Teaming network device

Put multiple physical ethernet devices into one logical one

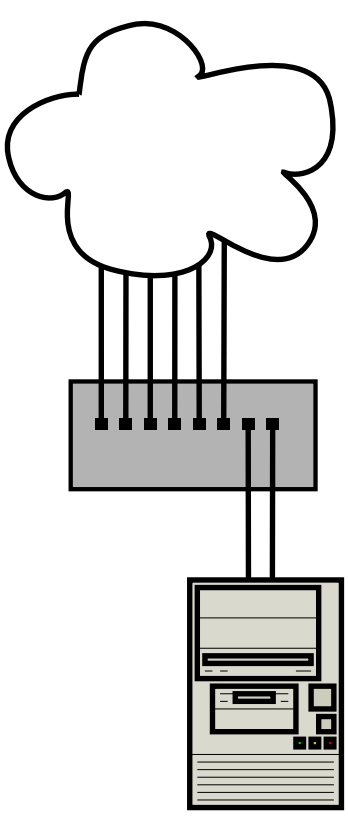
Red Hat

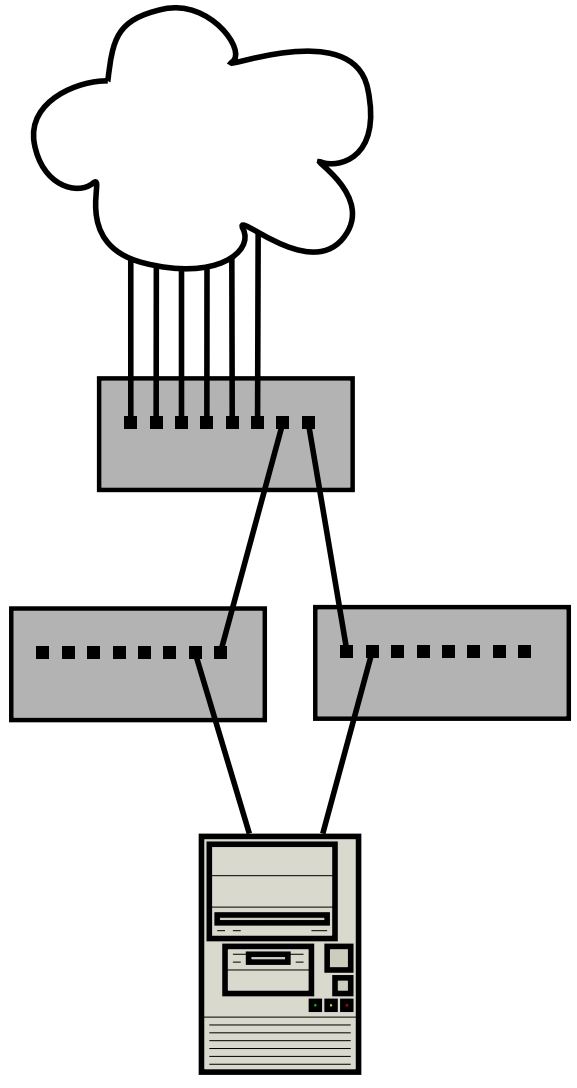
Jiří Pírko (jpirko@redhat.com)

Generic ideas about (ethernet) link aggregation

- Link layer (2nd OSI)
- Combining multiple network connections in parallel
- Main motivations
  - increase throughput beyond what a single line can provide
  - provide redundancy (failover)
- 802.3ad, LACP + many proprietary standards
- Current Linux implementation is bonding driver

link aggregation setup examples





Bonding driver

- Introduced in 2000
- Huge and messy, therefore buggy
- All logic is in kernel (monolith)
- Does what it should not do (ARP link validation, 802.3ad, ...)
- Too many config interfaces
- 12200 lines
- Not fixable due to backward compatibility concerns

redhat.

