MP-safe Networking in NetBSD

Ryota Ozaki <ozaki-r@iij.ad.jp>
Kengo Nakahara <k-nakahara@iij.ad.jp>

BSDCan 2017
2017-06-09

Contents

- Status report
  - Current status
  - Ongoing tasks
- Development process
  - ATF tests
  - Performance measurement infrastructure
- Future work

Current Status of the Project

- Many components of Layer 3 and below are MP-safe
  - `src/doc/TODO.smpnet` lists what are already MP-safe and what's not
- The big locks are still there by default
  - The kernel lock and softnet_lock
  - `NET_MPSAFE` kernel option omits them
- Stable enough for daily use as a router
  - Kernels with `NET_MPSAFE`

MP-safe Network Components (1/2)

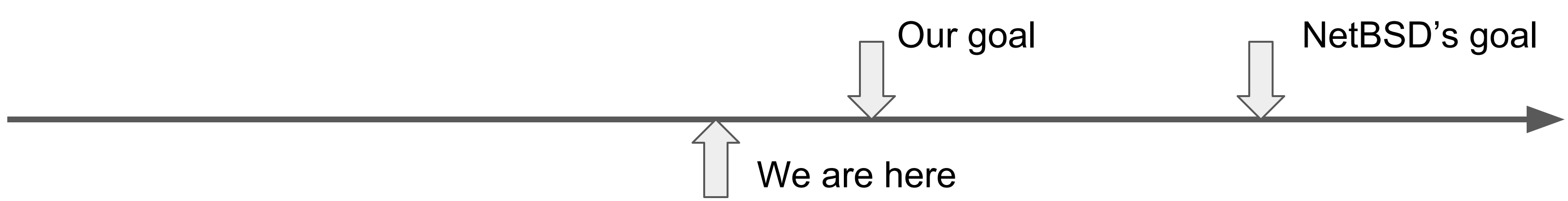
- Network device drivers
  - wm(4), vioif(4), vmx(4), ixg(4) and ixv(4)
  - Hardware multi-queue support
    - Except for vioif(4)
- Layer 2
  - Ethernet
  - bridge(4)
  - Fast forward

MP-safe Network Components (2/2)

- Layer 3
  - Routing table, IP addresses, ARP/ND, etc.
  - Except for MPLS and some options such as MROUTING
- Pseudo interfaces
  - gif(4), l2tp(4), pppoe(4), tun(4) and vlan(4)
- Others
  - pfil(9) and npf(7)
  - bpf(4)

Remaining Works

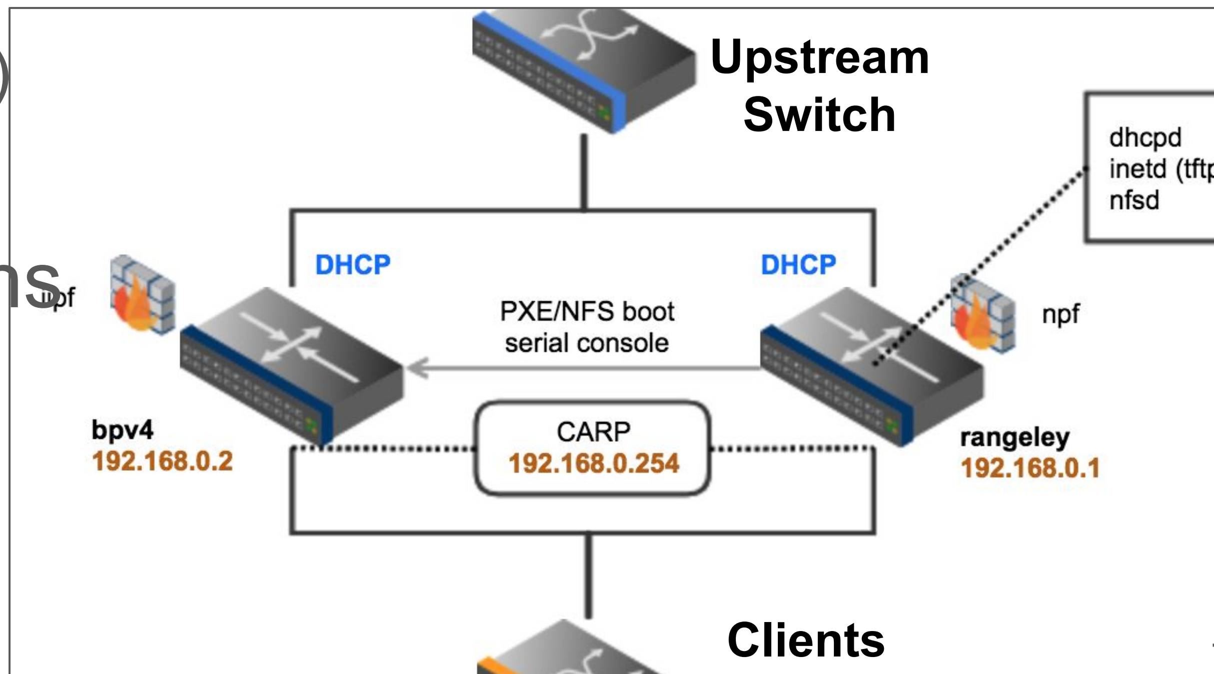
- Lots of components are still not MP-safe...
- Our targets (i.e., will be MP-safe in the near future)
  - ipsec(4) and opencrypto(9)
  - agr(4)
- Out of targets
  - Layer 4
  - Layer 2 other than Ethernet
  - Many pseudo interfaces such as gre(4)
  - Packet filters: ipf and pf



