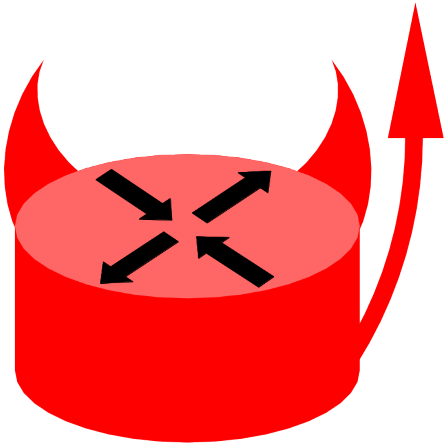


BSD Router Project

Don't buy a router: download it !

Olivier Cochard-Labbé
olivier@cochard.me

FOSDEM'15

Agenda

- Why a x86 software router ?
- Project Targets
- NanoBSD: FreeBSD for appliance
- BSDRP feature list
- Benchmarking forwarding performance
- Virtual lab
- Roadmap

Why a x86 software router ?

- My thoughts in 2009
  - x86 servers should be able to deliver more PPS

Why a x86 software router ?

- My thoughts in 2009
  - x86 servers should be able to deliver more PPS
- 2011
  - netmap and Intel DPDK were introduced

⇨ x86 is ready for high-performance network appliance

Why a x86 software router ?

- My thoughts in 2009
  - x86 servers should be able to deliver more PPS
- 2011
  - netmap and Intel DPDK were introduced

→ x86 is ready for high-performance network appliance

- 2012
  - Software Defined Network (SDN)
  - Network Functions Virtualization (NfV)

→ Virtualization solutions are mainly x86 based

Why a x86 software router ?

- My thoughts in 2009
  - Software Configuration Management (SCM) for large multi-vendors network didn’t exist… But NETCONF is coming
  - x86 world had lot’s of tools: Chef, Puppet, CFEngine

Why a x86 software router ?

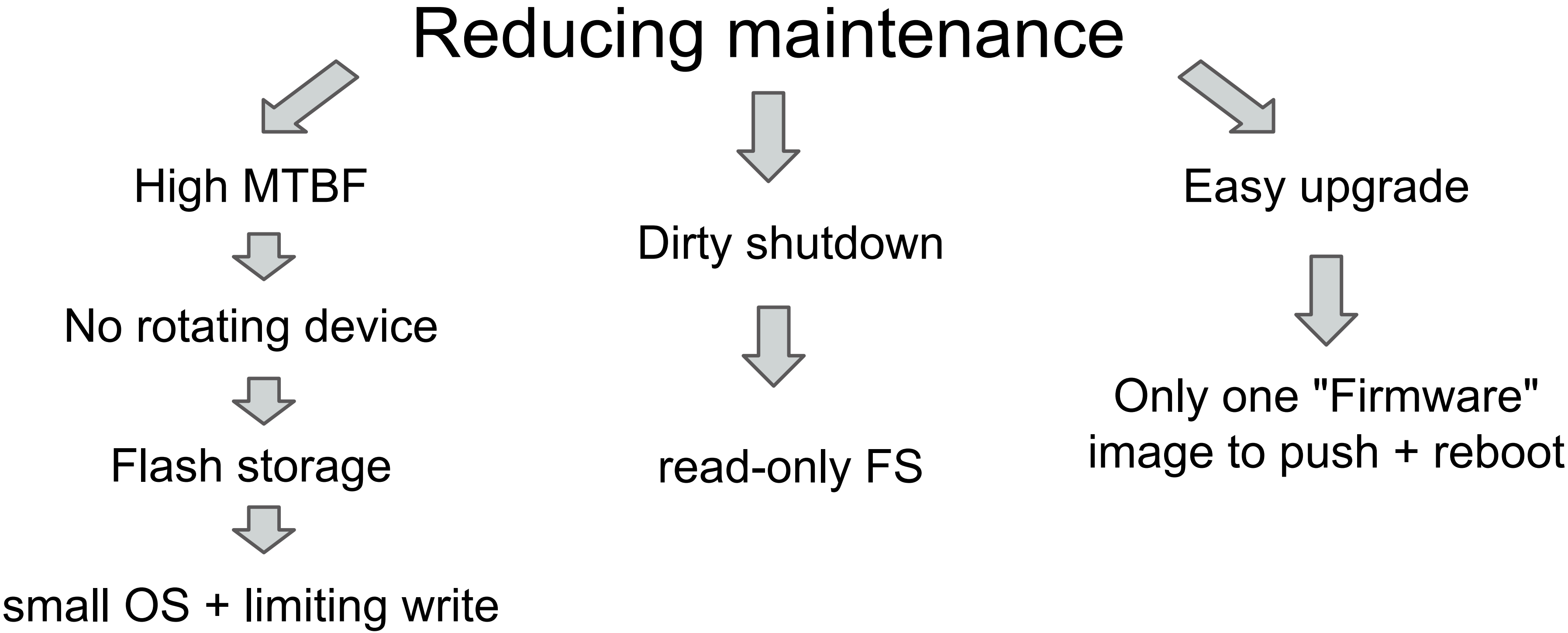
- My thoughts in 2009
  - Software Configuration Management (SCM) for large multi-vendors network didn’t exist… But NETCONF is coming
  - x86 world had lot’s of tools: Chef, Puppet, CFEngine
- 2015
  - NETCONF (23 RFC!!!) is still not production ready
  - More x86 tools: Ansible, Salt, etc…

