Quantitative Analysis of KVM, Container, and Unikernel Environment Based on TCP/UDP Network Performance

Bronco Loves Cloud

Yinghe Chen   Xi Wang   Xiaoying Yang
Miaoyan Zhang   Mingjie Zhang

Outline

- Introduction
- Environment Setup
- Experiment
- Result
- Conclusion

Introduction

KVM (Kernel-based Virtual Machine):open source solution that converts Linux in a Type 1 hypervisor.

Docker:open source platform for the deployment and management of Linux Containers.

Unikernel(OSv):open source virtualization platform that run application directly on hardware / hypervisor.

Introduction

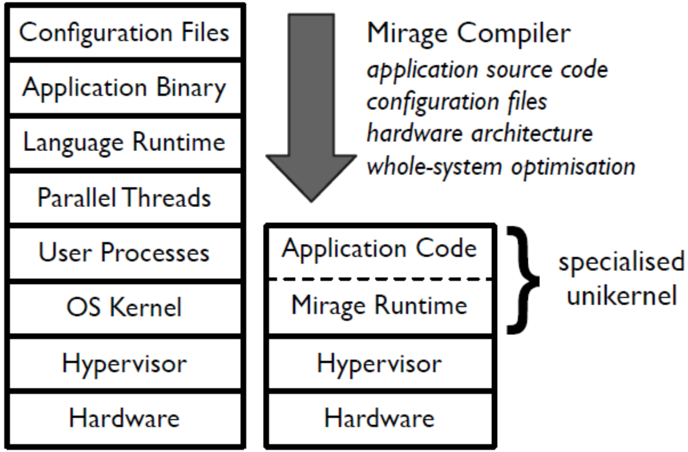


Figure 1: Contrasting software layers in existing VM appliances *vs.* unikernel's standalone kernel compilation approach.

Unikernel *vs* **KVM** *vs* **Docker**

Less image size than docker and KVM

Less startup time / overhead than kvm

Unikernel vs Linux-Network:

Throughput-higher

Transmit performance-lower

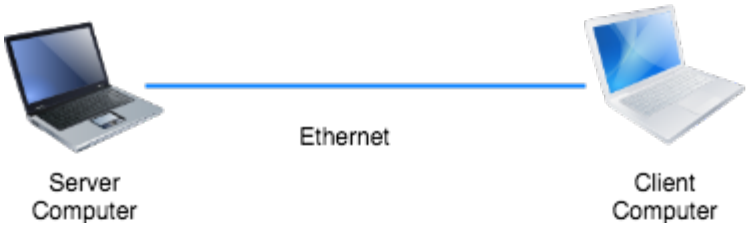
Problem

In this project, we quantitatively evaluate the network performance (TCP/UDP) for throughput using Netperf for different virtualization techniques: KVM, Container and Unikernel.

Environment Setup - Overall

Network benchmark: Netperf

One computer acts as the server, the other computer acts as the client, and the two computers are connected using ethernet directly.



Server Computer CPU	Intel(R) Pentium(R) CPU N3530 @ 2.16GHZ
Server Computer Memory	4G
Cable Speed	100Mbps

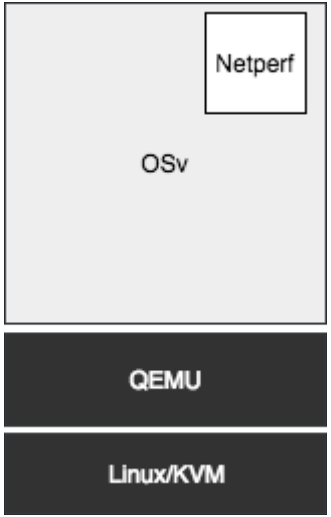
Environment Setup - Host OS

Host OS: Ubuntu 14.04.5 LTS
To have a comparison, Netperf is installed on the host OS as well.
netperf-2.7.0, from ftp://ftp.netperf.org/netperf/

```
# Install netperf
wget ftp://ftp.netperf.org/netperf/netperf-2.7.0.tar.bz2
tar xf netperf-2.7.0.tar.bz2
cd netperf-2.7.0
./configure
make

# Start netserver
cd netperf-2.7.0/src
./netserver -4
```

Environment Setup - OSv Guest

The OSv installed on top of the hypervisor QEMU-KVM.
OSv image is built from source using source code from its official Git repository.
Netperf is cross-compiled on Linux host then incorporated into OSv image.