Stability Tests (Dogfooding)

- Two routers using CARP for redundancy
  - Both enables `NET_MPSAFE`
  - NAPT (and NAT66)
- Packet filters
  - npf and iipf(*)
- Work well

  over 3 months



(*) iipf is yet another packet filter
developed by ryo@n.o
Of course it's MP-safe

Current High-priority Tasks

- ipsec(4)
  - MP-ification
  - Pseudo interface (if_ipsec)
  - Scalability in terms of the number of SA (>1000)
- opencrypto(9)
  - Better locking
  - Optimization: direct dispatch
    - Omit a context change needed for hardware offload
    - For uses of encryption instructions or tightly-coupled coprocessors
- Hardware accelerations for opencrypto
  - Support in-kernel AES-NI
  - Support Intel QAT

Hardware Accelerations (1/2)

- Support in-kernel AES-NI
    - AES-NI (AES New Instruction) has been implemented in recent Intel and AMD CPUs
    - To accelerate AES encryption and decryption
    - These instructions use FPU registers
    - NetBSD kernel does not support to use the FPU registers in kernel
        - FreeBSD and OpenBSD already support it :-/

Hardware Accelerations (2/2)

- Support Intel QuickAssist Technology (QAT)
  - Some recent Intel SoCs such as C2000 (Rangeley) have hardware cryptographic accelerators
  - We have a driver of QAT developed for our products but it’s not MP-safe yet
  - It requires a firmware (binary blob)
    - Not sure the firmware can be included in the NetBSD source tree
      - Depends on if the redistribution is allowed or not

Development Process

Typical Development Cycle of MP-ification of a Network Component

- Learn its source code and the protocol needed for it
- Clean up the code
- MP-ify the code
- Optimize it (if needed)

Testing and Benchmarking for Development Cycle

- Learn its source code and the protocol needed for it
  - Writing tests is helpful to understand the code/protocol
- Clean up the code
  - Tests to avoid regressions
  - Benchmarking to know performance changes
- MP-ify the code
  - Tests help to know locking bugs
  - Benchmarking tells performance degradations
- Optimize it (if needed)
  - Of course needs benchmarking

Tools for Testing and Benchmarking

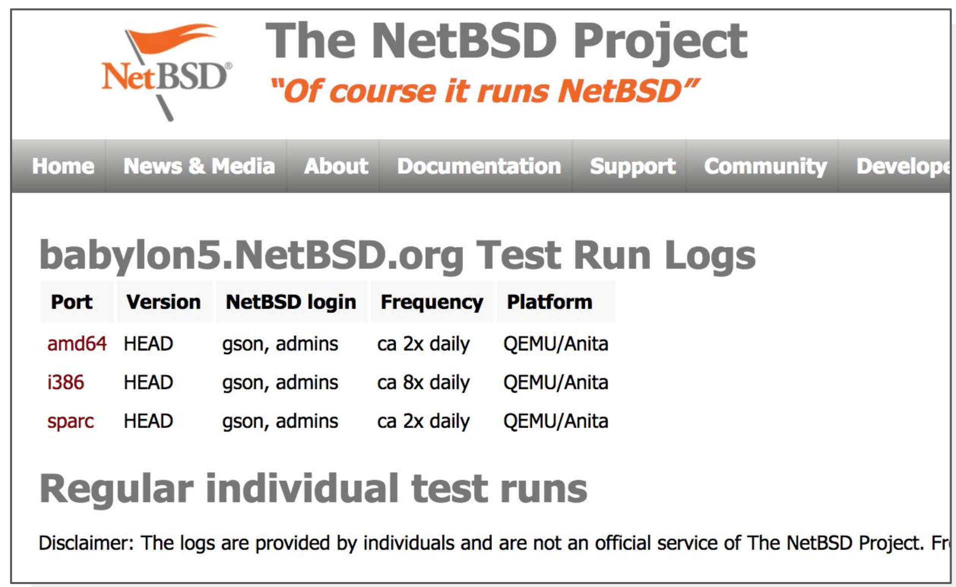
- **ATF tests** for testing
- **iiperf** and **iigraph** for benchmarking
    - We have developed them

