Development of vxlan(4) using rump kernel

Kazuya GODA
(k-goda@iij.ad.jp)

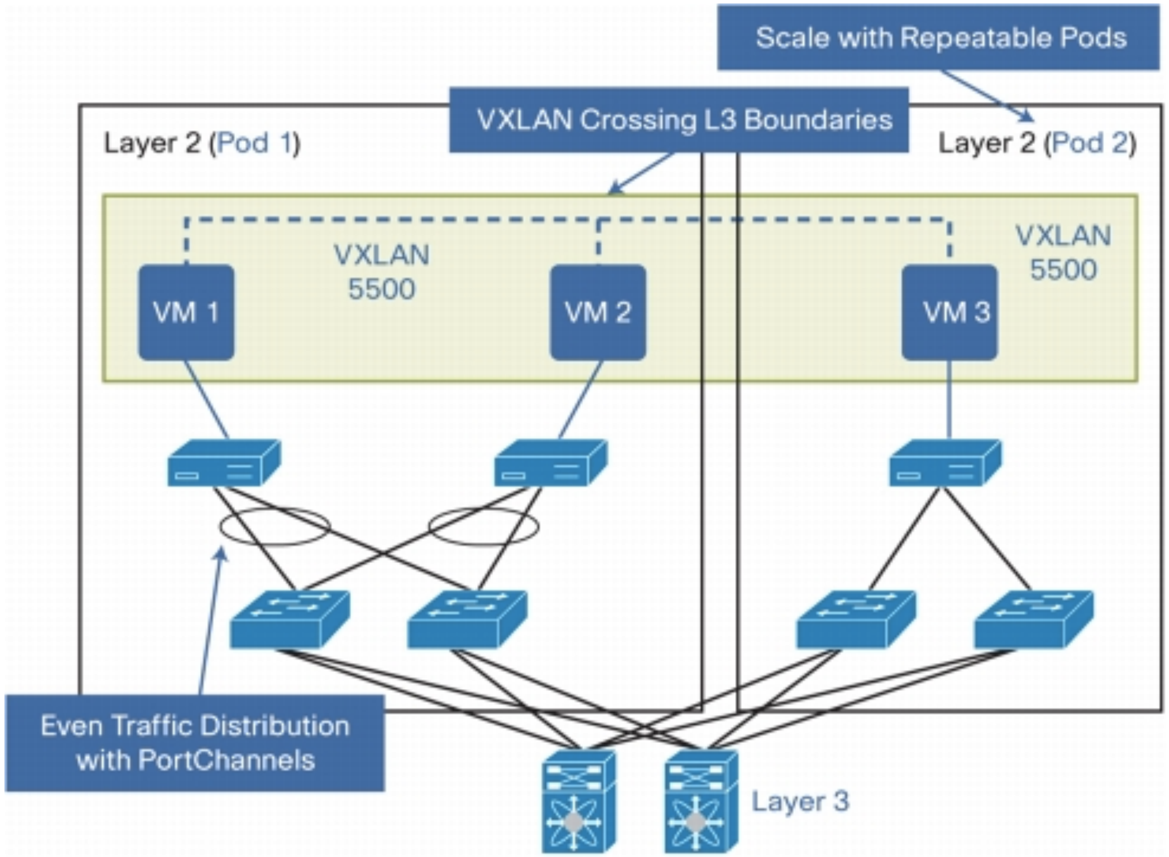
Introduction

- I’m porting vxlan(4) from FreeBSD
- This working uses rump kernel for developing

- I talk about how do I developed vxlan(4) using rump kernel

What’s vxlan ?

- vxlan
  - A Layer 2 overlay scheme over a Layer 3 network
  - It uses MAC-in-UDP encapsulation


Scale with Repeatable Pods
VXLAN Crossing L3 Boundaries
Layer 2 (Pod 1)
Layer 2 (Pod 2)
VXLAN 5500
VXLAN 5500
VM 1
VM 2
VM 3
Even Traffic Distribution with PortChannels
Layer 3

http://www.cisco.com/web/JP/product/hs/switches/nexus1000/prodlit/images/data_sheet_c78-492971-11.jpg

Development of vxlan(4) using rumpkernel

1. Setup development environment
   – Build and run rump kernel
   – Access host's network
2. Develop
   – Kernel
   – Userland
3. Debug

Development of vxlan(4) using rumpkernel

1. Setup development environment
   - Build and run rump kernel
   - Access host's network
2. Develop
   - Kernel
   - Userland
3. Debug

