Driver and H/W APIs Workshop

Agenda

Challenges supporting multiple S/W stacks - David

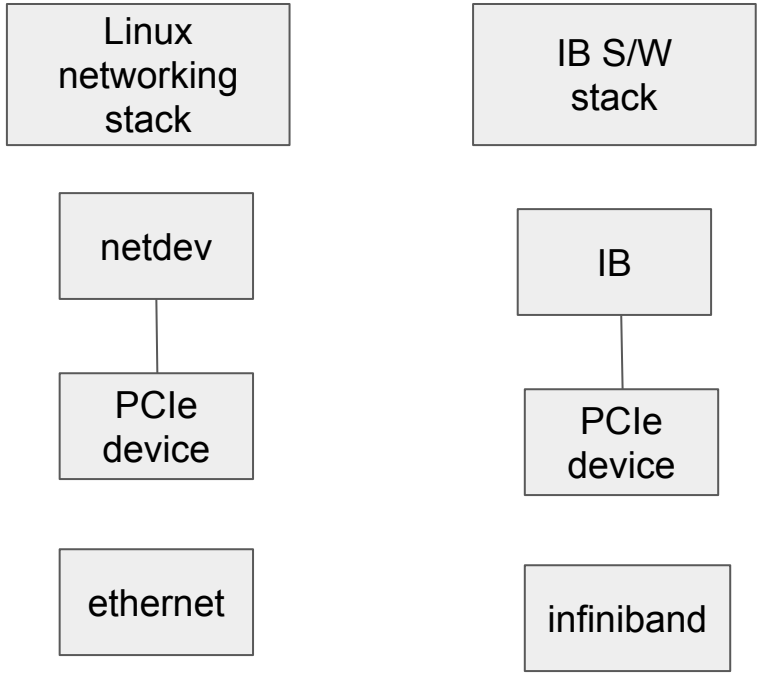
Multi-PF device - Jacob Keller, Jiri Pirko