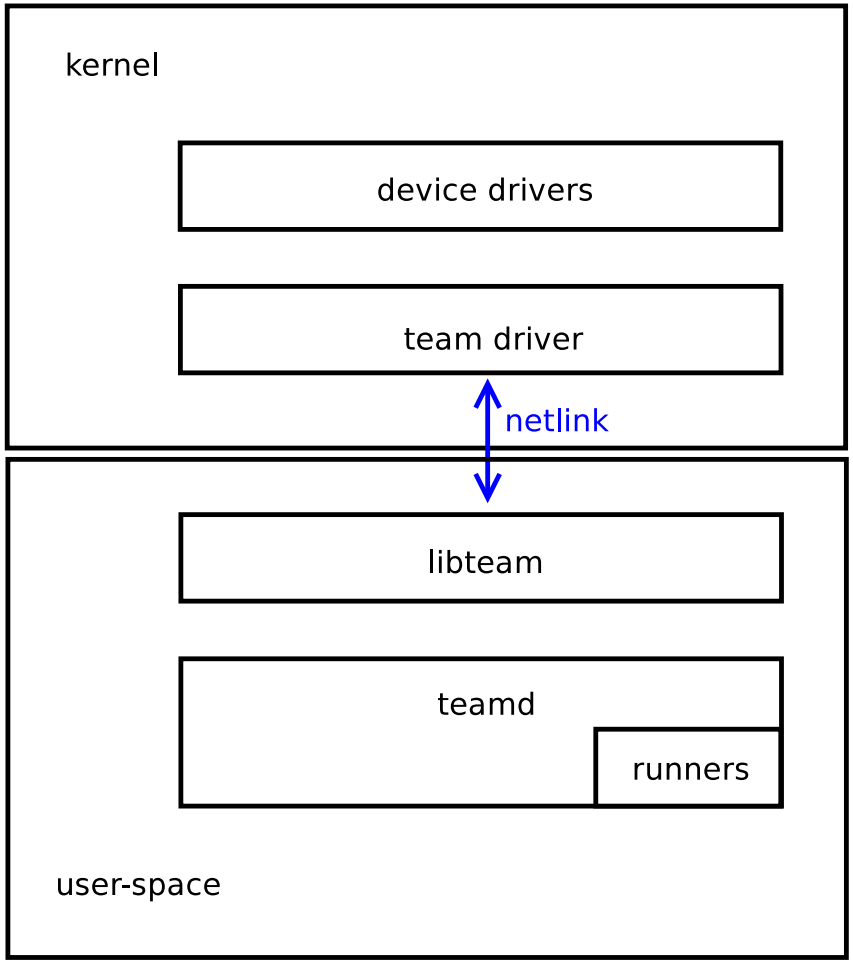
What the "Team device" is about?

- Think of it as of bonding-new_generation
- Basically it's tool to implement various kinds of link aggregation
- Alternative names would might be "trunking" or "link bundling" or "Ethernet/network/NIC bonding"
- "teaming" was chosen because it's nicest
- The goal of team device is to supersede bonding functionality and then kill it eventually