⇨ x86 based appliance can use any existing SCM

Project targets

- Targets
  - Medium sized Giga/TenGiga Ethernet router
  - Not for home: Use m0n0wall of pfSense
- No WebGUI
  - Compliant with existing FreeBSD’s user base
  - Large deployment should managed by any common SCM
- Audience: Network administrators
  - Manageable as an appliance (one firmware)

NanoBSD: FreeBSD for appliance



Reducing maintenance

High MTBF

No rotating device

Flash storage

small OS + limiting write

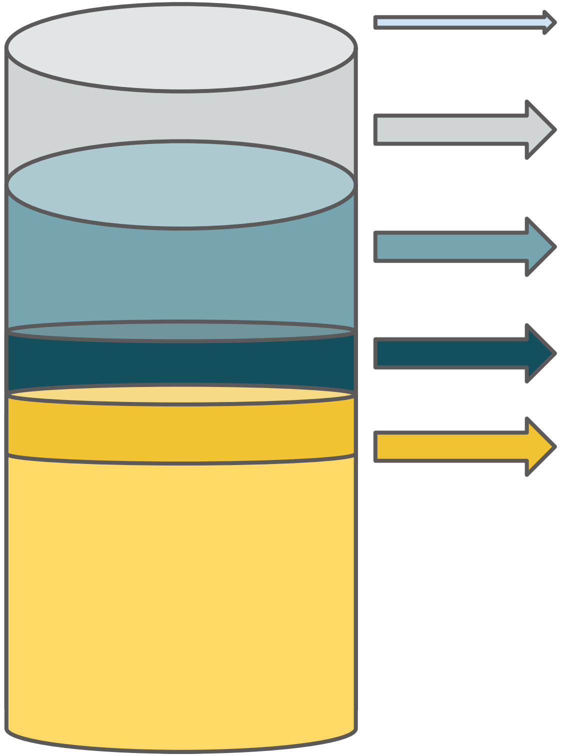
Dirty shutdown

read-only FS

Easy upgrade

Only one "Firmware"
image to push + reboot

NanoBSD: Image disk layout



MBR configurable boot-loader

Slice 1: system [224MB on BSDRP, 100MB free]

Slice 2: system (free for upgrade) [same size]

Slice 3: Configuration [15MB on BSDRP]

Slice 4: User data [15MB on BSDRP] optional and expandable if installed on disk bigger than 512MB

