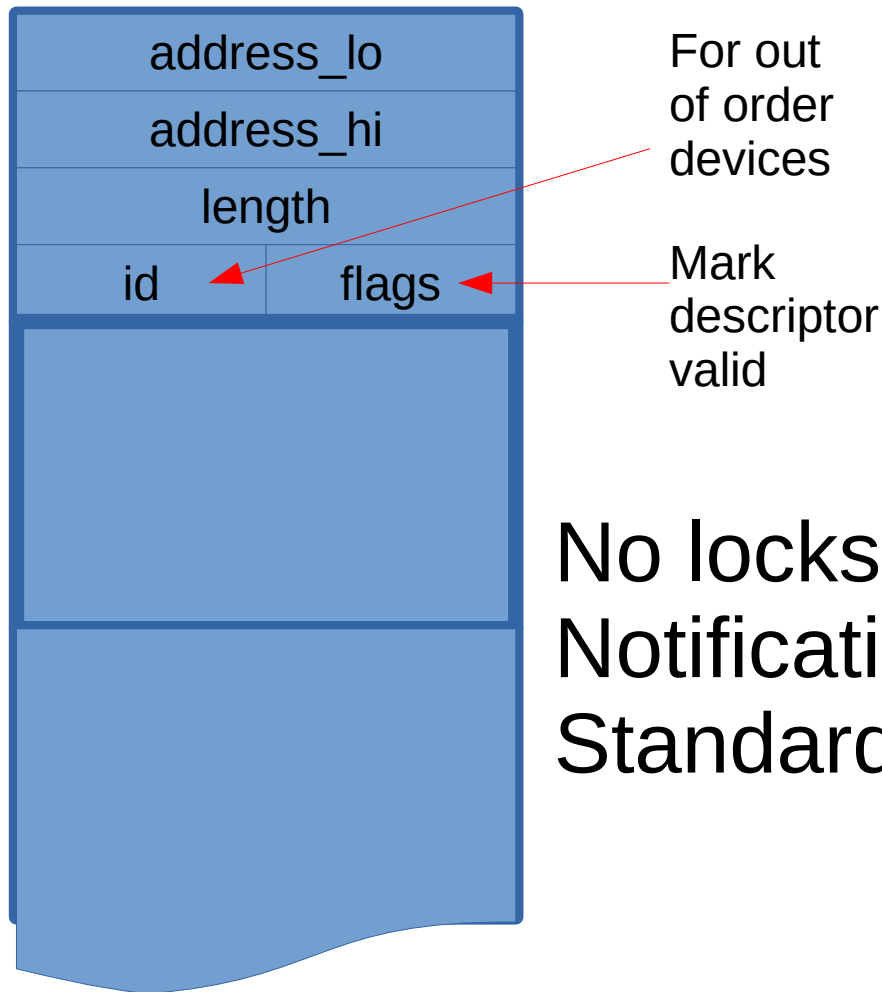


- Not the only option
multiple transports supported
- Standard VendorID/DeviceID registers
donated by Red Hat for use by Virtio
- Use these values → drivers will bind to device



Virtqueue ring

Device and driver write descriptors into a ring



No locks shared
Notifications identify the ring
Standard for DMA HW

Specification process

do I have to write a spec?

- Absolutely the right thing to do
- Does not have to be step 0!



- Virtio priorities:
 - Code compatibility
 - IPR compatibility
 - Interface compatibility

Code compatibility: avoid conflicting with others



- New device: reserve an ID. Spec patch:

```
diff --git a/content.tex b/content.tex
@@ -3022,3 +3022,5 @@ Device ID & Virtio Device  \\
\hline
+23      &  misc device \\
+\hline
\end{tabular}
```

- Existing device: reserve a feature bit. E.g. :

```
@@ -4800,5 +4802,6 @@ guest memory statistics
\item[VIRTIO_BALLOON_F_DEFLATE_ON_OOM (2) ] Deflate balloon on
  guest out of memory condition.
+\item[VIRTIO_BALLOON_F_XXXX (3) ] Reserved for
+  feature XXXX.
\end{description}
```

How to get it in the spec?

