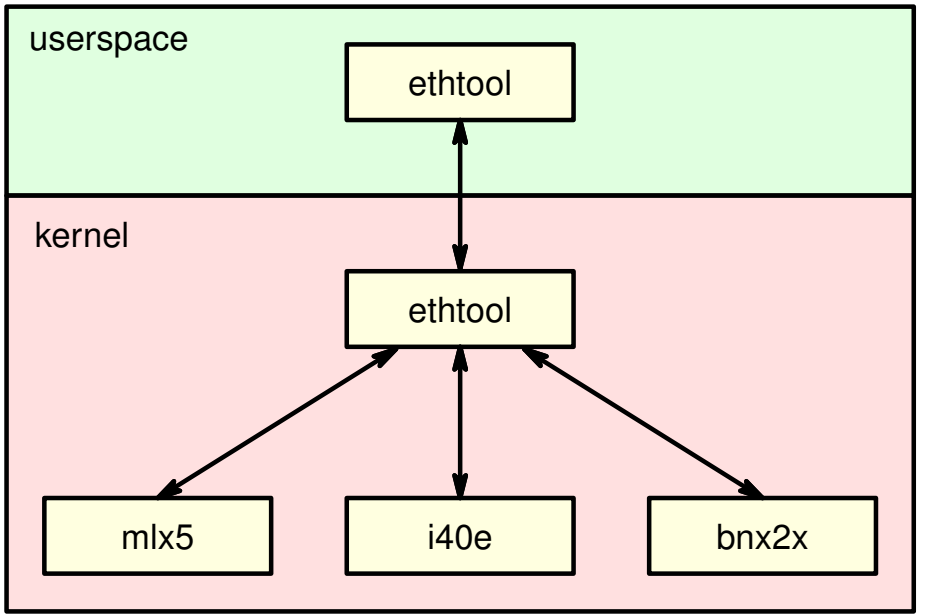
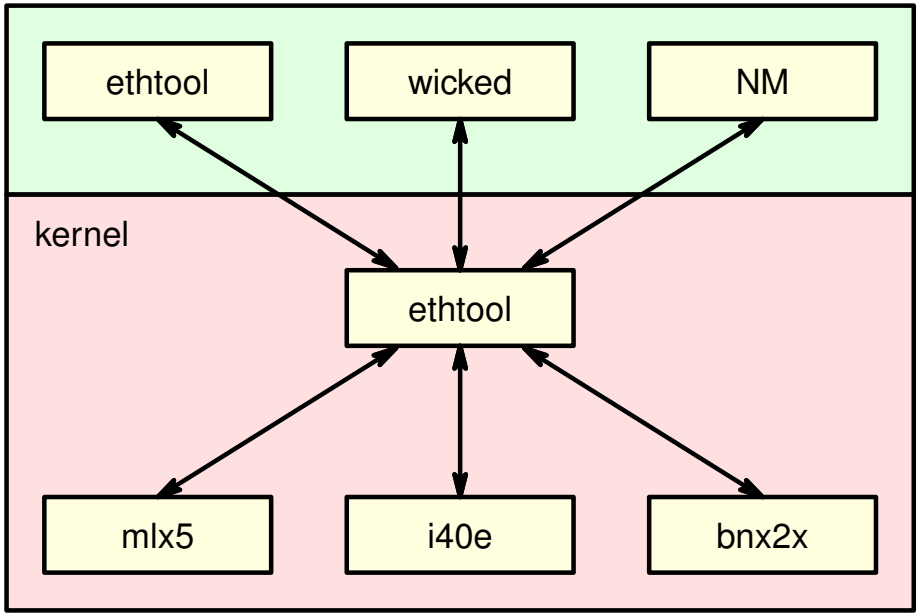
ethtool: road to netlink

Michal Kubeček
SUSE Labs
mkubecek@suse.cz

ethtool interfaces



ethtool interfaces



netlink API goals

- easy to extend with new attributes
- arbitrary size bit sets
- no fixed size strings
- extack error reporting and warnings
- no read/modify/write between userspace and kernel
- netlink notifications on changes (both netlink and ioctl)
- reduced need for kernel and userspace to be in sync
- API documentation

phase 1

- the goal of “phase 1” is ioctl-less ethtool
- all ethtool functionality available via netlink
- some features through other interfaces (rtnetlink, devlink)

- fall back to ioctl if netlink is unavailable
- compatibility with old kernels
- also fall back if feature not implemented via netlink
- no need to implement everything at once

phase 1 status

- RFC submissions
- v5 submitted in March 2019
- only first part: infrastructure and few messages
- some serious attention
- lot of changes requested

- rewriting the code to address requests from v5 review
- mostly done, hopefully v6 can be ready within this cycle
- once infrastructure is in, further progress should be easier‘

message types (v6)

- two enums, one for userspace messages, one for kernel
- prefixed with `ETHTOOL_MSG_` (`UMSG`/`KMSG`?)
- `ETHTOOL_MSG_*_{GET,SET,ACT}` for userspace requests
- `ETHTOOL_MSG_*_{GET,SET,ACT}_REPLY` for kernel replies
- `ETHTOOL_*_NTF` for kernel notifications
- `_SET`, `_GET_REPLY` and `_NTF` share the structure
- use nested attributes to structure the information

phase 1 status

Good news is that it is easier to list what is missing than what is ready. (In the complete series, that is.)