What’s ATF

- ATF: Automated Testing Framework
- A set of utilities for writing and running tests
- APIs for C/C++/shell
- Platform independent
    - It can be run on platforms other than NetBSD
        - Not all tests are valid
- Isolated testing environment utilizing rump kernels
    - NetBSD specific

ATF Tests for NetBSD

- NetBSD has a collection of test cases for userland programs, libraries and kernel subsystems
- >6,000 test cases
- Daily/Weekly runs for -current and releases on multiple architectures



http://releng.netbsd.org/test-results.html

Motivations to Write Tests

- Automation
- Learning how components behave
- Code changes without regressions
- Testing by anyone
- Quick checks
- Easy debugging (for MP-ifications)

ATF Tests Written for the Project

- >400 test cases for networking have been added since the release of NetBSD 7
  - NetBSD-7: 199
  - NetBSD-current: 612 (as of 2017-05-29)
- `src/tests/net`
  - **`arp`** `bpf` `bpfilter` `bpfjit` **`can`** `carp` `config` `fdpass` `icmp` `if` **`if_bridge`** **`if_gif`** **`if_l2tp`** `if_loop` **`if_pppoe`** **`if_tap`** **`if_tun`** **`if_vlan`** **`in_cksum`** **`ipsec`** **`mcast`** `mpls` **`ndp`** `net` `npf` `route` `sys`
    - **Red ones** are newly added directories

Examples of Test Cases

- IPv4/IPv6 forwarding
    - Includes tests for fast forwarding
- ARP
    - GARP and Proxy ARP
    - Cache expirations
    - arp(8) command options
- IPsec
    - Combinations of:
        - ESP and AH
        - Encryption/authentication algorithms
        - Tunnel mode and transport mode
        - IPv4 and IPv6

Writing Tests Using Rump Kernels

- A simple ping test

```
LIBS="-lrumpnet -lrumpnet_net -lrumpnet_netinet -lrumpnet_shmif"
SOCK1=unix://sock1; SOCK2=unix://sock2
BUS=./bus

atf_check -s exit:0 rump_server $LIBS $SOCK1
atf_check -s exit:0 rump_server $LIBS $SOCK2

export RUMP_SERVER=$SOCK1
atf_check -s exit:0 rump.ifconfig shmif0 create
atf_check -s exit:0 rump.ifconfig shmif0 linkstr $BUS
atf_check -s exit:0 rump.ifconfig shmif0 10.0.0.1/24

export RUMP_SERVER=$SOCK2
atf_check -s exit:0 rump.ifconfig shmif0 create
atf_check -s exit:0 rump.ifconfig shmif0 linkstr $BUS
atf_check -s exit:0 rump.ifconfig shmif0 10.0.0.2/24

atf_check -s exit:0 rump.ping -c 1 -n -w 3 10.0.0.1

atf_check -s exit:0 rump.halt $SOCK1
atf_check -s exit:0 rump.halt $SOCK2
```

Launching two servers

Initializing the first server

Initializing the second server

Test ping from the second to the first

Halting the servers

Helper Functions for Writing Tests for Network Components

- A simple ping test

```
SOCK1=unix://sock1; SOCK2=unix://sock2
BUS=./bus

rump_server_start $SOCK1
rump_server_start $SOCK2

rump_server_add_iface $SOCK1 shmif0 $BUS
export RUMP_SERVER=$SOCK1
atf_check -s exit:0 rump.ifconfig shmif0 10.0.0.1/24

rump_server_add_iface $SOCK2 shmif0 $BUS
export RUMP_SERVER=$SOCK2
atf_check -s exit:0 rump.ifconfig shmif0 10.0.0.2/24

atf_check -s exit:0 rump.ping -c 1 -n -w 3 10.0.0.1

rump_server_destroy_ifaces

$DEBUG && dump
cleanup
```

Launching two servers

Initializing the first server

Initializing the second server

Test ping from the second to the first

Do some common tests (e.g., destroying interfaces)

Dump network states for debugging

Halting the servers

Bonus

- Tests written in rump kernels are isolated each other
- We can run test cases in parallel
- >600 test cases finish in less than 200 sec.