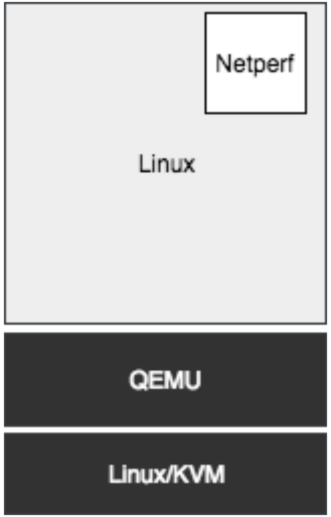


```
# Get OSv Source Code
git clone https://github.com/cloudius-systems/osv.git

# Install OSv Build Dependency
cd osv
python scripts/setup.py
git submodule update --init --recursive

# Build OSv Image
make
scripts/build

# Build netperf app
```

```
cd apps/netperf
make

# Include netserver in OSv image
cp apps/netperf/netserver.so tools/

# Add `/tools/netserver.so: tools/mkfs/
netserver.so` to usr.manifest
scripts/build # Rebuild image

# Start netserver
sudo ./scripts/run.py  -e "/tools/netserver.so -D
-4 -f" -c 4 --api
```

Environment Setup - Ubuntu Guest

To test the network performance of KVM, we install a Linux guest on top of the hypervisor QEMU-KVM. Netperf is then installed on the Linux guest.

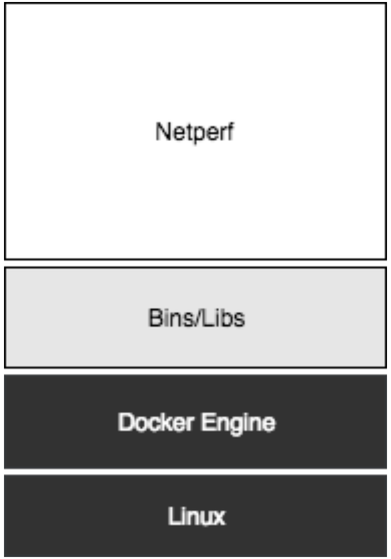


```
# Install QEMU KVM
sudo apt-get install qemu-kvm

# Create image
qemu-img create ubuntu.img 10G

# Download Ubuntu net install
http://archive.ubuntu.com/ubuntu/dists/trusty-updates/main/installer-amd64/current/images/netboot/mini.iso
```

```
# Start KVM guest with mini.iso as CD-ROM
qemu-system-x86_64 -hda ubuntu.img -cdrom mini.iso -net nic -net user

# Install Ubuntu as usual
# ...

# Start KVM guest again without CD-ROM
qemu-system-x86_64 -hda ubuntu.img -net nic -net user
# Install netperf inside guest Ubuntu as usual
```

Environment Setup - Docker

Docker is installed on the host Linux.
Netperf is installed in a docker container.



Environment Setup - Network Configuration

Netperf server: one control port of TCP 12865 and one TCP or UDP data port. Ports of TCP 12865, TCP 12866 and UDP 12866 be mapped between host physical network and hypervisor virtual network.

```
# Start KVM guest with port mapping
# TCP 12865 is Netperf control channel
# TCP 12866 is used as Netperf TCP data channel, use -P 12866 to specify
# UDP 12866 is used as Netperf UDP data channel, use -P 12866 to specify
qemu-system-x86_64 -hda ubuntu.img -net nic -net user -redir tcp:12865::12865  -redir tcp:12866::12866  -redir udp:12866::12866

# For OSv, use the same -redir options by changing scripts/run.py
# Inside scripts/run.py change `args += ["-redir", "tcp:8000::8000"]`
# To `args += ["-redir", "tcp:8000::8000", "-redir", "tcp:12865::12865", "-redir", "tcp:12866::12866", "-redir", "udp:12866::12866"]`
sudo ./scripts/run.py  -e "/tools/netserver.so -D -4 -f" -c 4 --api
```

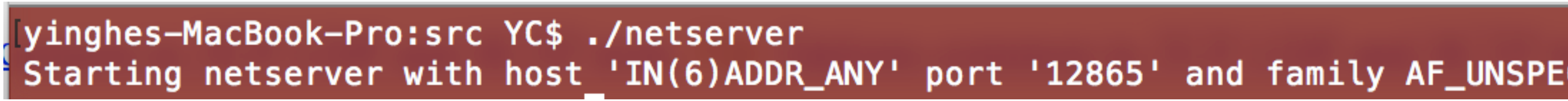
Environment Setup

All in all, we have the server with these software:

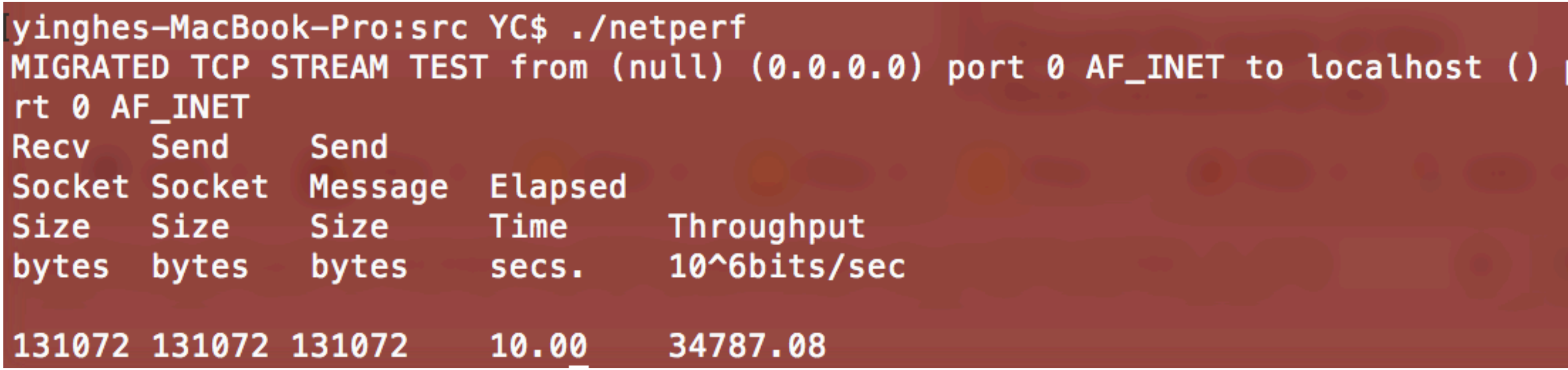
Host OS	Ubuntu 14.04.5 LTS
QEMU_KVM	Libvert 1.2.2 API: QEMU 1.2.2 Hypervisor: QEMU 2.0.0
OSv guest	v0.24
Ubuntu guest	Ubuntu 14.04.5 LTS
Docker	Docker 17.03.0-ce
Netperf	Netperf-2.7.0

Netperf Experiment: TCP_Stream Test

The TCP_Stream test is to transfer some quantity of data from the system running netperf to the system running netserver.

