Made to Measure

George Neville-Neil Jim Thompson

vBSDCon 2015

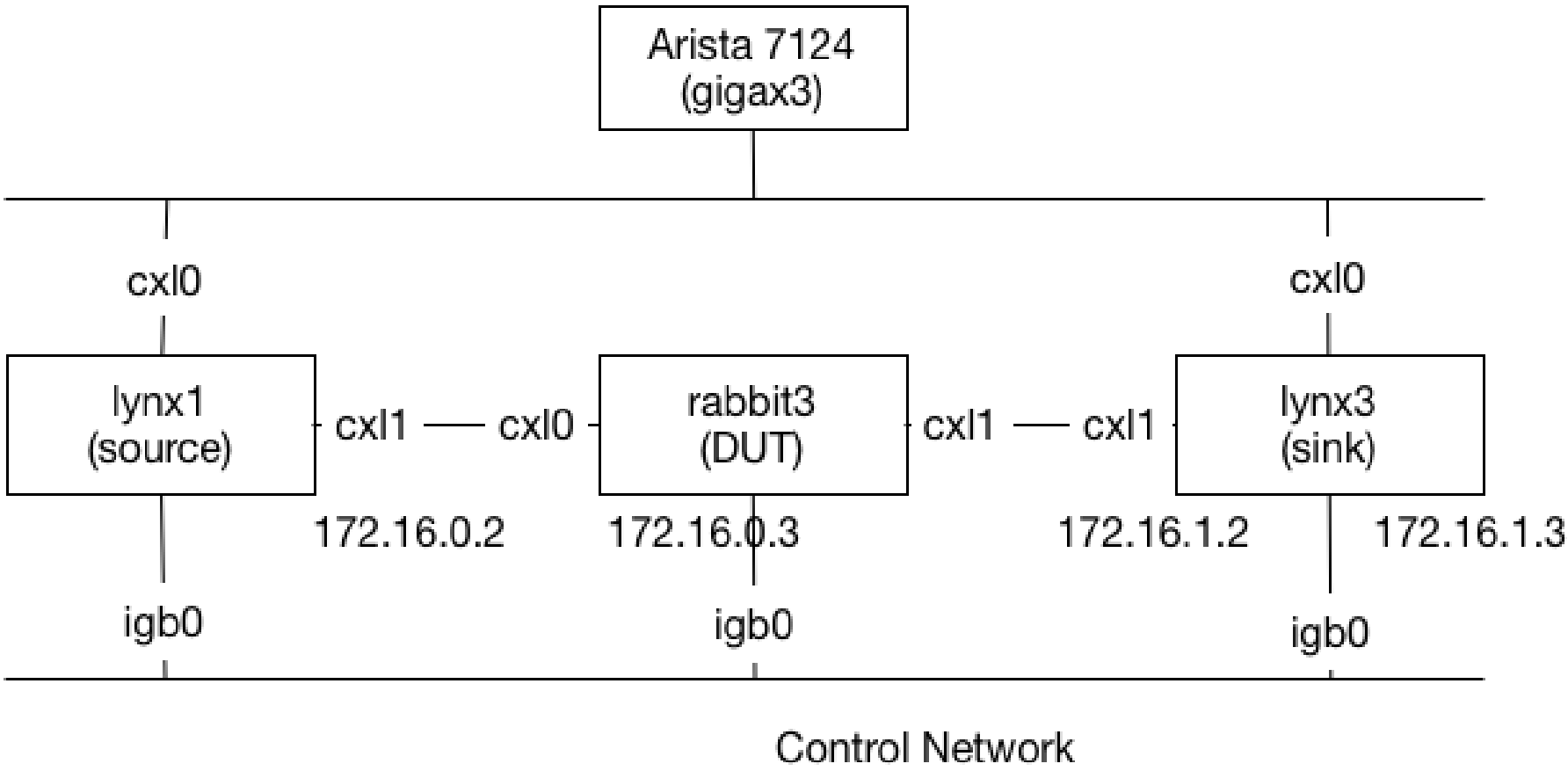
Benchmarks are Hard

- What do we measure?
- How do we measure it?
- How do we verify our measurements?
- Can our measurement be repeated?
- Can our measurement be replicated?
- Is our measurement relevant?
- How do we generate a workload?
- Does our measurement technology disturb the measurement?
  - Heisentesting