So what is missing?

- statistics and phy statistics
- tests
- per queue coalescing
- Rx classification rules management
- tunables and phy tunables
- dumps and flashing

dynamic name lists

- some name lists are provided by kernel already (e.g. features)
- device specific lists (stats, private flags, ...) must be dynamic
- some name lists are kept on both sides (link modes, timestamping, ...)
- they keep getting out of sync

- proposed interface makes more name lists dynamic
- allows requests like `"ethtool -s eth1 advertise new_mode on"` without updating ethtool
- objections raised

tunables and PHY tunables

- generic framework for assorted parameters
- global list of tunables with their types
- needs to be in sync between kernel and userspace

Would be easy to implement in netlink – but it's not clear if we need them. The concept looks like a workaround for inability to extend ioctl structures. Shouldn't we rather add new attributes where they belong?

What would make sense would be device/driver specific tunables (in a sense, an extension of private flags).

Is it a good idea, though? Do we want to encourage vendors to maintain their driver specific API?

dumps and flashing

Most of these features relate to a physical device rather than to a network device.

In other words, they should rather belong to devlink.

Register dump is a special case, it seems to mix registers of the physical device with register specific for a port.

To make things worse, current interface (both ioctl and `ethtool_ops`) passes an opaque block of binary data rather than a list of registers.

Maybe it's an opportunity to introduce a new interface for proper register dump and gradually migrate drivers to it. We could also split registers between devlink and ethtool.

ethtool and devlink

Some of the features provided by ethtool should rather belong to devlink.

Question: at which level to do that?

(A) Generic ethtool code in kernel calls devlink code.

Problem: duplicates the kernel–userspace interface.

(B) Userspace ethtool uses devlink.

Problems: needs 2 (3) sockets, limited devlink support in NIC drivers (4 vendors, 6 drivers).

(C) Only support these features in devlink utility.

People won't like it (or use it), limited support in NIC drivers.

ethtool and devlink

The main problem is that devlink support is currently limited only to few enterprise NIC drivers. It is not realistic to expect all NICs currently providing ethtool ops to have (native) devlink support any time soon (if ever).

Idea of a solution (Jiří Pírko): provide generic implementation of basic devlink functions to be used by drivers (probably on top of existing ethtool ops).

- this would allow to leave devlink features in devlink
- ethtool could use devlink even with devices without native devlink
- for NICs where it makes sense, we could replace by actual devlink later

error reporting

- ethtool netlink interface uses extack for errors and warnings
- allows useful error/warning messages
- at the moment, only for issues detected by generic ethtool code
- no way to pass the message from driver up

error reporting

- ethtool netlink interface uses extack for errors and warnings
- allows useful error/warning messages

- at the moment, only for issues detected by generic ethtool code
- no way to pass the message from driver up

- solution: pass extack to `ethtool_ops` handlers
- make drivers use it (the hard part?)
- reduce the number of log messages from ethtool ops
- be more strict about adding new ones

- should this wait for phase 2?

RTNL lock

With ioctl interface, all ethool request processing is done under RTNL lock.

- makes things easier at the expense of unnecessary serialization
- not only within ethtool but also with other code using RTNL lock

- netlink interface takes RTNL lock but tries to minimize the scope
- it would be nice to get away without it like devlink does

- difference: devlink is new, it can define its locking scheme
- ethtool has to live with existing ethtool ops handlers

`->begin()` **and** `->complete()`

- if defined, must be called before/after other ethtool ops
- only few drivers define them
- mostly seem to deal with power management

- problem: no handling of nesting these
- ioctl is safe as it calls everything under RTNL lock
- not fully: physical identification (blinking) releases RTNL lock
- so does recently introduced devlink callback

- solution 1: keep depth counter in `struct net_device`
- solution 2: keep depth counter in driver
- solution 3: could we get rid of these?

potential future issues

Most bitmaps in proposed interface use arbitrary size bitsets even if kernel backend uses 32-bit one in some cases. However, there are still some places where `nla_bitfield32` is still used:

- wake on LAN modes (8)
- message "level" (15)
- FEC modes (5, messy)
- Rx hash fields (8)

Can we rely on these staying within 32 bits forever?

And how abou some numeric values? E.g. link speed of 4 Tb/s sounds like a lot but 400 Gb/s ethernet is getting closer.

after phase 1

- selftests – netdevsim ethtool ops (preferrably earlier)
- no more extensions to ioctl interface
- `CONFIG_ETHTOOL_IOCTL` :-)
- convince other userspace tools to use netlink
- rework `ethtool_ops`: extack support, extensibility
- features unfeasible or not worth the effort with ioctl
- structured statistics
- structured offloading flags
- get rid of RTNL (problem: `ethtool_ops`)
- wireshark dissector?

links

WiP kernel and ethtool patchset:

`https://github.com/mkubecek/ethnl`

SUSE LabsConf 2018 presentation:

`https://www.youtube.com/watch?v=8B5mu9d5dII`