Software Defined Networking

Structure

1. SDN *Introduction*

Structure

1. SDN *Introduction*
2. *Container* Basics

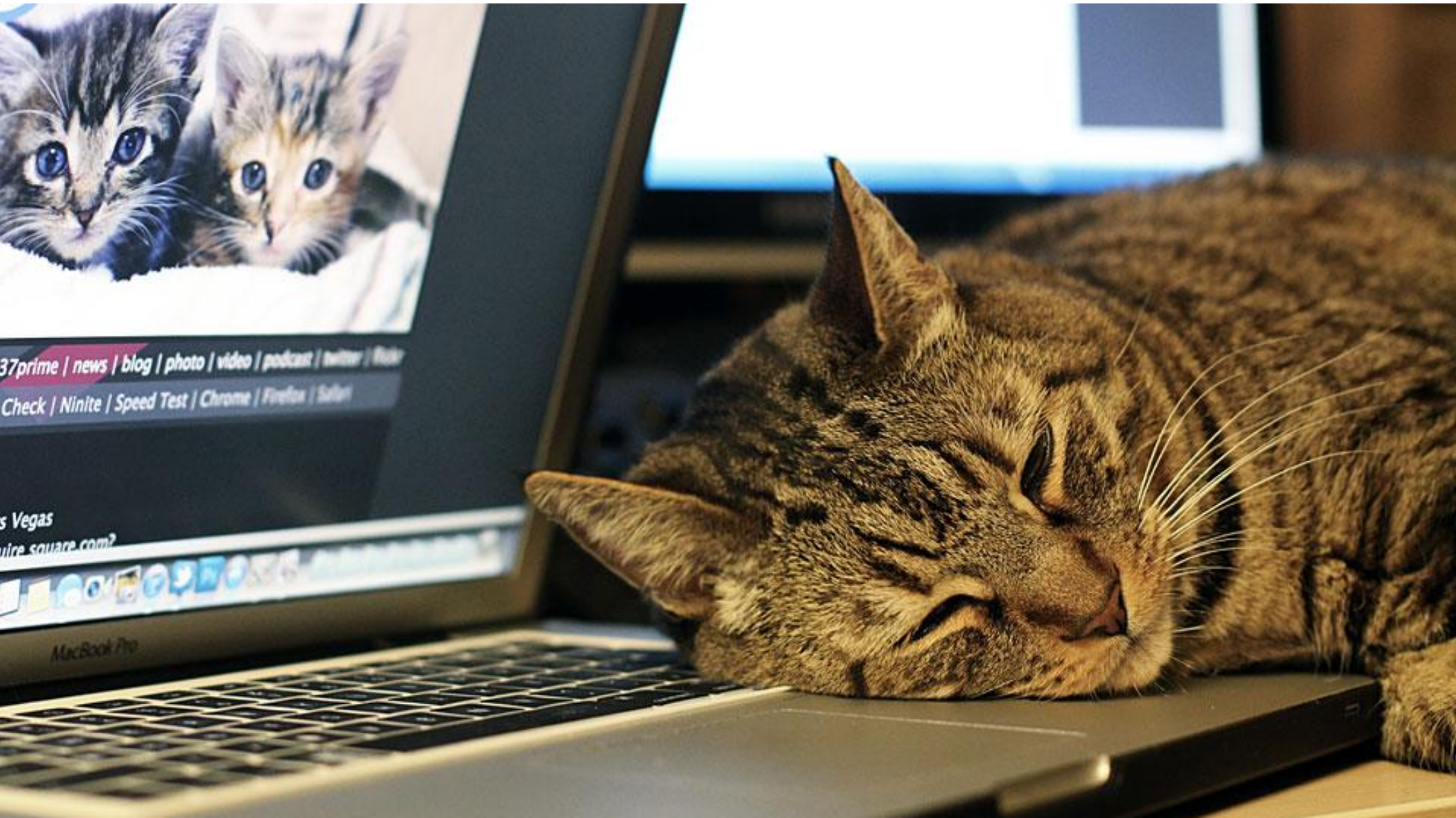
Structure

1. SDN *Introduction*
2. *Container* Basics
3. *Container* Networking

Structure

1. SDN *Introduction*
2. *Container* Basics
3. *Container* Networking
4. *Demo*

1. Introduction to SDN



Motivation

SDN is an abstract concept:
can mean many things

Why SDN?