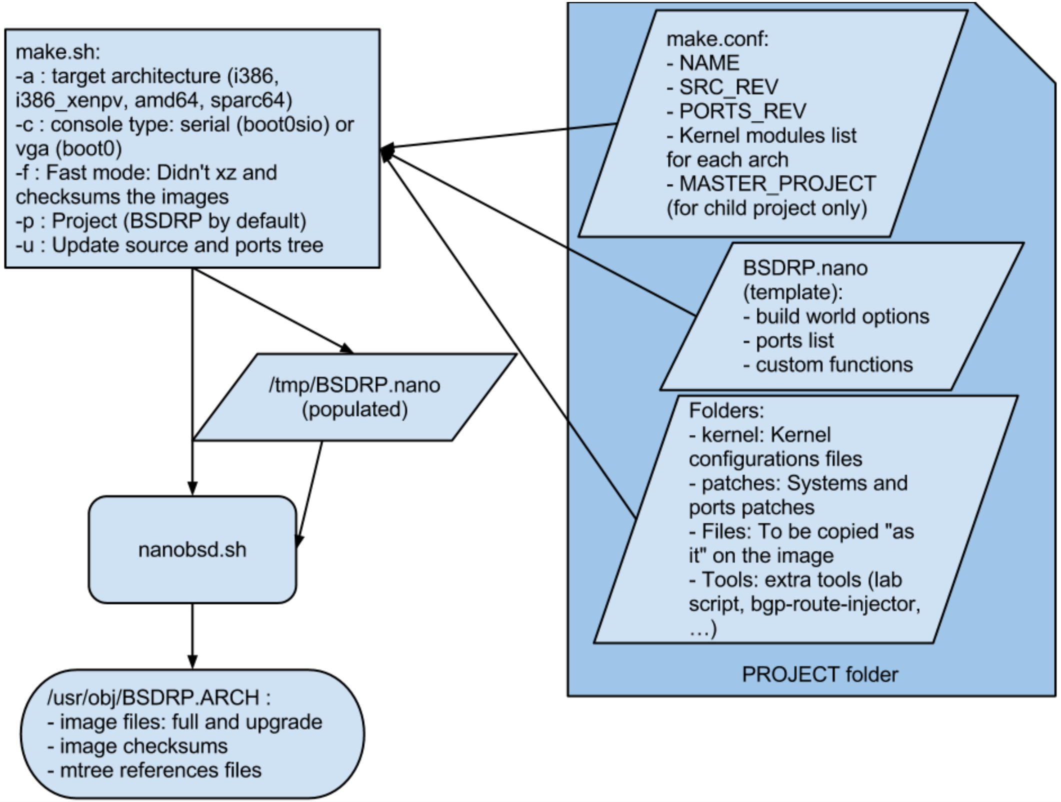
NanoBSD: system upgrade

```
$ cat new-firmware.img | ssh nanobsd upgrade
```

NanoBSD: Generating disk image

```
# Included in FreeBSD sources
cd /usr/src/tools/tools/nanobsd
# Set a custom name (default is “full”)
echo 'NANO_NAME="mynano"' > mynano.conf
# Use of glabel (media independent fstab)
echo 'NANO_LABEL="nanobsd"' >> mynano.conf
# Target a 2GB flash media (default size)
echo "UsbDevice generic-hdd 2000" >> mynano.conf
# Start nanobsd
sh nanobsd.sh -c mynano.conf
# Wait about 2 hours and install image on flash disk
dd if=/usr/obj/nanobsd.mynanobsd/_.disk.full of=/dev/da0 bs=128k
# Or use _.disk.image for upgrading existing system
```

BSDRP: NanoBSD on steroid



make.sh:
-a : target architecture (i386, i386_xenpv, amd64, sparc64)
-c : console type: serial (boot0sio) or vga (boot0)
-f : Fast mode: Didn't xz and checksums the images
-p : Project (BSDRP by default)
-u : Update source and ports tree

/tmp/BSDRP.nano (populated)

nanobsd.sh

/usr/obj/BSDRP.ARCH :
- image files: full and upgrade
- image checksums
- mtree references files

make.conf:
- NAME
- SRC_REV
- PORTS_REV
- Kernel modules list for each arch
- MASTER_PROJECT (for child project only)

BSDRP.nano (template):
- build world options
- ports list
- custom functions

Folders:
- kernel: Kernel configurations files
- patches: Systems and ports patches
- Files: To be copied "as it" on the image
- Tools: extra tools (lab script, bgp-route-injector, …)

PROJECT folder

BSDRP: Routing features

- All routing protocols supported by Quagga and Bird
  - BGP, RIP and RIPng (IPv6), OSPF v2 and OSFP v3 (IPv6), ISIS
- Multicast
  - DVMRP (mrouted)
  - PIM Dense Mode (pimdd)
  - PIM Sparse Mode (pimd)
- Multiple FIB: 16 Routing Tables available by default
- High availability
  - CARP
  - uCARP
  - VRRP (freevrrpd)

BSDRP: Traffic Shaping Features

- Traffic shaper with IPFW+dummynet
  - FIFO
  - WF2Q+ (Weighted Fair Queue)
  - RR (Deficit Round Robin)
  - QFQ (very fast variant of WF2Q+)
- Alternate queuing with ALTQ (not supported on all NIC)
  - CBQ (Class Based Queuing)
  - RED (Random Early Detection)
  - RIO (Random Early Drop)
  - HFSC (Hierarchical Packet Scheduler)
  - PRIQ (Priority Queuing)
- Committed Access Rate with netgraph
  - Single rate three color marker (RFC 2697)
  - Two rate three color marker (RFC 2698)
  - RED-like
  - Traffic shaping with RED