Extensions to PHC APIs for PTP timers - Maciek Machnikowski

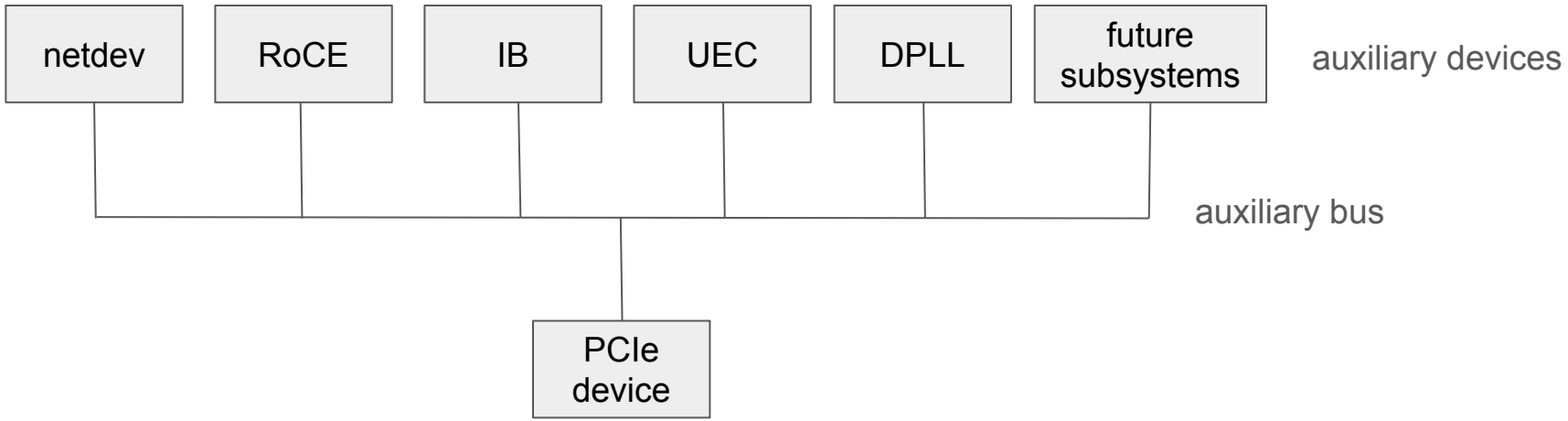
fwctl - David

Challenges with Multi-S/W stack Support

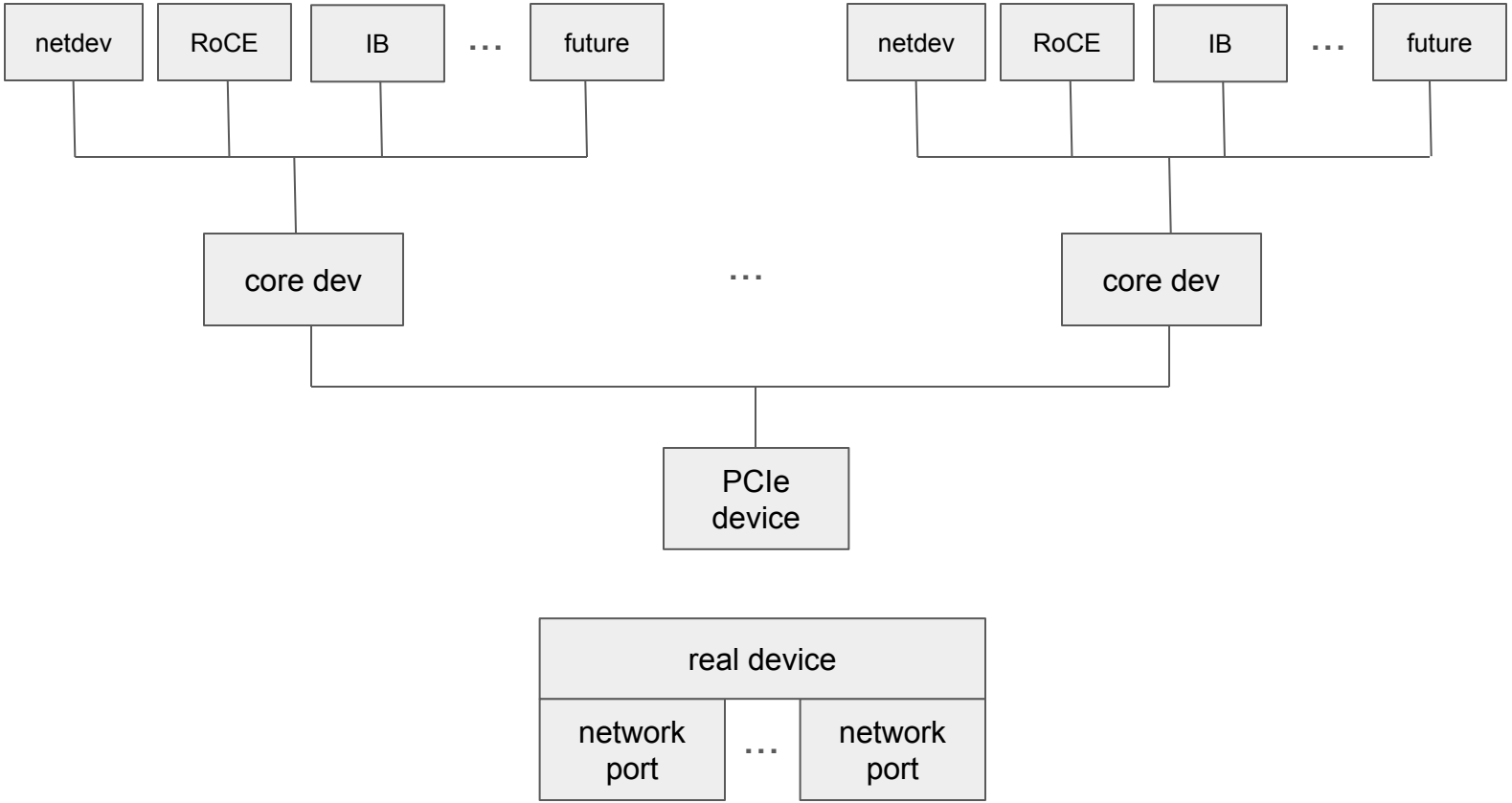
Legacy Devices



Current Devices



Device view


netdev
RoCE
IB
...
future
netdev
RoCE
IB
...
future
core dev
...
core dev
PCIe device
real device
network port
...
network port