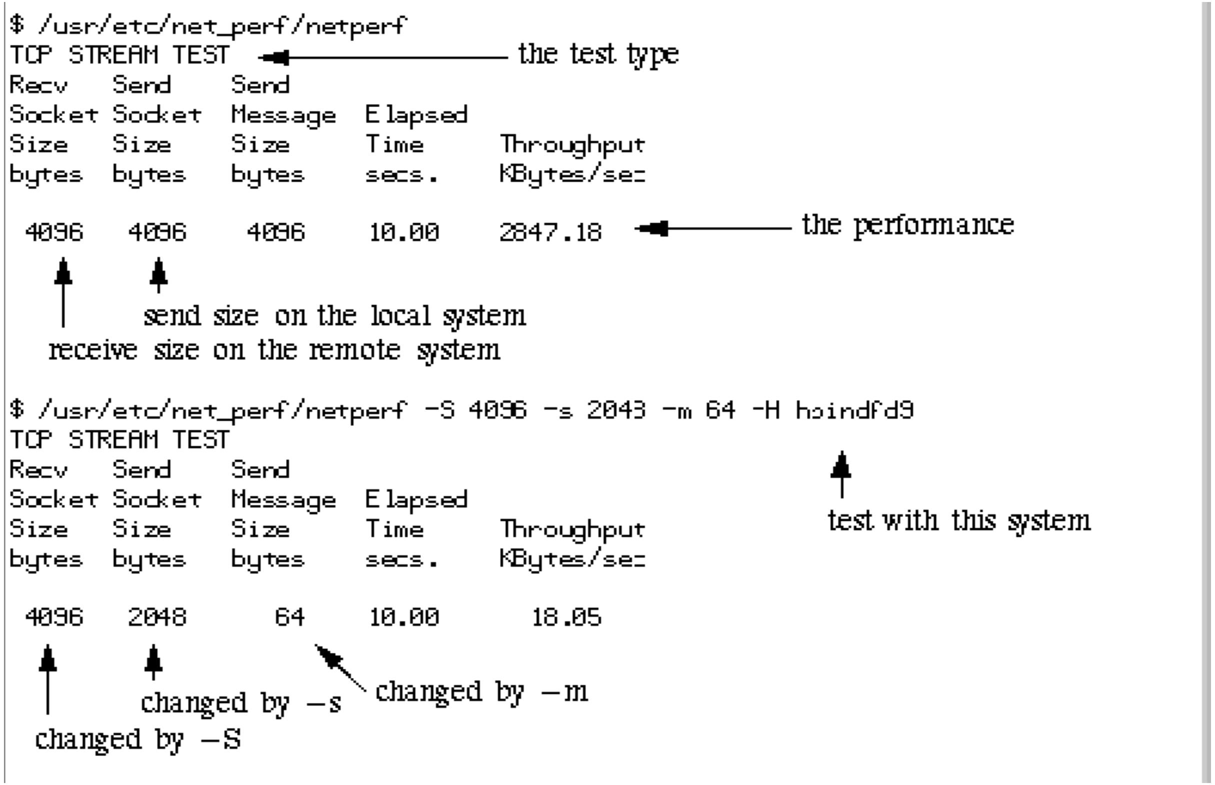


```
[yinghes-MacBook-Pro:src YC$ ./netserver
 Starting netserver with host 'IN(6)ADDR_ANY' port '12865' and family AF_UNSPE
```



```
[yinghes-MacBook-Pro:src YC$ ./netperf
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 0 AF_INET to localhost ()
rt 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

131072 131072 131072    10.00    34787.08
```

Netperf Experiment: TCP_Stream Test



```
$ /usr/etc/net_perf/netperf
TCP STREAM TEST
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    KBytes/sec

 4096   4096    4096    10.00    2847.18
```

```
$ /usr/etc/net_perf/netperf -S 4096 -s 2043 -m 64 -H hoindfd9
TCP STREAM TEST
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    KBytes/sec

 4096   2048      64    10.00      18.05
```

Netperf Experiment: TCP_Stream Test

Control Variables:

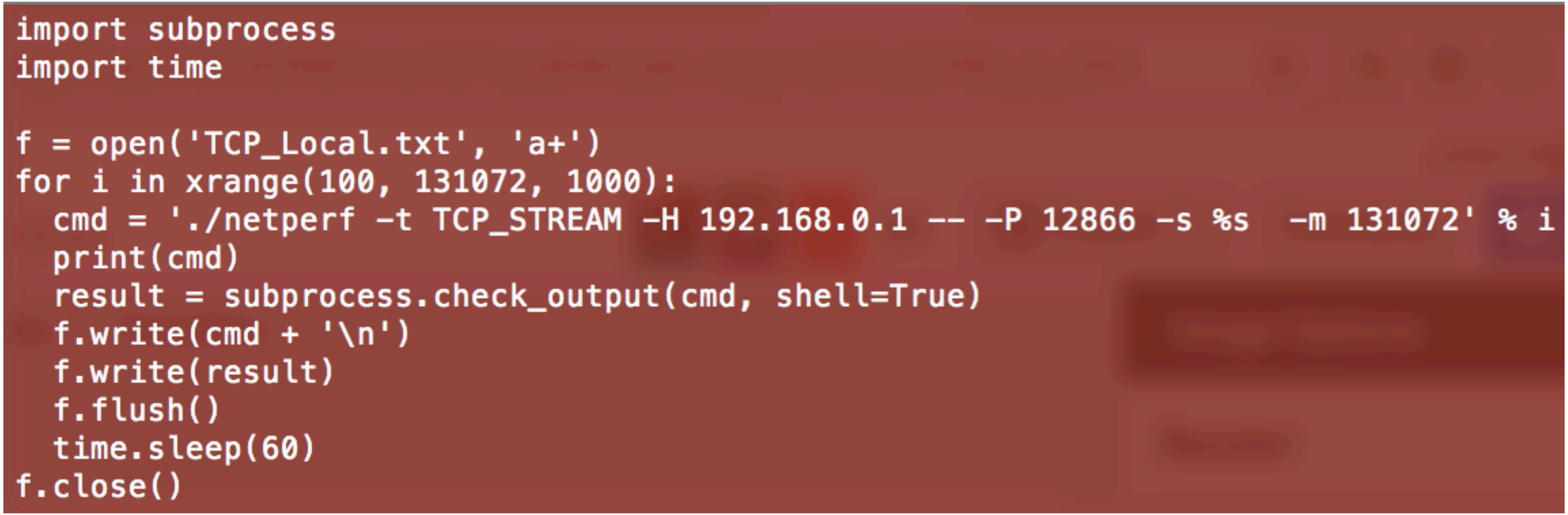
1. IPV4 Vs. IPV6 (Internet Protocol Version) : IPV4
2. Port Selection : 12866
3. Message Size to transfer : 131072 bytes