BSDRP: Other features

- VPN
  - IPSec (IKEv1 and IKEv2) with StrongSwan
  - SSL with OpenVPN
  - PPP with MPDv5: PPTP, PPPoE, L2TP, MLPPP, etc…
- Services
  - DHCP relay (dhcprelya) and Server (ISC)
  - NAT64 (Tayga)
  - netmap: ipfw (bride-mode only), packets generator/receiver
- Monitoring
  - Netflow (v5 and v9)
  - Process monitoring (monit)
  - SNMP (bsnmp)
- Tuned for routing

Benchmarking a router

- Router job: Forward packets between its interfaces at maximum rate

Benchmarking a router

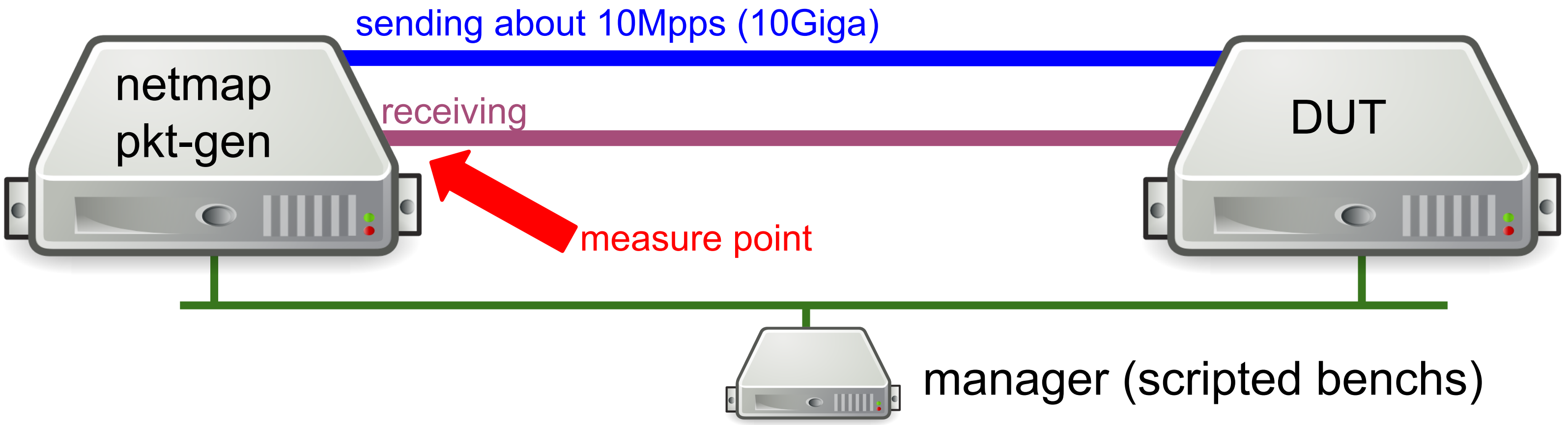
- Router job: Forward packets between its interfaces at maximum rate
  - Reference value is the **Packet Forwarding Rate** in packets-per-second (pps) unit
  - It’s **NOT** a bandwidth in bit-per-second (bps) unit !

Benchmarking a router

- Router job: Forward packets between its interfaces at maximum rate
  - Reference value is the **Packet Forwarding Rate** in packets-per-second (pps) unit
  - It’s **NOT** a bandwidth in bit-per-second (bps) unit !
- Some line-rate references
  - 1.48Mfps: Maximum Gigabit Ethernet
  - 14.8Mfps: Maximum TenGigabit Ethernet
- Full bench should follow RFC 2544 “Benchmarking Methodology for Network Interconnect Devices”

Benchmarking a router: Simplest lab

1. Measuring PPS forwarded with
   - smallest packet size: It’s the worse case
   - At maximum link rate



Bandwidth estimation from PPS

2. Do some stats with ministat(1)

```
$ ministat -s -w 60 before-tuning after-tuning
x before-tuning
+ after-tuning
+------------------------------------------------------------+
|x      *  x           *      +            + x              +|
| |________M______A_______________|                          |
|            |________________M__A___________________|       |
+------------------------------------------------------------+
    N           Min           Max        Median           Avg        Stddev
x   7            50           750           200           300     238.04761
+   5           150           930           500           540     299.08193
No difference proven at 95.0% confidence
```