Building rump kernel

- buildrump.sh
  - Utilities for building NetBSD kernel drivers as rump kernels for a variety of systems
  - https://github.com/rumpkernel/buildrump.sh
- buildrump.sh options
  - for using local source tree

options	argument
-s	source tree directory

  - for debugging

options	append flags
-D	-O2 -g
-DD	-DDEBUG
-DDD	-DLOCKDEBUG

Building & Running rump kernel

```
$ ./buildrump.sh -DDD -s ../netbsd-src fullbuild
>> NATIVE build environment probed
>>
>> NOTICE: Not a buildrump.sh-based repo in /home/k-goda/rumpkerneldev/netbsd-src
            :
$ ./buildrump.sh/rump/bin/rump_server -lrumpnet -lrumpnet_net -lrumpnet_netinet -
lrumpnet_virtif  unix://ctrl

$ rump.ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33648
     inet 127.0.0.1 netmask 0xff000000

$ rump.route show
Routing tables
Internet:
Destination      Gateway          Flags    Refs     Use    Mtu Interface
localhost        localhost        UH        -        -   33648  lo0
```

Access the host’s networking

- We would like to communicate with each other hosts because of testing

- A rump kernel networking is separated from host’s one

- Using virt(4) in rump kernel and bridging host’s tap and physical interface
  - virt interface is mapped host’s tap interface

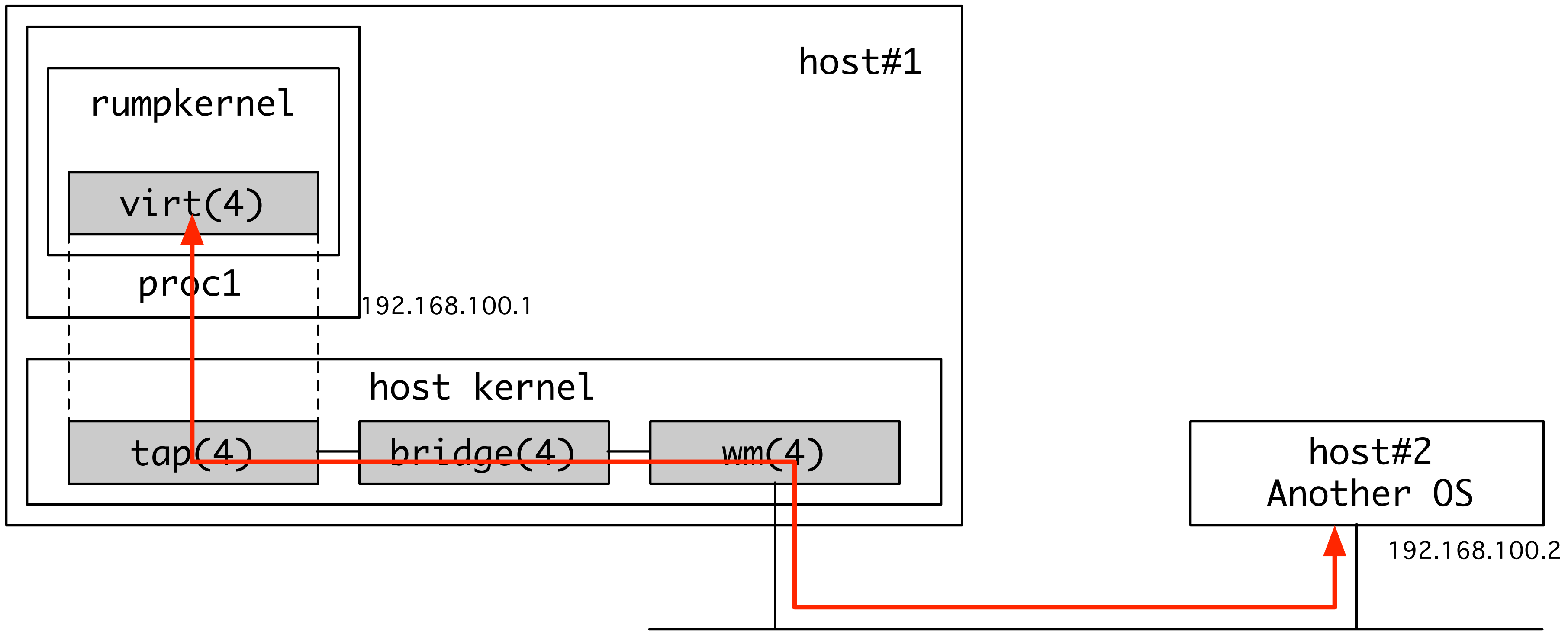
Access the host's networking

rump kernel side

```
# rump.ifconfig virt0 create
# rump.ifconfig virt0 192.168.100.1/24
```

host side

```
# ifconfig tap0 create up
# ifconfig bridge0 create up
# brconfig bridge0 add tap0 add wm0
```



Development of vxlan(4) using rumpkernel

1. Setup development environment
   – Build rump kernel
   – Access host's network
2. Develop
   – Kernel
   – Userland program
3. Debug

Create and Modify Files

- userland side

file	path	
create	sbin/ifconfig/vxlan.c	ifconfig command extension for vxlan(4)

- kernel side

file	path	
create	sys/net/if_vxlan.c	vxlan(4) component
create	sys/net/if_vxlan.h	vxlan(4) header file
modify	sys/netinet/udp_usrreq.c	add udpcb and encap udp tunneling proto path
modify	sys/netinet/udp_var.h	add udpcb