- *program the network* through an API
- reduce capex, opex
- enable *multi-tenancy*; massive networks
- avoid some *limitations* (eg. 4094 VLANS)

SDN - Motivation

1. Massive networks:
   - Google*
2. Multi-Tenant Virtualized Systems:
   - Amazon Virtual Private Cloud (VPC)
   - OpenStack Neutron
   - vmWare / vCloud *("Software Defined Data Center")*
3. Rapid Deployment of VMs and Containers
4. VM Mobility

* http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p183.pdf

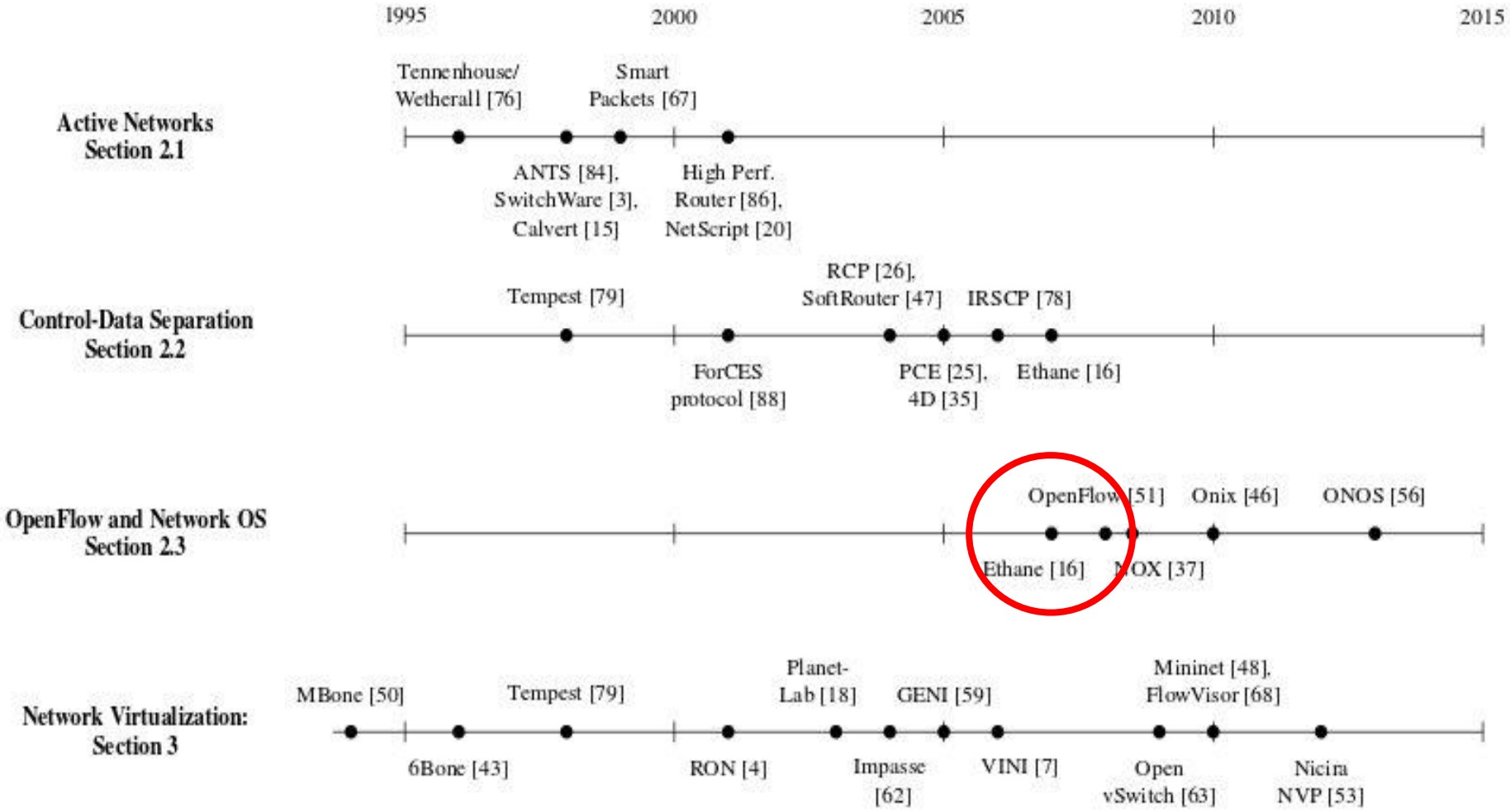
SDN History

Key concepts:

- separate *Control Plane* and *Data Plane* *(FE)*
- *consolidate Data Plane*: single control program
  - e.g. OpenFlow: switch / router / NAT / firewall

- SDN “standardization" began with OpenFlow
- many emerging trends (see www.sdxcentral.com)

SDN History



- https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf

Overlay Networking *Principles*

Virtual Network overlays the *physical one* via tunnel

- *Virtual Extensible LAN* (VXLAN, RFC 7348)
  - extension of 802.1q VLAN (*max **4094** LANs*)
  - *max **16 million** LANs per domain* (24 bit VNI)

Tunneling Concept

- carry VM MAC traffic from one end to another
- *Russian Doll principle* (VXLAN: L2 in UDP)
- *Generic Routing Encapsulation* (GRE)
  - now used by NVGRE to tunnel L2 over L3
- *Stateless Transport Tunneling* (larger VNID)

Why do Containers need SDN?

How do containers talk to each other?

- ***easy:*** just let them talk on ***same*** host
- ***hard:*** how to talk across ***different*** hosts?
- enable multi-tenancy/network segregation
- allow for access to network info for proxy, etc