Bandwidth estimation from PPS

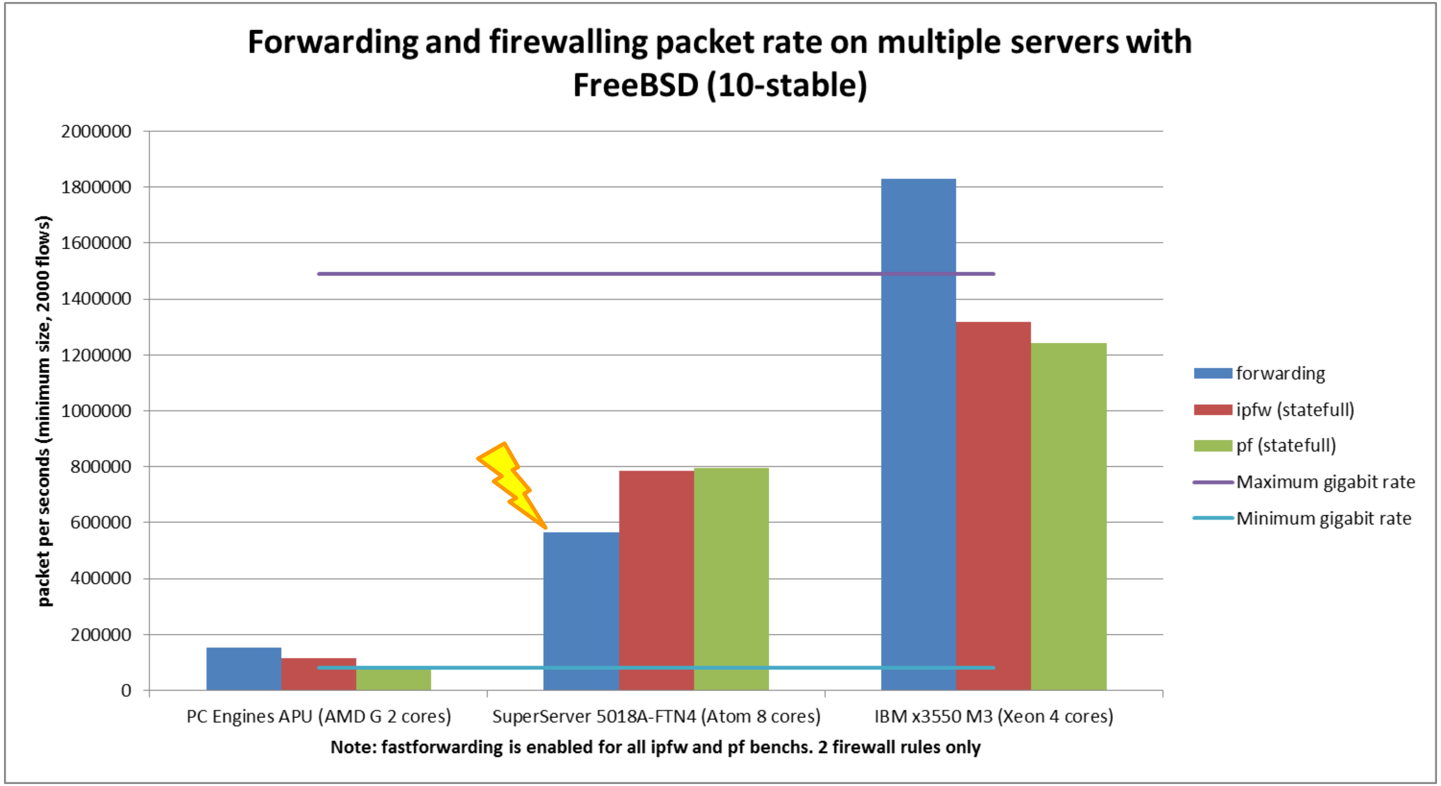
3. Estimate bandwidth (bit-per-second) using Internet Mix (IMIX) packet size distribution
    - IP layer

      ```
      PPS*( 7*40 + 4*576 + 1500 )/12*8
      ```

    - Ethernet layer

      ```
      PPS*(7*(40+14)+4*(576+14)+(1500+14))/12*8
      ```

Performance / hardware



Note: fastforwarding is enabled for all ipfw and pf benchs. 2 firewall rules only