- git clone <https://github.com/oasis-tcs/virtio-spec>
Edit :)
- sh makeall.sh (needs xelatex, e.g. from texlive)
- virtio-comment-subscribe@lists.oasis-open.org
- Patch: virtio-comment@lists.oasis-open.org
- If no comments – email, ask for a vote ballot
- Total time: up to 2 weeks



IPR compatibility: allow others to implement compatible devices

- Open-source an implementation
- Subscribe to virtio-dev@lists.oasis.org
- Agree to IPR rules (non-assertion mode)
- Send a copy of the patches (e.g. qemu, linux, dpdk) to virtio-dev@lists.oasis.org
- Virtio memory and IOMMU at this point now.

Interface compatibility



- Document assumptions for inter-operability
- Submit as comments
- Virtio membership is not required
- Membership is open - members vote on ballots
- Hints:
 - Document device and driver separately
 - Use MUST/SHOULD/MAY keywords
 - Ask for help!
- Virtio crypto, input, gpu added recently

Work-in-progress



- Platform and hardware specific optimizations
- Vendor-specific issues
- New transports
- New devices

Hardware is special



- Let's assume a pass-through device implementing virtio. Shouldn't this just work?
- Maybe – but not optimally!
- Hypervisor: processes descriptors one by one
- Hardware: can process many in parallel
- Needs to be told how many are available
- Include number of available entries in a kick

Platform issues

- Hardware Virtio device behind a PCI bus:

wmb()

dma_wmb()

- Software Virtio device:

interrupt

smp_wmb()



Cross-vendor compatibility

- Modular interface controlled by feature bits
- Drivers can limit to a subset for consistency
- Report negotiated features:
 - `cat /sys/bus/pci/devices/0000\:01\:00.0/features`
- TODO: report device features



Device quirks

- Don't do it!
- Mask affected features
- Treat it as a feature



Document in spec

- Blacklist device, use a vendor-specific driver

CFA: transports



- Vhost-user: virtio over unix domain sockets
 - QEMU
 - DPDK
 - SPDK

WIP: devices



- Memory device
- IOMMU device
- 9PFS?
- Audio anyone?

MST's inbox



- Balloon: page hinting capability
- GPU: EDID reporting
- Network: RSC
- Block: discard+write zeroes

Virtio 1.1 plans



- Freeze spec by end of November 2018
- Public review draft by end of year
- Public review to run until early next year
- Monthly draft snapshots planned

Summary

- Network effects and a set of unique properties make Virtio a compelling option for new interfaces
 - Large Software and Hardware ecosystem
- Join the fun
 - Easy to extend
 - A lot going on
 - Performance + new features



Questions?



Virtio input: add multitouch feature



- Feature bit: `VIRTIO_INPUT_F_MULTITOUCH` = 0
- New (multi-touch aware) device: device features = 0x1
- New driver: supported features = 0x1
- Driver features: 0x1 & 0x1 = 0x1
- Device and driver:

```
if (driver_features &
    (1 << VIRTIO_INPUT_F_MULTITOUCH))
    enable multi-touch support
```

- Updated device & driver: multi-touch enabled!

Compatibility: existing drivers

- Device features = 0x1
- Driver supported = 0x0
- Driver features = 0x0
- $0x0 \ \& \ (1 \ll \text{VIRTIO_INPUT_F_MULTITOUCH}) == 0$
- Device: option 1: disable multi-touch: compatible!
- Device: option 2: set status = fail
Not worse than building a new device!
Can suggest upgrading a driver.



Compatibility: existing devices

- Device features: 0x0
- Driver supported: 0x1
- Driver features: 0x0
- $0x0 \ \& \ (1 \ll \text{VIRTIO_INPUT_F_MULTITOUCH}) == 0$
- Driver: option 1: disable multi-touch
- Driver: option 2: set status = fail
Can suggest upgrading a device.