2. Container Basics

Why containers?

- **modularization** of server components:
  - typically *smaller* than VM images
    (e.g. docker Go container in ~12MB)
  - servers typically run *lots of processes*
  - *example:* OpenStack infrastructure
- **segregation** of server setup
  - *live upgrade* ("hot swap") of components
  - e.g. take component off load balancer, swap out, replace
- simplified **dependency management**
  - *force* config files, image managment, …
  - self-contained app containers

Containe*rology*

Types:

- LXC, CoreOS runC
- systemd nspawn
- Docker (bocker)

Technological Basis:

- Linux netlink, namespaces, cgroups

Orchestration:

- kubernetes
- Mesosphere
- Docker Swarm
- ...

Pros & Cons

- LXC "fat container" vs Docker "*thin container*"

Docker advantages:

- DSL for application deployment
- small/simple solution (1 process/container)
- *introspection*
- *modular* and portable deployment
- *uniformity:* *development image* (laptop) and *production image* are same

Docker disavantages:

- lock-in
- dockerization

Pros & Containers

LXC advantages:

- flexibility, "*OS in a box*"
- *multiple ways* of doing things
  - opposite of Docker, which *eliminates options* to keep things simple
- *powerful*:
  - bocker = docker in ~100 lines of bash
    https://github.com/p8952/bocker

LXC disadvantages:

- administration via big, unwieldy (bash) scripts
- low level of sophistication (costly to maintain)

3. SDN for Containers

Networking Docker Containers

- networking *multiple* containers/hosts?
- *flat layer 3s*
  - LXD fan
  - Project Calico
- *Underlay/Overlay*
  - VXLAN, GRE