Network Benchmarks are Harder

- Asynchrony
- Best effort delivery
- Lack of open source test tools
- Control of distributed systems

Lab Setup


Arista 7124
(gigax3)
cxl0
cxl0
lynx1
(source)
cxl1
cxl0
rabbit3
(DUT)
cxl1
cxl1
lynx3
(sink)
172.16.0.2
172.16.0.3
172.16.1.2
172.16.1.3
igb0
igb0
igb0
Control Network

Hardware Used

lynx1/lynx3	dual socket, 10 core, 2.8GHz E5-2680 Xeon processors
rabbit3	single socket, four core, 3GHz E5-2637 Xeon processor
NIC	Chelsio T520, dual port, 10G NIC
Switch	Arista 7124 10G switch

Modern Hardware

- 10Gbps is 14.8 million 64 byte packets per second
- 67.5ns per packet or 200cycles at 3GHz
- Cache miss is 32ns
- Multi-core
- Multi-queue
- Lining it all up

Test Automation: Conductor

- Set of Python libraries
- *Conductor* and 1, or more, *Players*
- Four Phases

Startup	Set up system, load drivers, set routes, etc.
Run	Execute the test
Collect	Retrieve log files and output
Reset	Return system to original state

Conductor Config

```
# Master config file to run an iperf test WITHOUT PF enabled.
[Test]
trials: 1

[Clients]
# Sender
client1: source.cfg
# DUT
client2: dut.cfg
# Receiver
client3: sink.cfg
```

Player Config

```
[Master]
player: 192.168.5.81
conductor: 192.168.5.1
cmdport: 6970
resultsport: 6971

[Startup]
step1:ifconfig ix0 172.16.0.2/24
step2:ifconfig ix1 172.16.1.2/24
step3:ping -c 3 172.16.0.1
step4:ping -c 3 172.16.1.3

[Run]
step1:echo "running"
step2:pmcstat -O /mnt/memdisk/pktgen-instruction-retired.pmc -S instruction-retired -l 25

[Collect]
step1:echo "collecting"
step2:mkdir /tmp/results
step3:cp -f /mnt/memdisk/pktgen-instruction-retired.pmc /tmp/results/
step4:pmcstat -R /tmp/results/pktgen-instruction-retired.pmc -G \
          /tmp/results/pktgen-instruction-retired.graph
step5:pmcstat -R /tmp/results/pktgen-instruction-retired.pmc -D /tm/results -g
step6:pmcannotate /tmp/results/pktgen-instruction-retired.pmc \
            /boot/kernel/kernel > /tmp/results/pktgen-instruction-retired.ann

[Reset]
step1:echo "system reset: goodbye"
```

Host to Host Baseline Measurement

iperf TCP based test

netperf Packet based test using `netmap(4)`

Baseline TCP Measurement

```
0.00-1.00   sec 1.09 GBytes 9.41 Gbits/sec
1.00-2.00   sec 1.10 GBytes 9.41 Gbits/sec
2.00-3.00   sec 1.10 GBytes 9.41 Gbits/sec
3.00-4.00   sec 1.10 GBytes 9.41 Gbits/sec
4.00-5.00   sec 1.10 GBytes 9.41 Gbits/sec
5.00-6.00   sec 1.10 GBytes 9.42 Gbits/sec
6.00-7.00   sec 1.10 GBytes 9.41 Gbits/sec
7.00-8.00   sec 1.10 GBytes 9.41 Gbits/sec
8.00-9.00   sec 1.10 GBytes 9.41 Gbits/sec
9.00-10.00 sec 1.10 GBytes 9.41 Gbits/sec
```