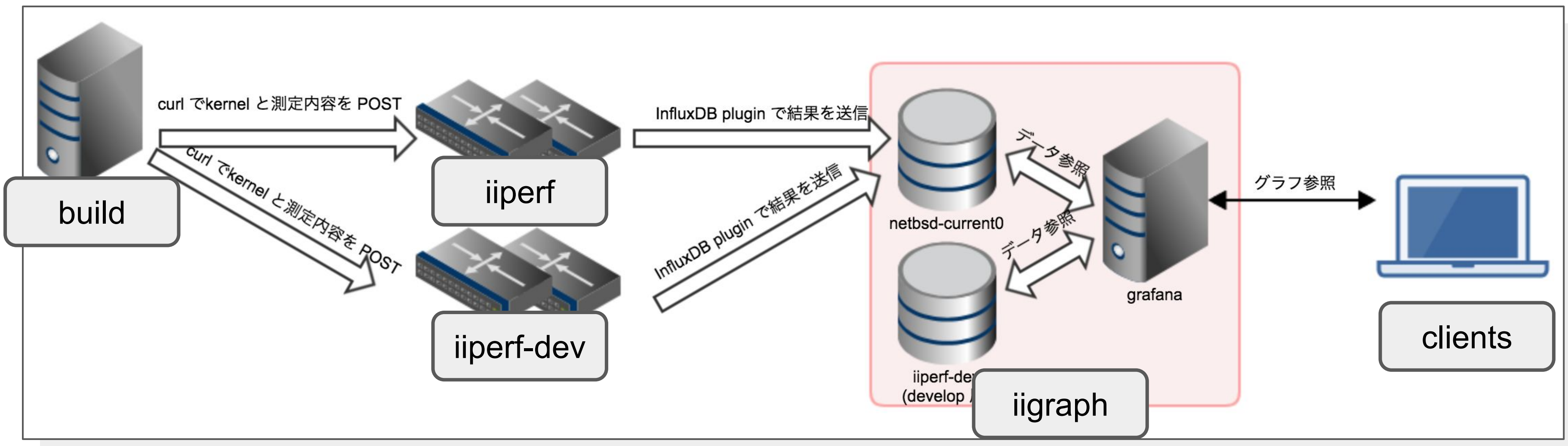
Performance Measurement Infrastructure

Requirements for Performance Measurement

- Automation
  - Environment setups
  - Measurement
  - Aggregation of results and statistics
  - Accumulation of results over a long period of time
- Detections of performance changes
  - Especially unexpected degradations
- Reproducibility
  - Each trial
  - Infrastructure itself

iiperf and iigraph

- iiperf
  - Performance measurement
- iigraph
  - Datastore and visualization



Developed and deployed by
s-yamaguchi@IIJ and suzu-ken@IIJ

Features of iiperf

- Automatic setups
  - Setup iiperf itself by Ansible
  - Setup DUTs by iiperf
- Performance measurement by *ipgen(*)*
- Gathering results and statistics between trials
  - `netstat -s` and `intrctl list`
- Posting results to iigraph and/or Wiki
- Interfaces
  - REST API for management (Web UI and CLI)
  - Web UI to see results

(*) A packet generator using netmap implemented by ryo@n.o
See https://github.com/iij/ipgen and
https://www.netbsd.org/gallery/presentations/msaitoh/2016_AsiaBSDCon/ipgen.pdf

Features of iigraph

- Datastore by InfluxDB
  - Time-series data
- Visualization by Grafana
  - Time-series graphs
  - Meta information to reproduce
    - A git commit ID of a tested kernel
    - uname -a
    - Kernel config used by the test

iiperf Measurement Parameters (1/4)

- Number of cores
  - Just 1 core
  - Iterate on 1, 2, 3 and 4 cores

iiperf Measurement Parameters (2/4)

- Number of flows
  - Change the number of flows that are delivered to a CPU by controlling values of 5-tuples
    - To evaluate scalability in terms of the number of flows
  - Flow list
    - A set of 5-tuples
  - Flow list generator
    - Generate a flow list by emulating the RSS hash value generator of a device
    - Support Intel GbE and 10GbE

iiperf Measurement Parameters (3/4)