Docker netlink - Evolution

I. pipework bash script using iproute2

Docker netlink - Evolution

I. pipework bash script using iproute2

II. rewrite of pipework into Go (tenus)

Docker netlink - Evolution

I. pipework bash script using iproute2

II. rewrite of pipework into Go (tenus)

III. tenus becomes docker/libcontainer

Docker netlink - Evolution

I. pipework bash script using iproute2

II. rewrite of pipework into Go (tenus)

III. tenus becomes docker/libcontainer

IV. vishvananda fork of libcontainer

most comprehensive netlink support so far

`https://github.com/vishvananda/netlink`

Docker netlink - Evolution

I. pipework bash script using iproute2

II. rewrite of pipework into Go (tenus)

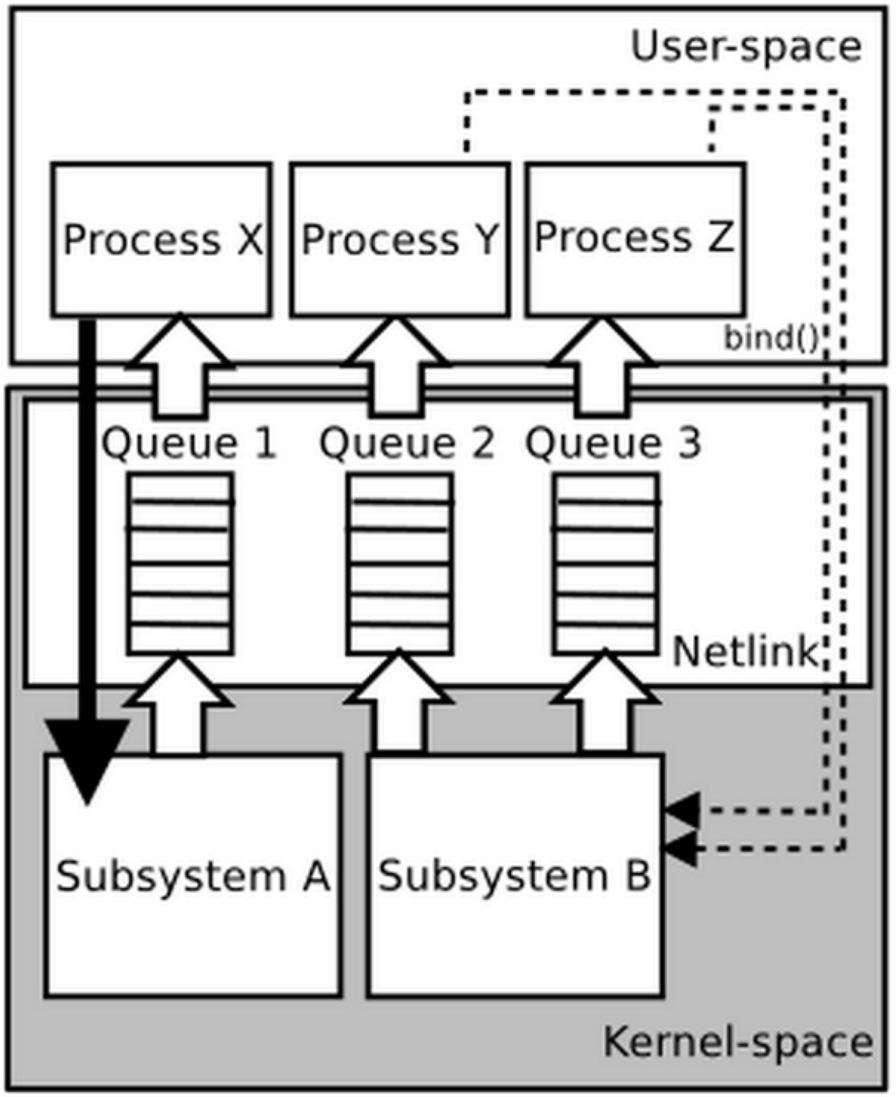
III. tenus becomes docker/libcontainer

IV. vishvananda fork of libcontainer
most comprehensive netlink support so far
https://github.com/vishvananda/netlink

V. evolving docker libnetwork
based on vishvananda netlink / netns
https://github.com/docker/libnetwork

What is netlink

- **kernel/userland IPC**
  - socket-based
  - extensible (TLV)
- **get/set parameters**
- **issue commands**
- **await events**
  - notifications
  - multicast
  - publish/subscribe