Manipulated Variables:

TCP Socket Size (Start with 100 up to 20000 bytes)

Netperf Experiment: TCP_Stream Test

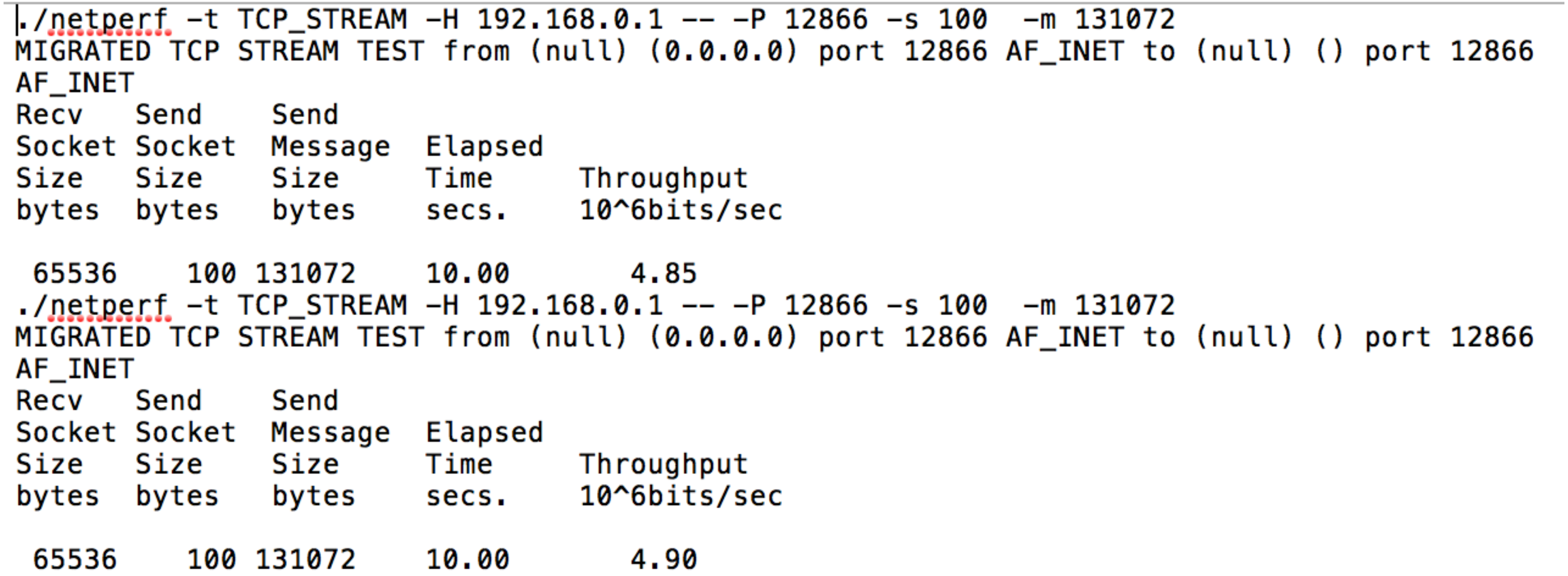
We chose python as our script language to run this experiment.



```
import subprocess
import time

f = open('TCP_Local.txt', 'a+')
for i in xrange(100, 131072, 1000):
  cmd = './netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s %s  -m 131072' % i
  print(cmd)
  result = subprocess.check_output(cmd, shell=True)
  f.write(cmd + '\n')
  f.write(result)
  f.flush()
  time.sleep(60)
f.close()
```