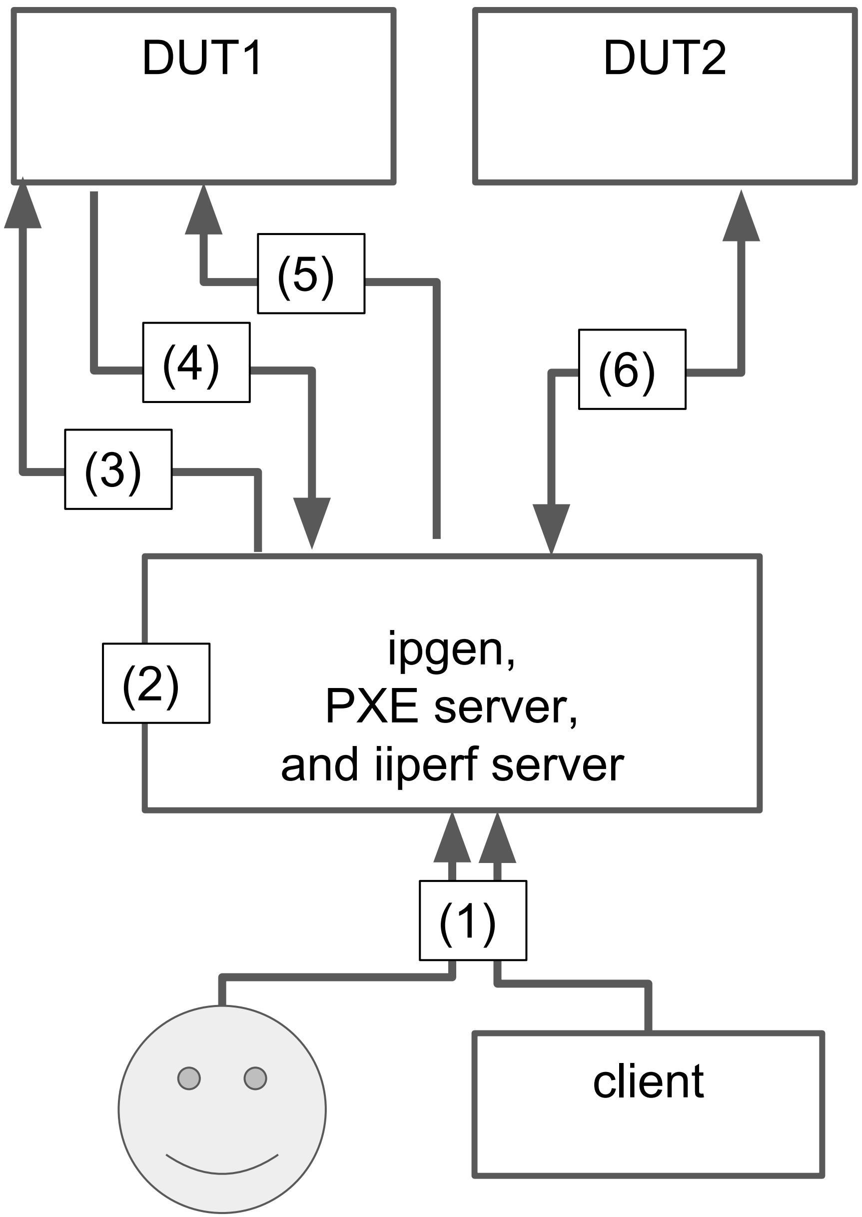
- Network configurations
    - Simple IPv4/IPv6 forwarding through one DUT
    - Simple bridging through one DUT
    - Bridging with VLAN tagging/untagging through two DUTs
    - Tunneling over gif/l2tp/IPsec through two DUTs
    - PPPoE (upward/downward) between Two DUTs

NOTE: Tunneling protocols use multiple tunnels between DUT1 and DUT2 to measurement scaling during tunnels. Scaling during flows in single tunnel is future work.

iiperf Measurement Parameters (4/4)