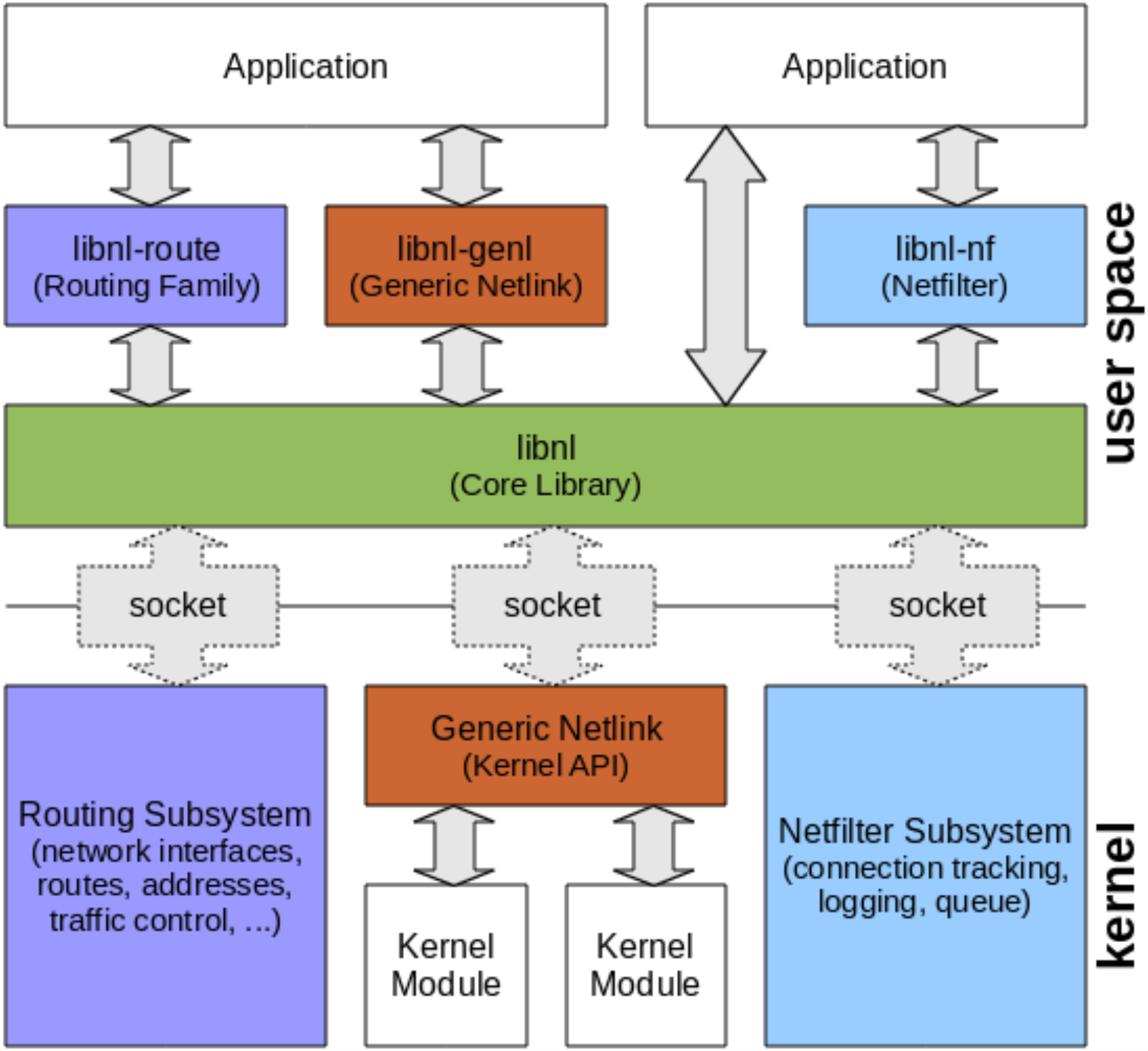
netlink main uses

SDN: separate control/dataplane (RFC 3549)