Working Across S/W Stacks

- Holistic view of all H/W resources in use
  - Not by silo (S/W stack), but all of them at once - with metadata showing stack
  - S/W view and H/W view
- Device specific details can vary across vendors
  - New H/W is developed for reasons
  - Inevitable that there are vendor unique features or details
  - Vendors are trying to be more open but there are roadblocks
  - Saying “out-of-tree” until some feature or configuration knob is “standardized” only hurts users
- Introspection filters
  - process, uid, virtual device, PCI device

devlink

- An API at the PCI device layer
  - Seems logical to use across S/W stacks
- Legacy is a netdev focus
  - ALL code changes go through netdev and its maintainers’ lens and view of the world
- Crossing S/W subsystems means support for other subsystems' details
  - Kuba has made no secret of his disdain for Infiniband. Conflict wrt what constitutes a legitimate feature or change?
- Ready to expand devlink to RDMA / IB concepts?
  - Memory regions, domains, queue details - and queues used outside of netdev
  - all flow steering rules
  - vendor specific functionality

Multi-PF device

Extensions to PHC APIs for PTP timers

fwctl

Realities

- It is **2024** not **2004**; Linux is the dominant DC OS, not “a hobbyist OS”
  - We should keep in mind the openness to ideas that got us here
- Linux and its ecosystems thrive when we are optimistic about possibilities
  - Everything must evolve to survive
- Linux is about choices
  - Focus on creating solid primitives / building blocks with well defined interfaces
  - Allow them to be put together in a way that people decide what overhead they want and what they do not

Realities

- Linux is used by and driven by businesses
- Many established, entrenched camps - not going to change
  - netdev / socket API, RDMA, DPDK (userspace stacks), ...
- OOT drivers - established and forced in so many ways
  - OOT changes are fore real use cases, real problems
  - Existence of OOT modules is not helpful to users, vendors or the growth and development of Linux
- H/W vendors are not going to open source their firmware and device designs
  - Devices have differences; S/W needs to acknowledge and deal with it

Kernel APIs that Enable Varying Degrees of Bypass

- Device: /sysfs, UIO and VFIO
- Networking: RDMA, OVS, ebpf, XDP, AF_XDP, userspace stacks (DPDK)
- AF_XDP
  - Bypasses Linux networking stack for datapath - packets do not traverse the stack
  - Deemed acceptable by netdev maintainers why? Because it uses networking APIs and ndos for control and some level of monitoring
  - Some heavy rationalization that it is a step in the right direction as it involves more standard APIs and code

What is fwctl?

- New subsystem intended to bring some common rules and order to the growing pattern of exposing a secure FW interface directly to userspace
- Focus on debugging, configuration, and provisioning
  - Vendor specific details
- Define and document the rules that a device must follow to expose a compatible sysfs style RPC for a locked down kernel

fwctl

- Move in the direction of **common code** to best extent possible
  - Very similar to the AF_XDP argument
  - Open source driver, open source userspace tooling
  - Device specific passthrough for device specific commands
- Decoupling from a given S/W stack and its abstractions
- Allows self-documenting design for tunables
- No delay between firmware release and usability of some knob
  - No waiting for changes to propagate out to kernels, distros, ... == huge benefit for users
- Other domains have similar needs - e.g., CXL, NVMe

fwctl

- Pushback along the lines of “devlink” or other standard API will suffer is a strawman argument
  - Linus has deflected such reasoning as well. See ebpf scheduler response
- Users and Linux ecosystem are better off with in-tree code that everyone can review and work on
  - Another explicit comment from Linus

https://lore.kernel.org/all/CAHk-=wg8APE61e5Ddq5mwH55Eh0ZLDV4Tr+c6_gFS7g2AxnuHQ@mail.gmail.com/

Existing devlink param

```
$ devlink dev param
pci/0000:0b:00.0:
   name io_eq_size type generic
    values:
      cmode driverinit value 1024
  name event_eq_size type generic
    values:
      cmode driverinit value 4096
  name flow_steering_mode type driver-specific
    values:
      cmode runtime value smfs
  name fdb_large_groups type driver-specific
    values:
      cmode driverinit value 15
name esw_port_metadata type driver-specific
  values:
    cmode runtime value true
name esw_multiport type driver-specific
  values:
    cmode runtime value false
name hairpin_num_queues type driver-specific
  values:
    cmode driverinit value 2
name hairpin_queue_size type driver-specific
  values:
    cmode driverinit value 1024
```

mlx5 Example of Current Tunables

Download mft tools and install

- https://network.nvidia.com/products/adapter-software/firmware-tools/

sudo mst start

sudo mlxconfig -a -d /dev/mst/<dev entry> q

long list of tunables

sudo mlxconfig -a -d /dev/mst/<dev entry> i

Detailed description of tunables

Detailed View of What is Needed vs devlink

(see command outputs)