Update on Network Team Driver project

What's that, what's done, what's planned

Red Hat

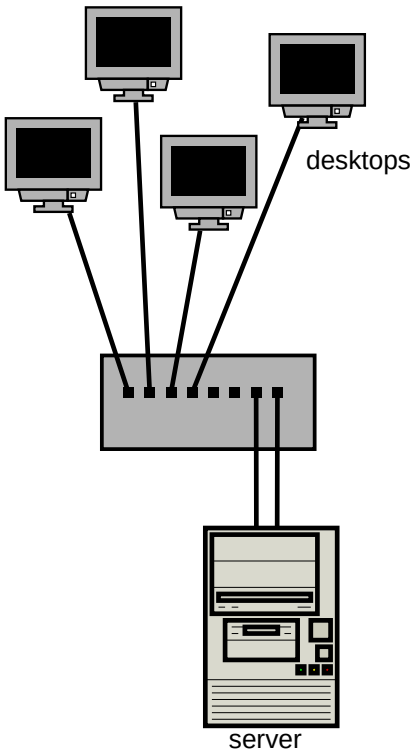
Jiří Pírko (jpirko@redhat.com)

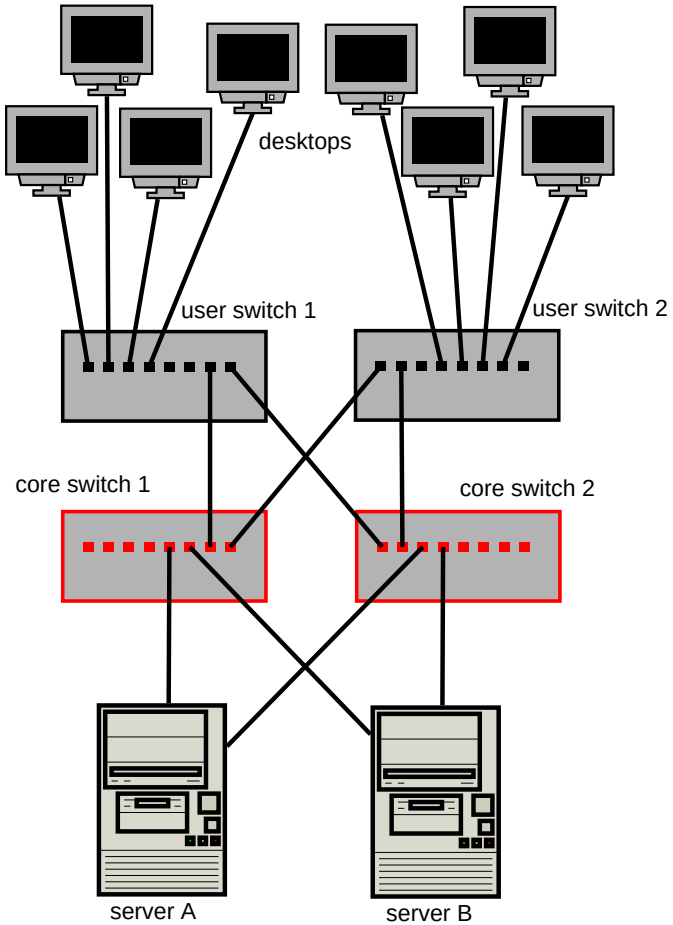
Section 1

Lightning talk part

redhat.