- inspired by BSD 4.4 routing sockets NETLINK_ROUTE

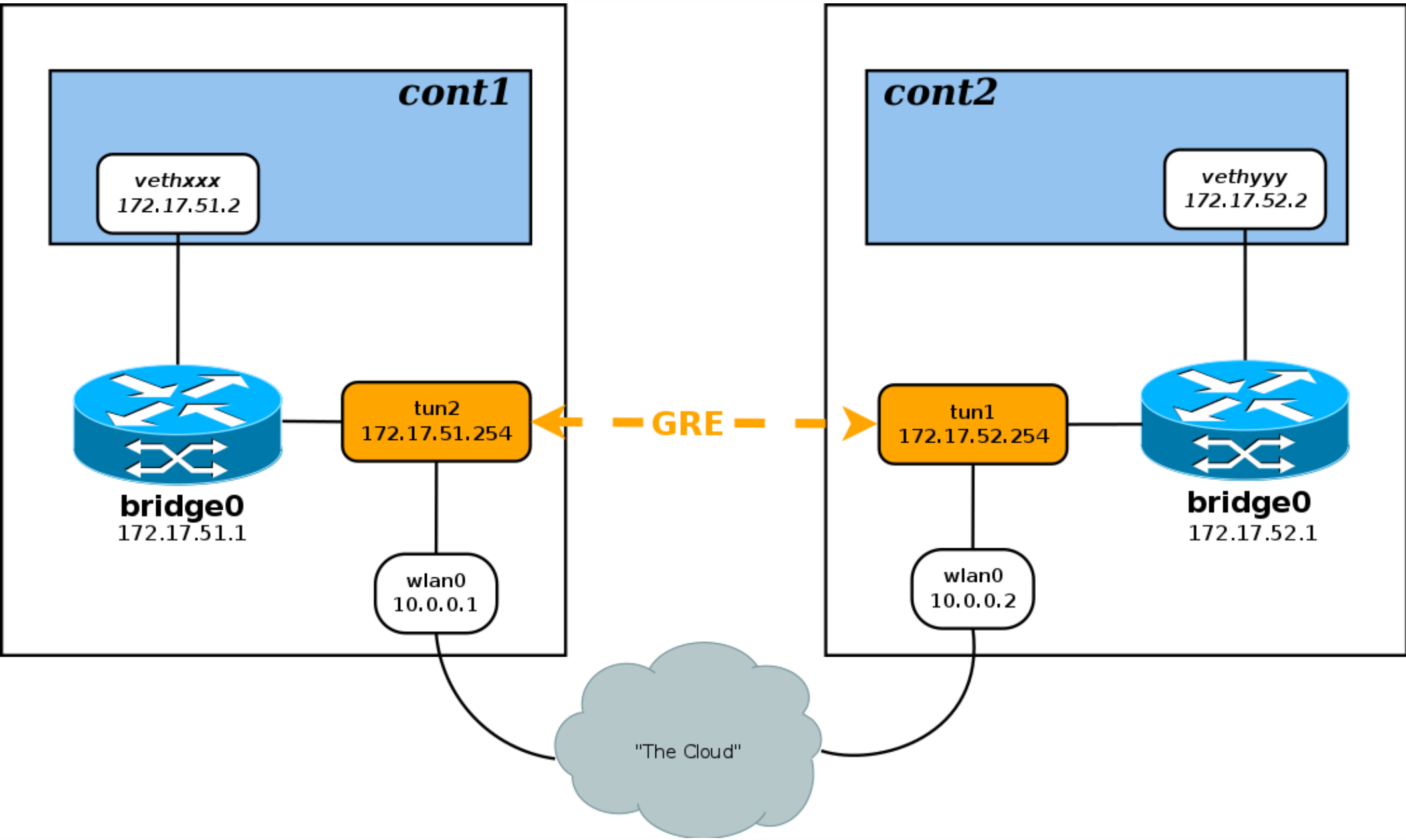
Networking *control plane* (iproute2):