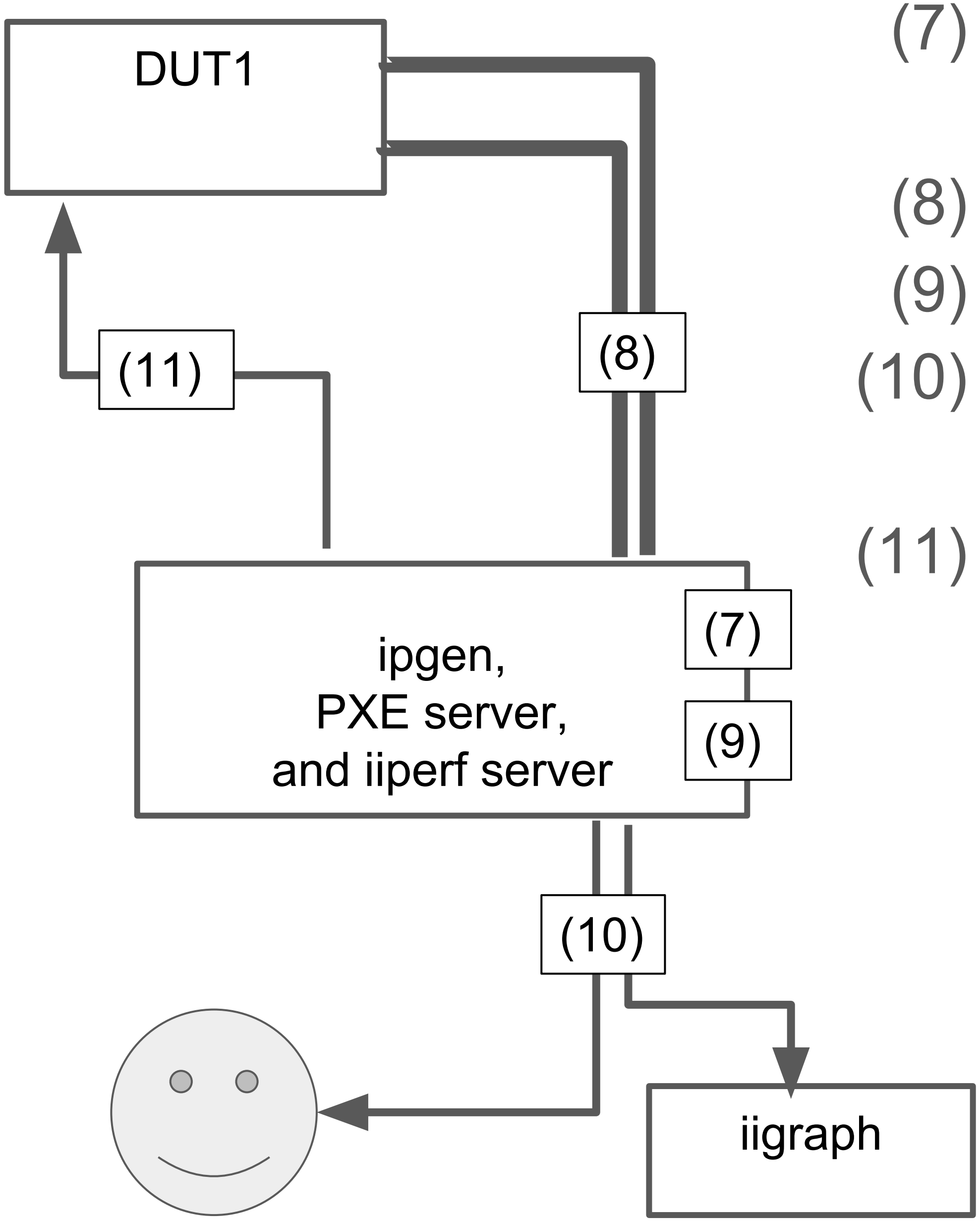
- Evaluation methods
  - High rate short packets
    - 64 bytes (for IPv4) and 66 bytes (for IPv6)
    - 100Mbps to 1Gbps
  - RFC 2544 throughput
    - A method to evaluation throughput of a router by changing offered traffic with bisecting
      - Increase offered traffic if no packet dropped, decrease otherwise
    - Variable trial duration times
    - Variable tolerable error rates

Procedures of a Run (1/2)



(1) Register a job via Web UI or REST API
  (a) Test parameters
  (b) A DUT kernel
(2) Setup PXE boot for DUTs
(3) Reboot a DUT via ssh
(4) The DUT boots via PXE
(5) Setup the DUT via ssh
(6) (if needed) Repeat (3) - (5) to DUT2