link aggregation setup examples





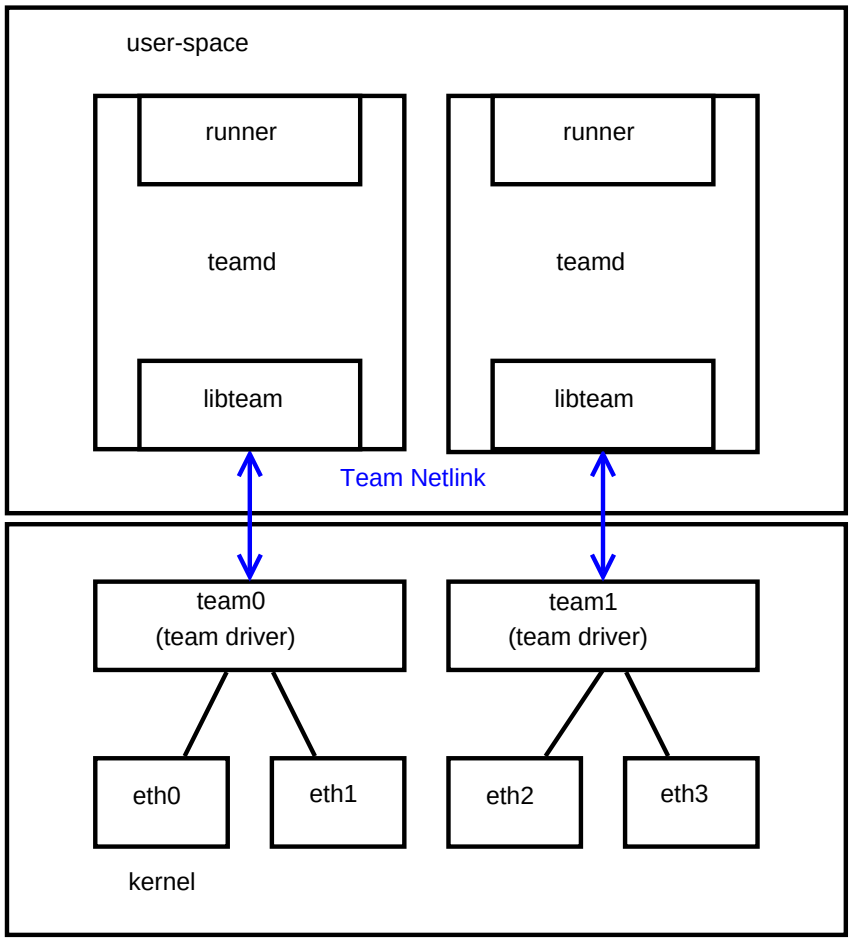
redhat.

Team driver project overview

- http://www.libteam.org
- Provides an alternative to existing bonding driver
- Implements various kinds of link aggregation
- One interface to user-space - Team Netlink
- Minimum of the code runs in kernel-space
  - Introduces NO performance penalty (fast TX/RX paths is in kernel)
- Control logic is implemented in user-space daemon (teamd)
- Status
  - Package libteam-1.0 in stable repository of F17/F18
  - Working on package for Debian
  - kdump support (Rawhide, F19)

redhat.

Team device architecture



redhat.

Features comparison (selected ones)

feature	bonding	team
broadcast,round-robin TX policy	Yes	Yes
active-backup TX policy	Yes	Yes
LACP (802.3ad) support	Yes	Yes
Hash-based TX policy	Yes	Yes
User can set hash function	No	Yes
TX load-balancing support (TLB)	Yes	Yes
TX load-balancing support for LACP	No	Yes
Ethtool link monitoring	Yes	Yes
ARP link monitoring	Yes	Yes
NS/NA (IPV6) link monitoring	No	Yes
RCU locking on TX/RX paths	No	Yes
port prio and stickiness	No	Yes
separate per-port link monitoring setup	No	Yes
multiple link monitoring setup	Limited	Yes
VLAN support	Yes	Yes
multiple device stacking	Yes	Yes

redhat.

What's planned

- Network Manager basic support (F19 or F20)
- Network Manager full GUI support (F20 or F21)
- RX balancing support (ala bonding ALB)
  - bridge/openvswitch support
  - Connection tracking
- Autoconfiguration support using LLDP (link layer discovery protocol)
- Extended custom TX hash function setup
  - Pre-defined protocols (eth, vlan, ipv4, ipv6, udp, tcp, sctp, ...)
  - User-defined custom protocols
- Extended teamd API support
  - Full featured teamd control API - wrapped by `libteamd`
  - Support teamd runners over teamd API (standalone processes)

Section 2

Addendum

redhat.