Netperf Experiment: TCP_Stream Test

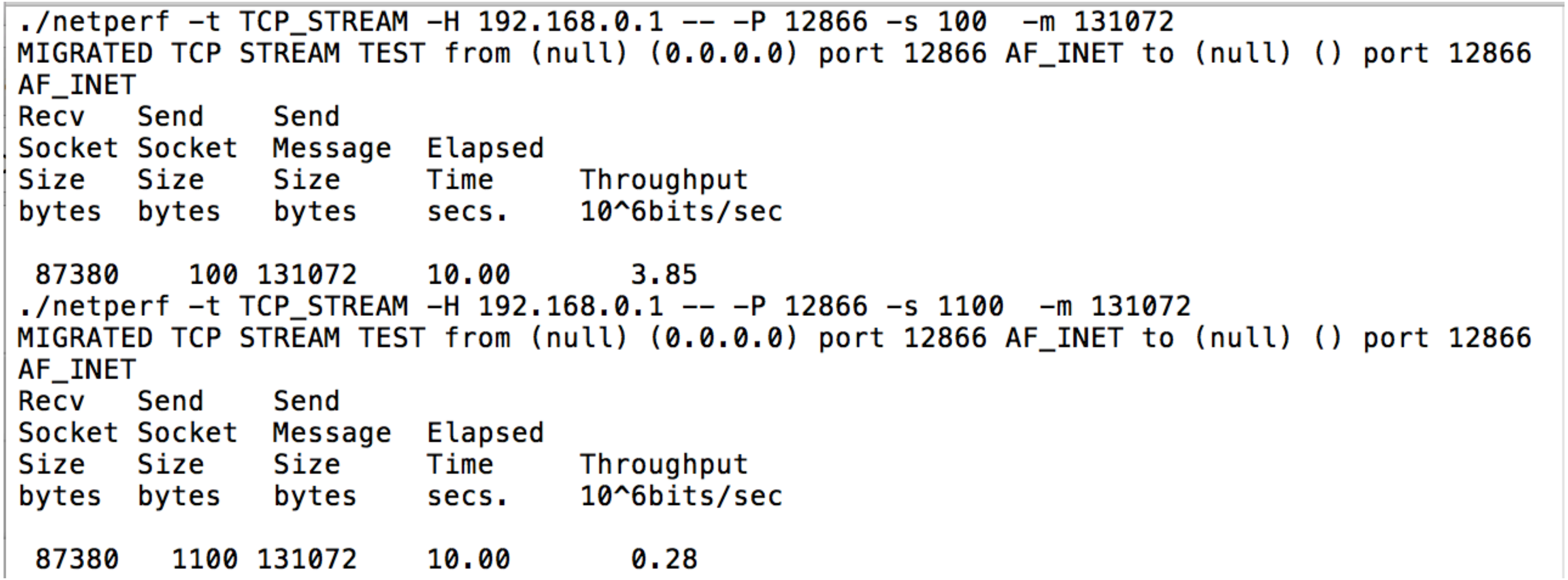
OSV Test Result:



```
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 65536    100 131072    10.00       4.85
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 65536    100 131072    10.00       4.90
```