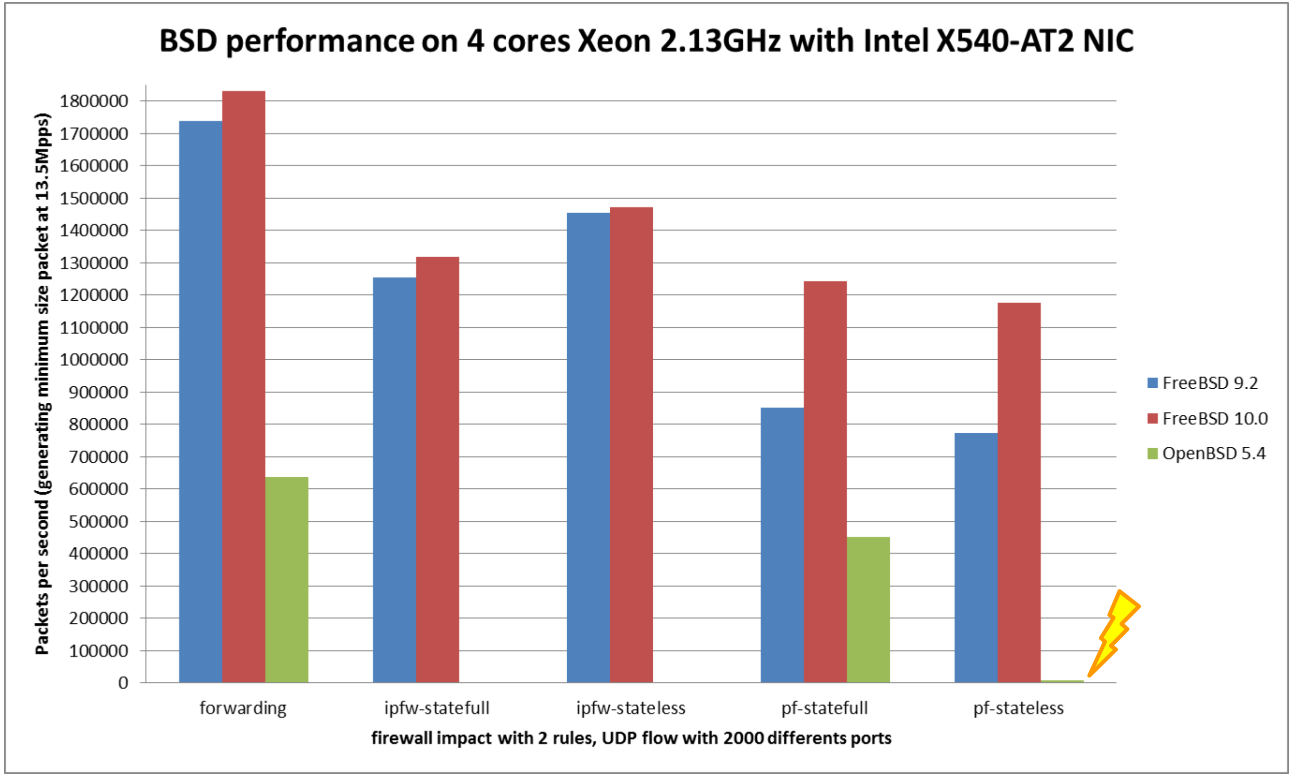
IMIX estimation
(Ethernet bandwidth)

1.81 Mpps = 5 Gb/s
1.31 Mpps = 3.7 Gb/s
1.22 Mpps = 3.4 Gb/s

566 Kpps = 1.6 Gb/s
784 Kpps = 2.2 Gb/s
796 Kpps = 2.2 Gb/s

154 Kpps = 436 Mb/s
114 Kpps = 324 Mb/s
88 Kpps = 250 Mb/s

Performance / BSD releases



IMIX estimation
(Ethernet bandwidth)