Team Netlink

- Uses Generic Netlink
- Setting options and getting events
- Port list message
  - Read-only (kernel-space to user-space only)
  - Contains interface indexes, link info (ethtool)
- Option list message
  - Both directions
  - User-space sets options and that changes behaviour
  - Various data types
- Accesible directly by wrap-up `teamnl` utility (debug)

redhat.

team driver

- Present in upstream Linux kernel since 3.3
- One instance - one network interface (e.g. `team0`)
- Code partition:
  - Netdev API code
  - Necessary fast-path code (transmit and receive packets)
  - Netlink communication (Team Netlink)
  - Modes code
- Team "modes":
  - One mode - one kernel module
  - Each mode determines specific low-level behaviour
  - Well defined API between team core and mode code
  - round-robin, broadcast, active-backup, loadbalance
  - Easy to add more

redhat.

libteam

- Does user-space wrapping of Team Netlink communication
- Wraps-up Route Netlink messages (newlink, dellink etc.)
- Uses libnl (genl, rtnl)
- Exports API to user for controlling kernel team driver instance (setting options)
- Allows to register "handlers" for watching team driver instance events (port and option changes)
- Python binding available

redhat.

teamd

- Uses libteam, libdbus, jansson, libdaemon
- One instance controls one team driver instance
- On startup it creates team driver instance
- Config file in JSON format
- Higher logic is implemented in Runners
- Link monitoring is implemented in Link-watches
- Run-time control via D-Bus API
- Easy to use:

```
# teamd -f team0.conf
```

redhat.

teamd Runners + Link-watches

- Runners
  - Only one has to be selected
  - Determine behaviour
    - Set-up team driver instance (mode, options)
    - Monitor team driver instance behaviour and react to that
  - Kinds:
    - Round-robin / Broadcast
    - Active-backup
    - LACP (802.3ad)
    - Load-Balancing
- Link-watches
  - Link monitoring
  - Kinds:
    - ethtool
    - ARP ping
    - IPv6 NS/NA ping (Neighbor Discovery Protocol)
  - Can be set as global or per-port
  - Possible to set multiple link-watches

redhat.

teamd D-Bus API

- Service name: `org.libteam.teamd.[teamdevname]`
- Methods:
  - ConfigDump - Get teamd JSON config
  - StateDump - Get teamd state
  - PortAdd, PortRemove - Adds/removes port
  - PortConfigUpdate - Updates config for a given port
- Wrapped up by `teamdctl` utility
- Might be eventually extended for possibility of runner implementation (idea)

redhat.

teamd config example 1

team0.conf

```
{
  "device": "team0",
  "runner": {"name": "roundrobin"},
  "ports": {"eth1": {}, "eth2": {}}
}
```

redhat.

teamd config example 2

team0_ab_ethtool.conf

```
{
  "device": "team0",
  "runner": {"name": "activebackup"},
  "link_watch": {"name": "ethtool"},
  "ports": {
    "eth1": {
      "prio": -10,
      "sticky": true
    },
    "eth2": {
      "prio": 100
    }
  }
}
```

redhat.

Advantages comparing to bonding

- Extensibility. Anyone can easily add features/change behaviour
- Better system stability (daemon crash is always better than kernel panic/memory corruption etc.)
- Better debugging posibilities
- Minimum kernel changes required

redhat.

team driver implementation

- RCU-locking (multiple incoming/outgoing packets can go in parallel with setting up the device)
- Exploits `rx_handler` on RX path to intercept packets
- Uses `dev_queue_xmit()` to pass packets to NIC driver on TX path
- `netdevice_notifier` events are passed via Netlink
- Option infrastructure - for easy add options to modes

redhat.

Extension possibilities

- Designed to be easily extendable by any advanced user
- Kernel-space extension
  - Add mode (should be rarely needed)
- User-space extension
  - Make libteam based application
  - Make libteam python binding based application
  - Add teamd runner (preferred)

The end.

Thanks for listening.