Netperf Experiment: TCP_Stream Test

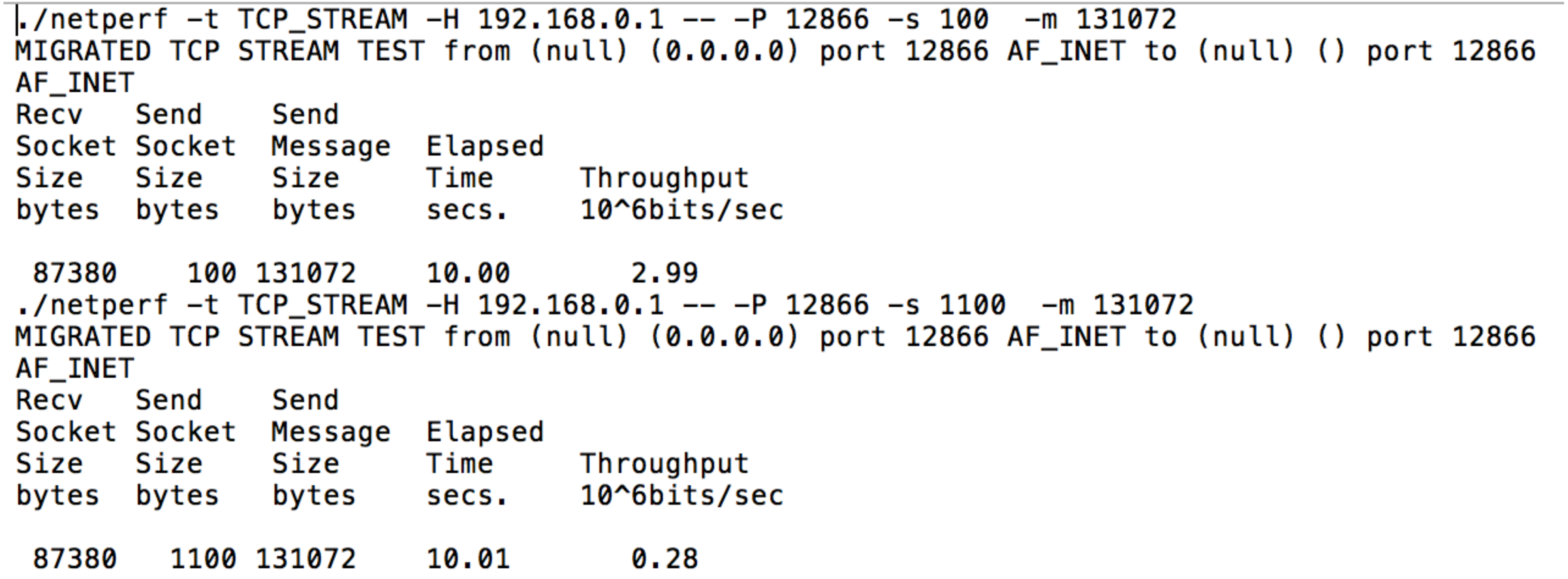
Docker Test Result



```
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380    100 131072    10.00       3.85
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 1100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380   1100 131072    10.00       0.28
```

Netperf Experiment: TCP_Stream Test

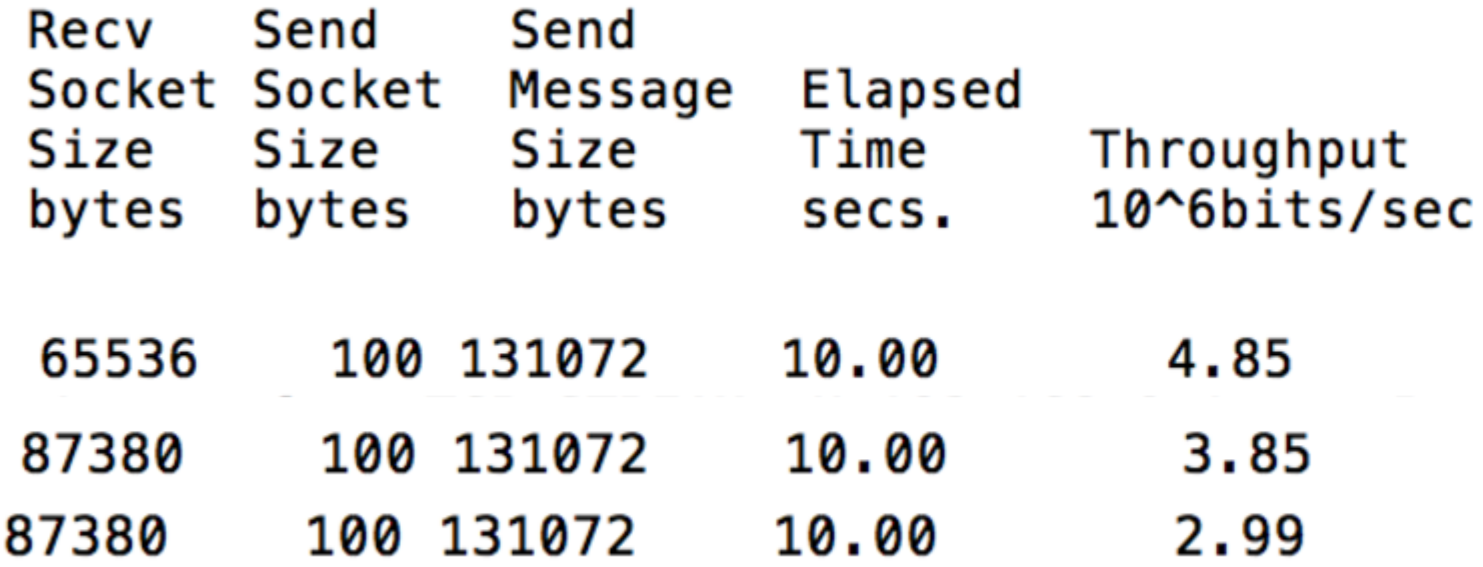
KVM Test Result



```
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380    100 131072    10.00       2.99
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 1100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380   1100 131072    10.01       0.28
```