Baseline pkt-gen Measurement

- Source

```
827.257743 main_thread [1512] 14697768 pps
828.259812 main_thread [1512] 14668997 pps
829.261742 main_thread [1512] 14695277 pps
830.263743 main_thread [1512] 14685547 pps
```

- Sink

```
866.466039 main_thread [1512] 11943109 pps
867.468024 main_thread [1512] 11946111 pps
868.469126 main_thread [1512] 11942020 pps
869.471027 main_thread [1512] 11939957 pps
```

Baseline Discussion

- TCP uses full sized packets
- pkt-gen uses minimum sized (64 byte) packets
- The DUT cannot quite keep up

IPSec and its Algorithms

- Encryption is computationally expensive
- Offloaded co-processors
- On chip instructions `AES-NI`

Measurement Methods

- Two host setup
- iperf3 using TCP
- Conductor sets up the tests
- 10 rounds of 10 seconds each

Baseline

- Using NULL methods
- No authentication or encryption
- No TCP offload on the NIC cards

Min	Max	Median	Avg	Stddev
2.24	2.48	2.25	2.2844444	0.079390036

Authentication

- HMAC-SHA1 authentication
- Transport mode
- No encryption

Min	Max	Median	Avg	Stddev
615	632	628	623.3	7.9867947

3DES with MD5 HMAC

- 3DES Encryption with MD5 Authentication
- Still commonly used

Min	Max	Median	Avg	Stddev
143	144	144	143.9	0.31

AES-CBC

- Tunnel Mode
- Encryption only, as a baseline, do *NOT* do this.
- Results for 128 and 256 bits with and without hardware support
- Still called “rijndael” in FreeBSD

HW Suppt	Min	Max	Median	Avg	Stddev
N	729	746	742	738.9	6.0
Y	1120	1190	1160	1156	0.02

AES-CBC with SHA1

- Tunnel Mode
- Encryption with SHA1 authentication

HW Suppt	Min	Max	Median	Avg	Stddev
N	393	420	400	403.59	8.55
Y	303	418	400	400	11

AES-GCM

- Tunnel Mode
- Both encryption and authentication
- Results for 128 and 256 bits with and without hardware support

HW Suppt	Min	Max	Median	Avg	Stddev
N	273	280	276	276.55	2.12
Y	1220	1300	1270	1268.88	0.023

HW Suppt	Min	Max	Median	Avg	Stddev
N	227	261	260	213.48	98.101
Y	1070	1250	1100	1130	0.065

Overall Picture

Algorithm	Min	Max	Median	Avg	Stddev
NULL	2240	2480	2250	2284.44	0.079
HMAC-SHA1	615	632	628	623.3	7.98
AES-GCM Soft 128	273	280	276	276.55	2.12
AES-GCM Soft 256	227	261	260	213.48	98.101
AES-GCM Hard 128	1220	1300	1270	1268.88	0.023
AES-GCM Hard 256	1070	1250	1100	1130	0.065

“Fast”" Forwarding

- The network turbo button
- `net.inet.ip.fastforwarding`

Forwarding Measurements

Size	TX	RX	Stddev	% Line Rate
64	14,685,502	1,069,691	165	7
128	8,215,485	1,051,849	177	14
256	4,464,323	952,227	154	21
512	2,332,123	949,432	165	41
1024	1,192,770	948,172	100	80
1500	820,229	820,215	1.44	100

Fast Forwarding Measurements

Size	TX	RX	Stddev	% Line Rate
64	14,685,502	1,093,090	634	7
128	8,215,485	1,079,852	549	14
256	4,464,323	1,273,975	141	28
512	2,332,123	1,267,776	136	54
1024	1,192,770	1,192,755	11595	100
1500	820,229	820,215	2.08	100

Why?

- Fewer function calls?
- Better code?
- How do we find out?