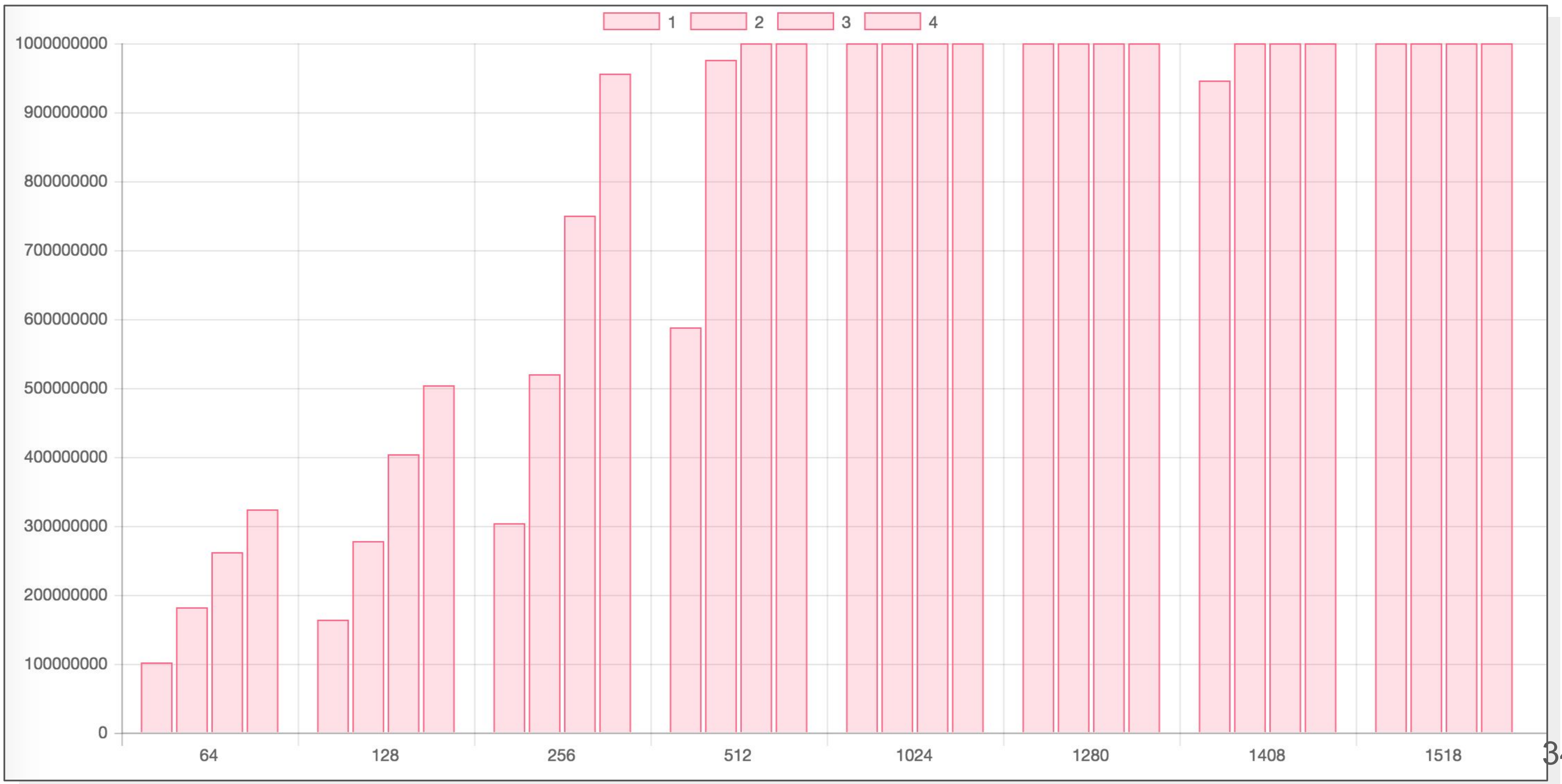
Procedures of a Run (2/2)



(7) Generate a flow list and ipgen parameters
(8) Run ipgen
(9) Parse the result
(10) Show and/or send the result
(11) Cleanup DUT(s) via ssh