forwarding

1.74 Mpps = 4.9 Gb/s

1.81 Mpps = 5 Gb/s

638 Kpps = 1.8 Gb/s

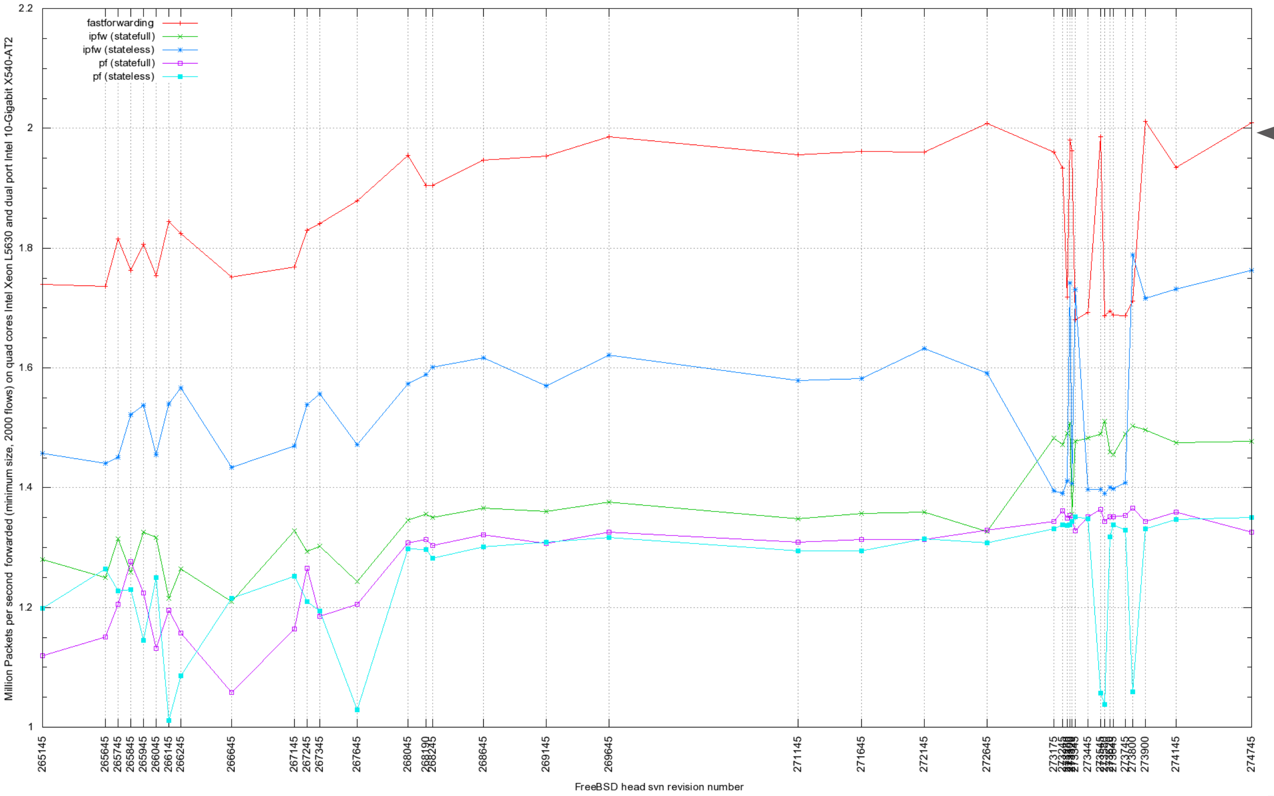
pf-stateful

851 Kpps = 2.4 Gb/s

1.24 Mpps = 3.51Gb/s

452 Kpps = 1.28 Gb/s

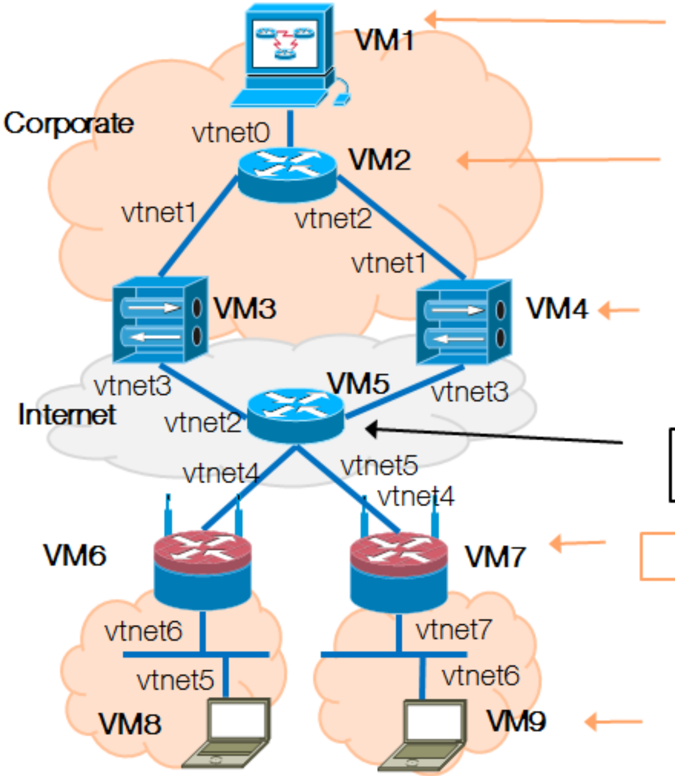
Performance / time



Should be lot's more once projects/routing will be merged to HEAD ("*with some locking modifications is able to forward 8-10MPPS on something like 2xE2660*")