DTrace Analysis

- Look at packet path call graphs
- Measure time from `ether_input` to `ether_output`

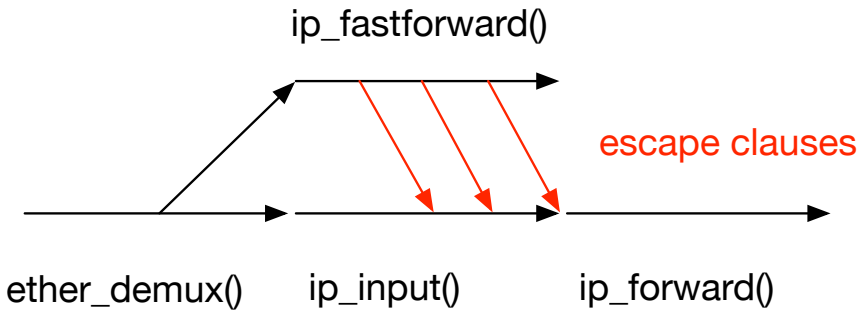
Call graph Comparison

```
1  ether_input()
2  netisr_dispatch_src()
3  ether_nh_input()
4  ether_demux()
5  netisr_dispatch_src()
6  ip_input()
7  ip_forward()
8  ip_output()
9  ether_output()
```

```
1  ether_input()
2  netisr_dispatch_src()
3  ether_nh_input()
4  ether_demux()
5  ip_fastforward()
6  ether_output()
```

Fast Forward Siding

- `ip_fastforward` makes its decision sooner



Time Analysis

- Normal Path

```
value  ------------- Distribution ------------- count
512 |                                           0
1024 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  1414505
2048 |@                                          35478
4096 |                                           481
8192 |                                           0
```

- Fast Forwarding Path

```
 value  ------------- Distribution ------------- count
512 |                                           0
1024 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  1721837
2048 |@                                          41287
4096 |                                           490
8192 |                                           0
```

DTrace Script

```
::ether_input:entry
{
        self->before = timestamp;
}
::ether_output:return
/self->before > 0/
{
        @count = quantize(timestamp - self->before);
}
tick-30sec
{
        normalize(@count, 10);
        printa(@count);
        printf("\n");
        clear(@count);
}
```

Is Fast Forwarding Free?

- Duplicates many checks of `ip_input()`
- Delays host packet processing

Cost of Fast Forwarding on TCP

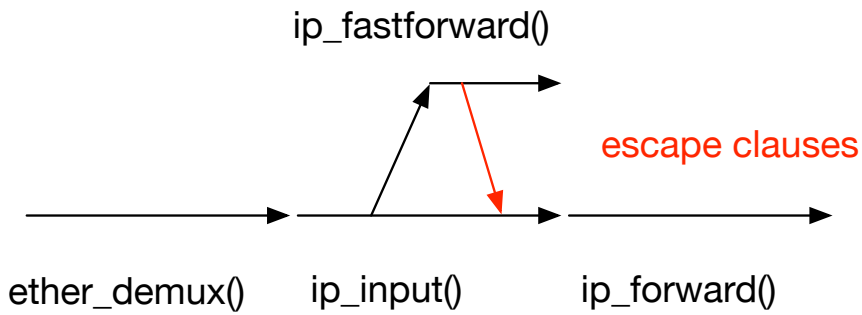
```
x iperf3-twohost-fastforward
+ iperf3-twohost-nofastforward
+-------------------------------------------------------------------------------------------+
|                                                                          +  +             |
|x x  xx          x     xx x   x x          +    +     +                  +  + +  +       +|
|    |___________A____M_____|                        |_________________A_______M______|    |
+-------------------------------------------------------------------------------------------+
    N           Min           Max        Median           Avg        Stddev
x  10          7.56          7.75          7.69         7.657   0.071499806
+  10          7.82           8.1          8.04          7.99    0.09854497
Difference at 95.0% confidence
        0.333 +/- 0.0808908
        4.34896% +/- 1.05643%
```