An Example of a Result of iiperf

- IPv4 forwarding
- RFC 2544 throughput
- 1 core up to 4 cores

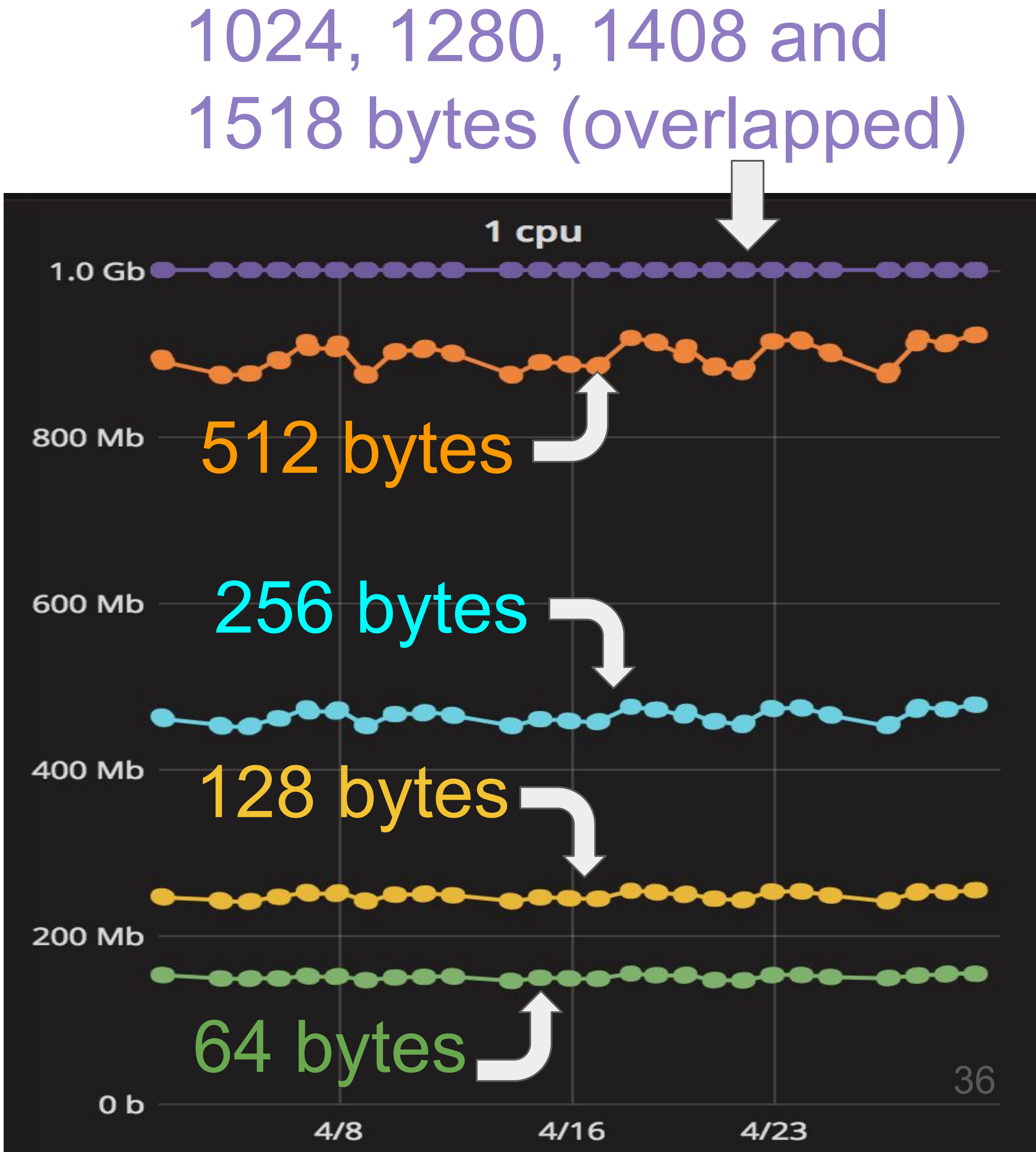


Results of Runs for a Month

2017-04-29 23:35:40	640632ef068000ad83221f6e2525337ae7f27764	NetBSD dut1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #63: Sat Apr 29 22:00:10 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-29 22:51:20	640632ef068000ad83221f6e2525337ae7f27764	NetBSD DUT1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #63: Sat Apr 29 22:00:10 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-29 22:50:00	640632ef068000ad83221f6e2525337ae7f27764	NetBSD dut1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #63: Sat Apr 29 22:00:10 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-29 00:24:40	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD dut1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-29 00:21:30	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD DUT1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-28 23:37:00	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD dut1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-28 23:35:10	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD DUT1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-28 23:35:00	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD DUT1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@ hostname /disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64

Performance Changes for a Month

- Results of Apr 2017
- RFC 2544 throughput
- IPv4 forwarding
- 1 core



Meta Information of Each Run

- Date time
- Git revision
- Kernel uname (include kernel config file name)

kernel info

Time	commit id	uname
04-28 23:35:00	4e92ac179c9d109cd4577c2bf42019a47014802d	22:00:25 JST 2017 s-yamaguchi@hostname:/disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-28 22:50:30	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD dut1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@hostname:/disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-28 22:50:20	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD dut1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@hostname:/disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-28 22:48:50	4e92ac179c9d109cd4577c2bf42019a47014802d	NetBSD DUT1 7.99.71 NetBSD 7.99.71 (BPV4_NETMPSAFE) #62: Fri Apr 28 22:00:25 JST 2017 s-yamaguchi@hostname:/disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64
2017-04-28 00:23:00	c542c99a04cb7d0d4eccda1f54be722172a9f5af	NetBSD dut1 7.99.70 NetBSD 7.99.70 (BPV4_NETMPSAFE) #61: Thu Apr 27 22:00:09 JST 2017 s-yamaguchi@hostname:/disk1/home/s-yamaguchi/netbsd-mp/work.iiperf/obj/sys/arch/amd64/compile/BPV4_NETMPSAFE amd64

Future Work

- Complete tasks of ipsec(4) and opencrypto(9)
- Improve scalability
  - The number of flows
  - The number of SAs on IPsec
- Improve single-thread performance
  - E.g., optimize psref(9)
- `NET_MPSAFE` by default
  - until NetBSD 9...?

BACKUP