Start: 30th April 2014
End: 20th Nov. 2014

Virtual Lab



Internal host

Internal OSPF router

VPN Gateways
(OSPF router on internal side)

Internet router

VPN-Routers

Remote hosts

Virtual Lab

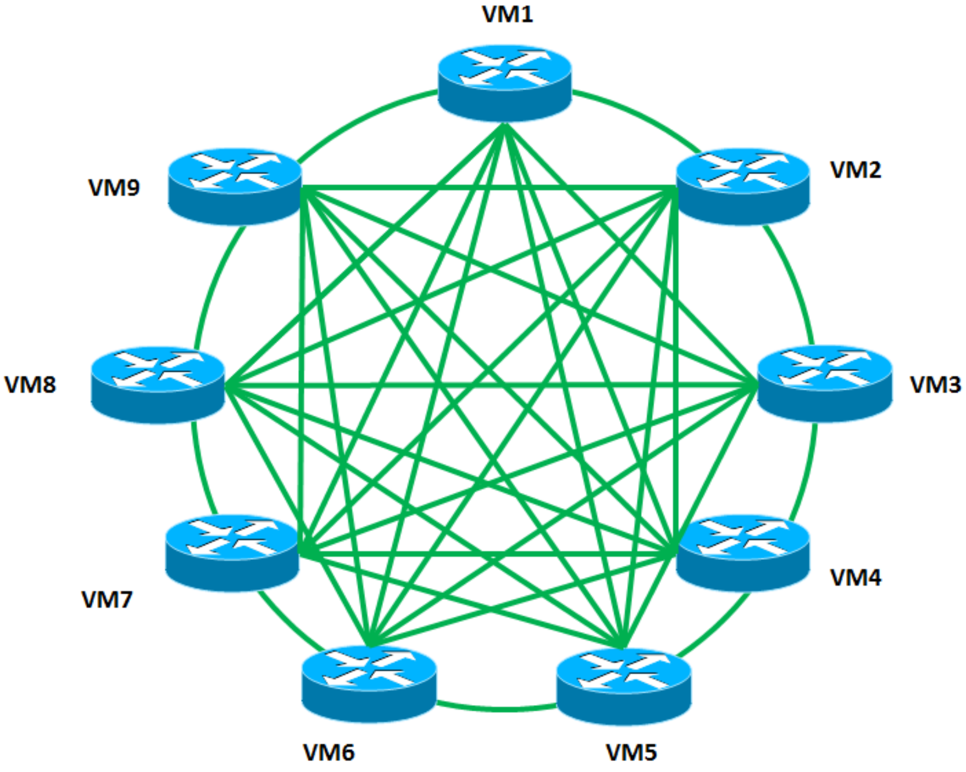
- Shell scripts provided for multiple hypervisors
    - Bhyve
    - VirtualBox (even a powershell script!)
    - Qemu/KVM
- Allow setup full-meshed lab in one command line

Virtual Lab

```
$ BSDRP-lab-bhyve.sh -i BSDRP-1.54-full-amd64-serial.img -n 9
BSD Router Project (http://bsdrp.net) - bhyve full-meshed lab script
Setting-up a virtual environment with 9 VM(s):
(etc…)
VM 1 have the following NIC:
- vtnet0 connected to VM 2.
- vtnet1 connected to VM 3.
- vtnet2 connected to VM 4.
- vtnet3 connected to VM 5.
- vtnet4 connected to VM 6.
(etc…)
VM 2 have the following NIC:
- vtnet0 connected to VM 1.
(etc…)
```

bhyve is light: Live demo running smoothly 9 BSDRP VMs on a PC Engines APU (AMD G-T40E, 1Ghz dual core, 4Go of RAM)

Virtual Lab



Roadmap

- Being SCM ready/compliant
  - We can’t add all SCM clients…but we need to provide maximum compatibilities
  - Python (Ansible) or Ruby (Puppet, Chef) based
    - RUN DEPS packages size are huge! (*need to upgrade from 512MB size image to 1GB*)
  - CFengine client is very light
- Carefully following these projects
  - FreeBSD MPLS Implementation project
  - DXR+netmap prototype

http://bsdrp.net

Questions ?

http://bsdrp.net

THANKS!