Layer 1 boring stuff and less Layer 1 yet still boring stuff

Florian Fainelli <f.fainelli@gmail.com>

PHY testing

- Electrical testing:
  - Cable: short/open pairs, distance
  - Voltage/waveforms measurements
  - Typically disruptive to the link state
- Requirements:
  - Must run for an undefined amount of time
  - Must communicate the disruption to the user clearly
  - Future proof interface (as much as possible)
  - Might have input/output data

PHY testing

- Initial candidates:
  - IEEE compliance tests: waveforms, no I/O data
  - Cable diagnostics: Output data
  - Packet generator/tester: Input data
- None of this are killer features, yet it’s important for lots of product manufacturers

PHY testing

- ETH_SS_PHY_TEST:
  - String set of supported PHY test modes
- ETHTOOL_GPHYTEST:
  - Obtain PHY specific test (with optional data) results
- ETHTOOL_SPHYTEST:
  - Enter PHY specific test (with optional data)

PHY testing

- New ethtool_test_flags:
  - ETH_TEST_FL_DONE: test is completed (success)
  - ETH_TEST_FL_START: request test start
  - ETH_TEST_FL_STOP request test stop
  - ETH_TEST_FL_RUNNING: test is running/in progress

PHY testing

```
struct ethtool_phy_test {
        __u32   cmd;
        __u32   flags;
        __u32   mode;
        __u32   len;
        union {
                __u8 data[0];
        } test_data;
};
```

PHY testing

- In kernel API:
  - int (*get_phy_test_len)(struct net_device *,
  - struct ethtool_phy_test *);
  - int (*get_phy_test)(struct net_device *,
  - struct ethtool_phy_test *, u8 *);
  - int (*set_phy_test)(struct net_device *,
  - struct ethtool_phy_test *,
  - const u8 *);

PHY testing

- User: Enter a test mode:
  - Set appropriate mode value
  - Set ETH_TEST_FL_START
- Kernel:
  - Verifies parameters
  - Calls appropriate PHY driver function
  - Set net_device::operstate = IF_OPER_TESTING
- Success:
  - Set ETH_TEST_FL_RUNNING/DONE
- Failure:
  - Set ETH_TEST_FL_FAILED
  - Reset PHY and restart PHY state machine

PHY testing

- User: exit a test mode
  - Set appropriate mode value
  - Set ETH_TEST_FL_STOP
- Kernel:
  - Verifies parameters
  - Calls appropriate PHY driver function
  - Clears ETH_TEST_FL_{DONE,RUNNING,START}
  - Resets PHY and restarts state machine
    - (implicitly sets net_device::operstate to a correct value)

PHY testing

- Open questions:
  - Generalize in-kernel API?
    - get_test_len(net_device *, u32 stringset, union { })
  - ethtool → netlink timeframe?
  - Less structured data?
    - NL80211_CMD_TESTMODE + NL80211_ATTR_TESTDATA?

PHYLINK

- Abstraction interface between:
  - Clause 22/45 PHY devices (MDIO)
  - Fixed (emulated) links (MAC to MAC)
  - SFP/SFF w/ optional EEPROM/diagnostics
- Rationale:
  - Systems having all of the above like SOHO switches
  - Maximize code share-ability and correctness
  - Better control over auto-negotiation process
  - Simply supporting the hardware!

PHYLINK

- Uses:
  - Device Tree for HW description:
    - phy-handle
    - fixed-link
    - sfp/sff
  - Linux i2c subsystem for EEPROM/diagnostics
  - PHY library for Clause 22/45 devices
- Provides:
  - SFP bus abstraction layer

PHYLINK

- SFP bus:

```
struct sfp_upstream_ops {
        int (*module_insert)(void *priv, const struct sfp_eeprom_id
*id);
        void (*module_remove)(void *priv);
        void (*link_down)(void *priv);
        void (*link_up)(void *priv);
        int (*connect_phy)(void *priv, struct phy_device *);
        void (*disconnect_phy)(void *priv);
};
```

PHYLINK

- On-going araes of work:
  - SFP/SFF quirks (invalid EEPROM etc.)
  - Lack of signals (LOS, TX-DISABLE etc.)
  - Wider adoption: Marvell PP2, NETA etc.
- Would be nice if NIC vendors used it:
  - If nothing else, the SFP bus infrastructure
  - Would help build a more robust implementation

DSA

- Feature complete! Obviously not :)
- On-going work:
  - Multicast without bridge (each port as separate network device not grouped)
  - https://github.com/netdsa/dsatest
  - Bug fixing
  - Getting more devices supported: Microchip KSZ

DSA

- CPU/DSA network devices:
  - Two headed links between two devices
  - No representors
  - ethtool overlays onto master DSA device to access CPU statistics, etc.
  - Devlink to expose?
    - Statistics
    - Link state
    - Register dumps

Lightweight devices

- Different yet converging problems:
  - Memory footprint:
    - David Ahern’s LWT: https://github.com/dsahern/linux net/lwt-dev
  - Control only interfaces:
    - Florian’s L2-only: https://github.com/ffainelli/linux/commits/l2-only
    - Si-Wei’s IFF_HIDDEN_DEVICE:

      https://patchwork.ozlabs.org/patch/893971/

Lightweight devices

- Possible path forward:
  - Continue with David’s patches
  - Add tuning knobs to select different features to be disabled by caller:
    - sysfs
    - IPv4/v6 stack
    - Notifications?
  - Might involved a preliminary phase to clean-up net_device (yeah..right..)

Too many interfaces

- ethtool::rxnfc
- tc: cls_u32, cls_flower etc.
- nftables/iptables

Too many interfaces

- Issues:
  - It’s all the same shared HW being programmed
  - Different user-space programming interfaces and in-kernel representations
  - Slightly different feature sets:
    - cls_flower is more feature complete to specify matches/actions
    - ethtool::rxnfc supports location placement (directed or automatic) and returns it to user-space
    - nftables/iptables is last to come, no upstream driver supports flow offload yet, you lose!
  - Lack of reviewer enforced that one API over the other must be implemented for new drivers

Too many interfaces

- Let’s move with tc anyway:
  - It’s complicated so it all gives us job security :)
  - Must add equivalent of location placement to make it strictly equivalent to ethtool
- In kernel representation:
  - Largely conflated with flow_dissector which upsets people
  - How about:
    - Netlink attributes → ethtool data structures → driver?