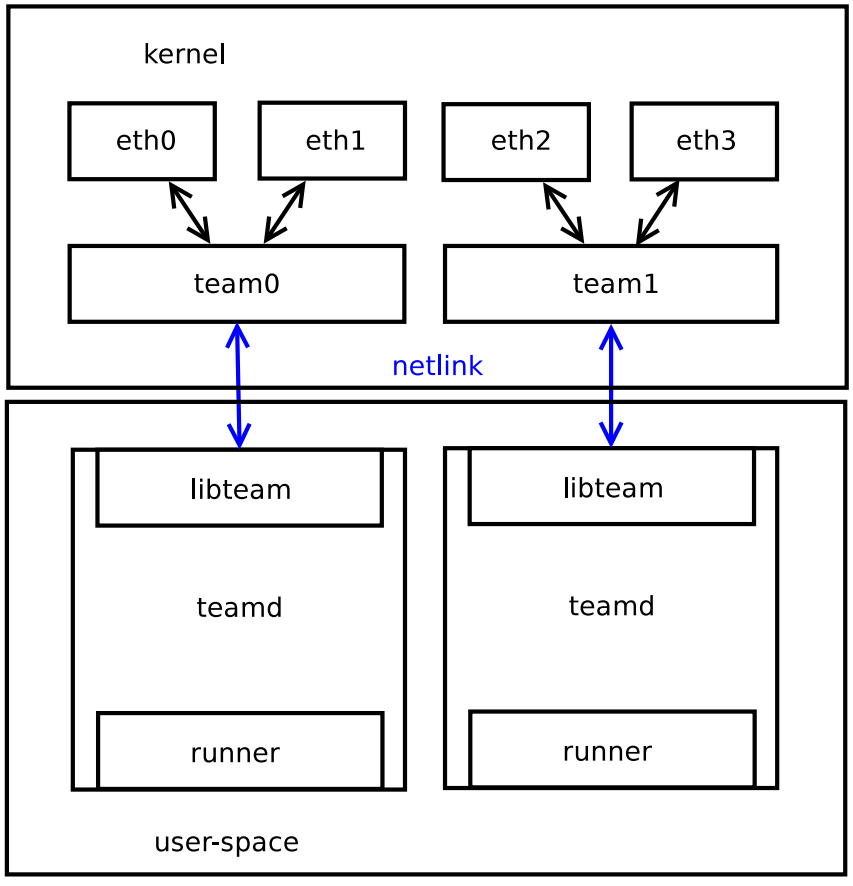
Team device overview

- Team is coming with modular approach
- User-space based controlling
- Minimum of the code is in kernel
  - "Puppet"
- Control logic is implemented in user-space daemon
  - "Puppeteer"
- Enslaved network interfaces are called "ports"

Team device architecture (class-like-view)



Team device architecture (instance-like-view)



team driver (kernel)

- Only necessary fast-path code. (1400 lines)
- Netlink communication (generic Netlink). (600 lines)
- Team "modes"
  - One mode, one kernel module
  - Determine basic low-level behaviour
  - Well defined API between team core and mode code
  - round-robin, active-backup, ... easy to add more

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

libteam

- Team generic Netlink wrap-up
- Uses libnl (genl, rtnl)
- Exports API to user for controlling kernel team driver instance
- Allows to register "handlers" for watching team driver instance events
- Python binding available + more bindings are going to be present

teamd

- Uses libteam, libdbus, jansson, libdaemon
- One instance puppet-controls one team driver instance
- On startup it creates team driver instance (team interaface)
- Config file in JSON format
- Runners
  - One and only one has to be selected
  - Determine behaviour ("pulling strings, getting punches")
  - Either part of teamd (C) or separate application (D-BUS)
    - Active-backup (link monitoring, ARP/NA monitoring)
    - 802.3ad
    - Multiple switch load-balancing
    - ...whatever whoever wants/implements...
- Easy to use:

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

teamd config example 1

team0.conf

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

teamd config example 2

team0_abl.conf

```
{
    "device":   "team0",
    "runner":   "activebackup_linkmon",
    "ports":    {
                    "eth1": {
                        "prio": -10,
                        "sticky": true
                    },
                    "eth2": {
                        "prio": 100
                    }
                }
}
```

Extension possibilities

- kernel extension
  - Add mode
- user-space extension
  - Make libteam based application
  - Make libteam python binding based application
  - Add teamd runner
    - Preferred
    - Might be written in language of your choice (D-BUS API)

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.

Status/get involved

- Kernel bits are present in 3.3
- libteam (including teamd) packaged in Fedora 16 (updates-testing) and Rawhide
- Infrastructure is 90% done, more functionality implementation pending (802.3ad, load-balancing, ARP/NA monitoring)
- Developers/testers wanted
- http://www.libteam.org
- #teamdev at freenode
- libteam@lists.fedorahosted.org

The end.

Thanks for listening.