Combination of the Two

- We do not want to choose between “fast” and “slow”
- The `ip_fastforward` path removes functionality
- We can do better
- Remove duplicated code
- Try to forward the packet early and fall back

Try Forward Siding

- Remove duplicated code
- Try to forward then fall back



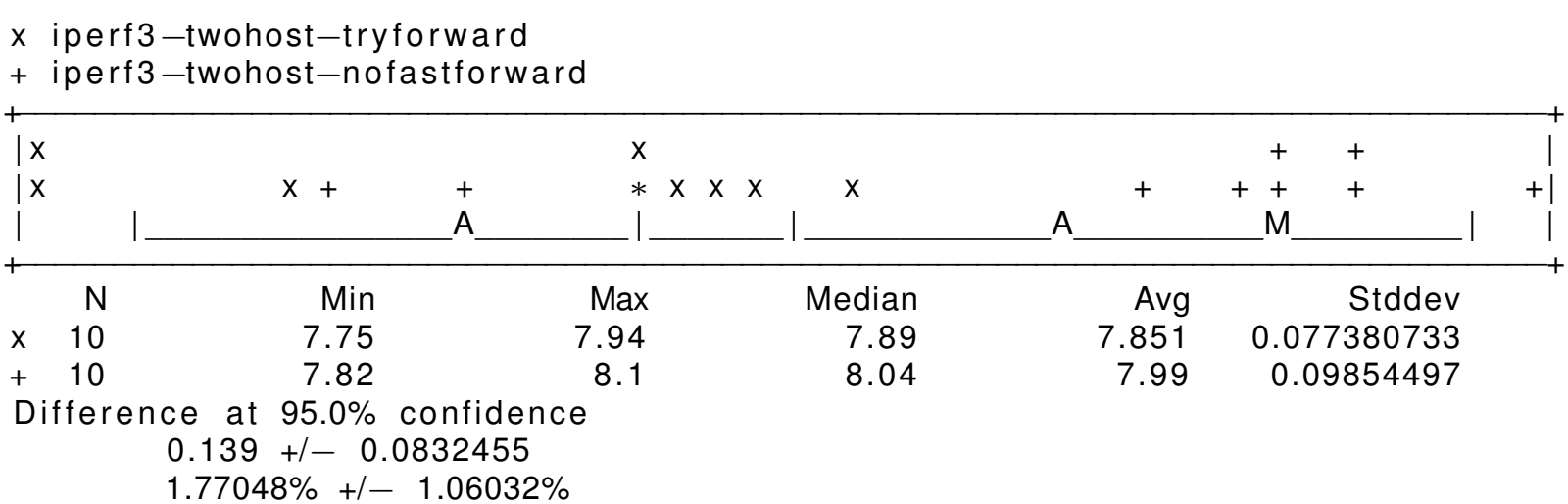
Try Forward Measurements

Size	TX	RX	Stddev	% Line Rate
64	14,685,502	1,082,963	882	7
128	8,215,485	1,083,249	373	14
256	4,464,323	1,194,525	1247	28
512	2,332,123	1,134,556	963	54
1024	1,192,770	1,090,835	1222	91
1500	820,229	820,219	21	100

Percent of Line Rate for Old, Fast, and Try

Size	Old	Fast	Try
64	7	7	7
128	14	14	14
256	21	28	28
512	41	54	54
1024	80	100	91
1500	100	100	100

Impact of Try Forward



An Ongoing Longitudinal Study

- AsiaBSDCon: March, 2015
- BSDCan: June, 2015
- Conducted continuously
- Reported several times per year
- Covering more subsystems
- Next Update: EuroBSDCon 2015

Where to get it all

Netperf `http://github.com/gvnn3/netperf`

- Includes scripts and results

Conductor `http://github.com/gvnn3/conductor`

- The test framework

pfSense `http://www.pfsense.org`

FreeBSD `http://www.freebsd.org`

Raj Jain *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*