- interfaces
- VXLANs
- routing / neighbor tables
- namespaces
- firewalling (netfilter)
- traffic shaping + policing (RTM_xx_QDISC)
- IPSec (NETLINK_XFRM)


Application
Application
libnl-route
(Routing Family)
libnl-genl
(Generic Netlink)
libnl-nf
(Netfilter)
user space
libnl
(Core Library)
socket
socket
socket
Routing Subsystem
(network interfaces,
routes, addresses,
traffic control, ...)
Generic Netlink
(Kernel API)
Kernel
Module
Kernel
Module
Netfilter Subsystem
(connection tracking,
logging, queue)
kernel

4. Demo

Demo Container Network



Featured vishvanada funcs

```
// Link manipulation
func LinkAdd(link Link) error
func LinkDel(link Link) error
func LinkList() ([]Link, error)
func LinkSetUp(link Link) error
func LinkSetDown(link Link) error

// Address manipulation
func AddrAdd(link Link, addr *Addr) error
func AddrDel(link Link, addr *Addr) error
func AddrList(link Link, family int) ([]Addr, error)

// Route manipulation
func RouteAdd(route *Route) error
func RouteDel(route *Route) error
func RouteList(link Link, family int) ([]Route, error)
func RouteGet(destination net.IP) ([]Route, error)
```

Why Linux Netlink?

```
struct ip_tunnel_parm {
        char                    name[IFNAMSIZ];
        int                     link;
        __be16                  i_flags;
        __be16                  o_flags;
        __be32                  i_key;
        __be32                  o_key;
        struct iphdr            iph;
};

int tnl_add_ioctl(const char *name, void *p) {
        struct ifreq ifr;
    // ...

        strncpy(ifr.ifr_name, name, IFNAMSIZ);
        ifr.ifr_ifru.ifru_data = p;

        err = ioctl(fd, SIOCADDTUNNEL, &ifr);
    // ...
}
```

Why Linux Netlink?

```
struct ip_tunnel_parm {
        char                    name[IFNAMSIZ];
        int                     link;
        __be16                  i_flags;
        __be16                  o_flags;
        __be32                  i_key;
        __be32                  o_key;
        struct iphdr            iph;          // NESTED STRUCTURE
};

int tnl_add_ioctl(const char *name, void *p) {
        struct ifreq ifr;
    // ...

        strncpy(ifr.ifr_name, name, IFNAMSIZ);
        ifr.ifr_ifru.ifru_data = p;

        err = ioctl(fd, SIOCADDTUNNEL, &ifr);
    // ...
}
```

Conclusion

- *SDN is quickly evolving*
  - still early stages
  - room for innovation and ideas!
- *SDN is Startup Defined Networking*
  - SocketPlane (acquired by docker)
  - Nicira (now vmWare NSX)
  - Insiemi (now Cisco ACI)
  - Cumulus Linux (debian from $629/year)
- Go libnetwork uses netlink/iproute2
  - so can you!

Conclusion

- *SDN is quickly evolving*
  - still early stages
  - room for innovation and ideas!
- *SDN is Startup Defined Networking*
  - SocketPlane (acquired by docker)
  - Nicira (now vmWare NSX)
  - Insiemi (now Cisco ACI)
  - Cumulus Linux (debian from $629/year)
- Go libnetwork uses netlink/iproute2
  - **so can you!**