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

Modern Hardware

- 100 Gbps is 148 million 64 byte packets per second
- 6.75ns per packet or 20 cycles 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

iperf3 TCP based test

pktgen 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 (2) and Four (4) host setups
- iperf3 using TCP
- Conductor sets up the tests
- 10 rounds of 10 seconds each

Overall Picture

Algorithm	Min	Max	Median	Avg	Stddev
NULL	2240	2480	2250	2284.44	0.079
HMAC-SHA1	615	632	628	623.30	7.980
AES-GCM Soft 128	273	280	276	276.55	2.120
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	1130.00	0.065
NULL 4 Host	3360	3390	3380	3380.00	0.009

Where to get it all

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

- Includes scripts and results

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

- The test framework

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

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

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