- rump kernel side

file	path	
create	sys/rump/net/Makefile.rumpnetcomp	
create	sys/rump/net/lib/libvxlan/vxlan_component.c	
create	sys/rump/net/lib/libvxlan/Makefile	

kernel

- kernel side
  - A implementation of vxlan(4) is typically developed without concern for rump kernel

- rump kernel side
  - We need to build driver components to librump libraries
    - vxlan driver is named librumpnet_vxlan
  - Make files
    - sys/rump/net/lib/libvxlan/vxlan_component.c
    - sys/rump/net/lib/libvxlan/Makefile

Inside of vxlan_componet.c

```
#include <sys/param.h>

#include "rump_private.h"
#include "rump_net_private.h"

int vxlanattach(int);

RUMP_COMPONENT(RUMP_COMPONENT_NET_IF)
{
   vxlanattach(0);
}
```

userland side

- A implementation of ifconfig vxlan extension is typically developed without concern for rump kernel
- Build for using normal tool chain, output ifconfig and rump.ifconfig
- using rump.ifconfig for rump kernel

Development of vxlan(4) using rumpkernel

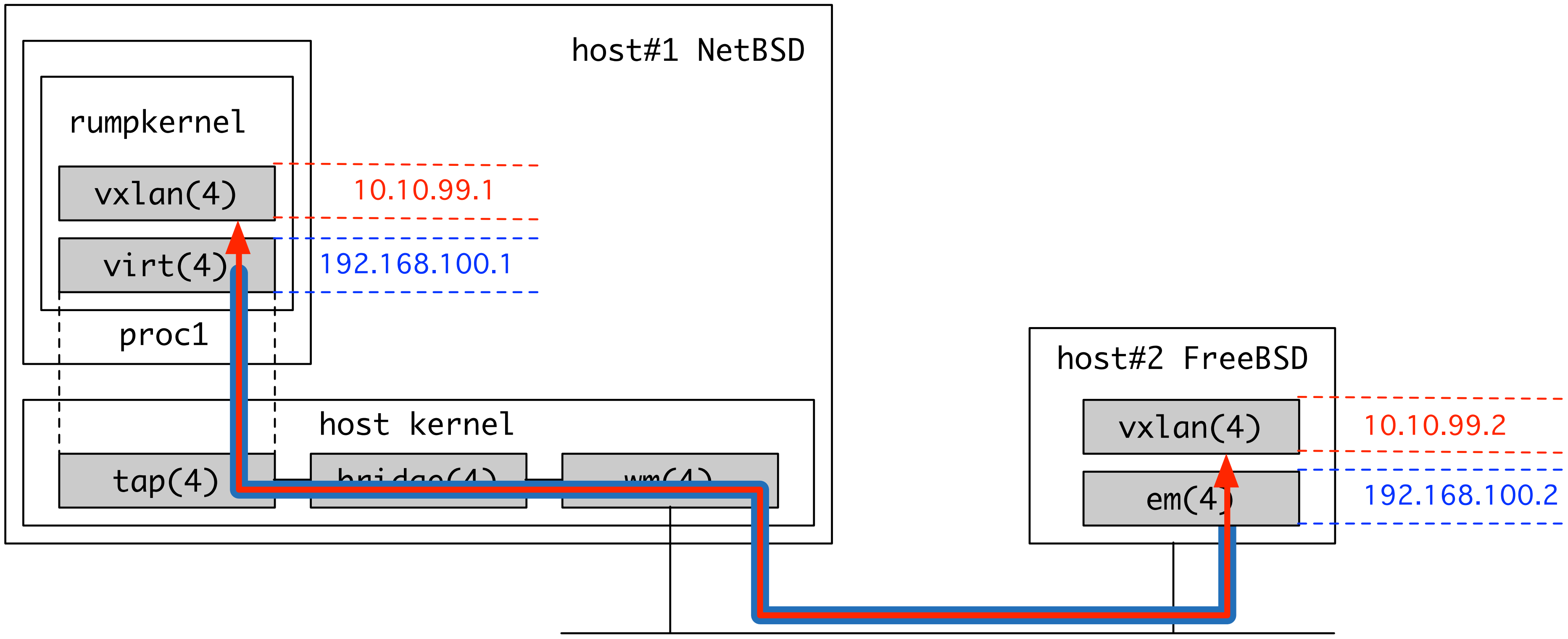
1. Setup development environment
   - Build rump kernel
   - Access host's network
2. Develop
   - Kernel
   - Userland program
3. Debug

Debug

- Using gdb
  - A rump kernel is userland process
- Called panic(9) in kernel, rump kernel call abort(3) and exit
  - We can get panic message but can’t get backtrace…
    - insert backtrace(3) linrumpuser but it not good solution

(if possible) DEMO

- a rump kernel on host#1 sends ping to host#2
  - the host#1 is NetBSD, the host#2 is FreeBSD



Thank you !!

I will publish the source code

in the near term