Netperf Experiment: TCP_Stream Test

<u>WE Find THE DIFFERENCE !!!</u>



```
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 65536    100  131072    10.00       4.85
87380     100  131072    10.00        3.85
87380     100  131072    10.00       2.99
```

Netperf Experiment: TCP_Stream Test

What we really care is the throughput speed because it changes as the socket size changes

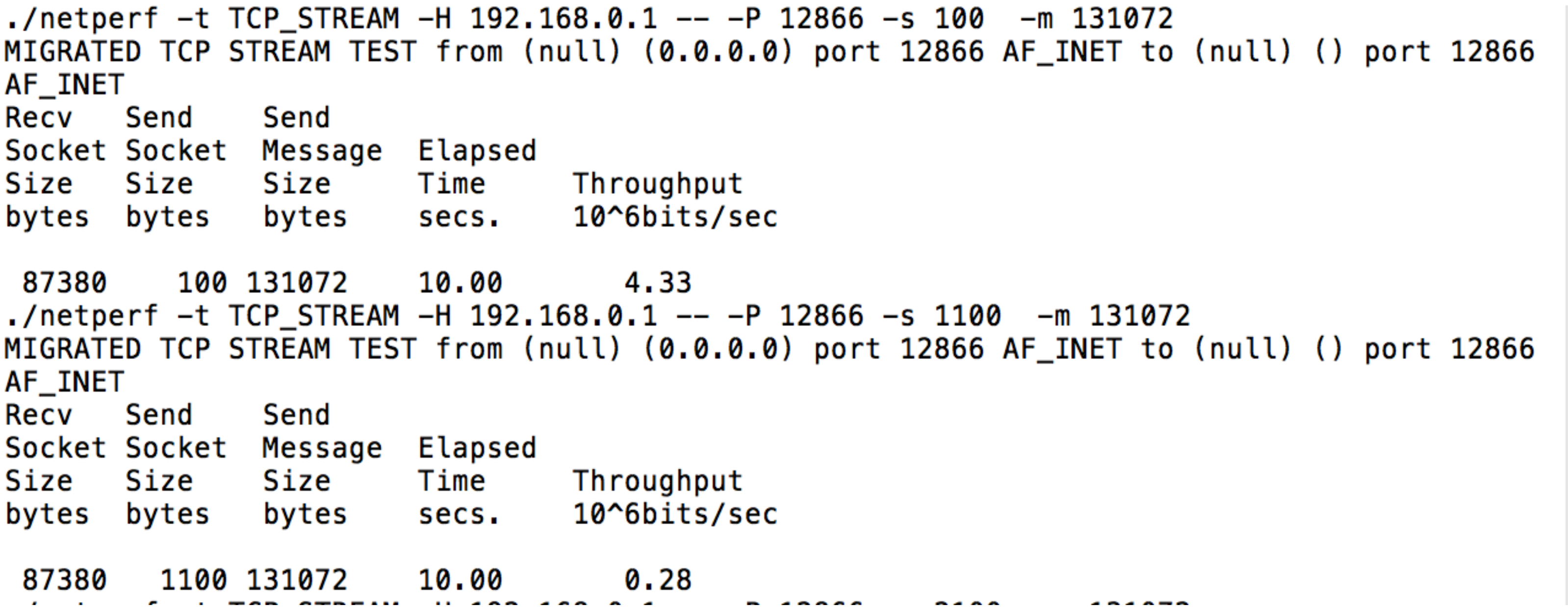
In order to prevent our experiment result is accidental, we runned second experiment with socket size starting from 300 to 20000 bytes with increase of 1000 each time.

AND WE FIND THE SAME THING !!!!

But How Can We Compare the Performance ?

We use local OS as our benchmark and compare the ratio of specific environmental throughput to local OS throughput.

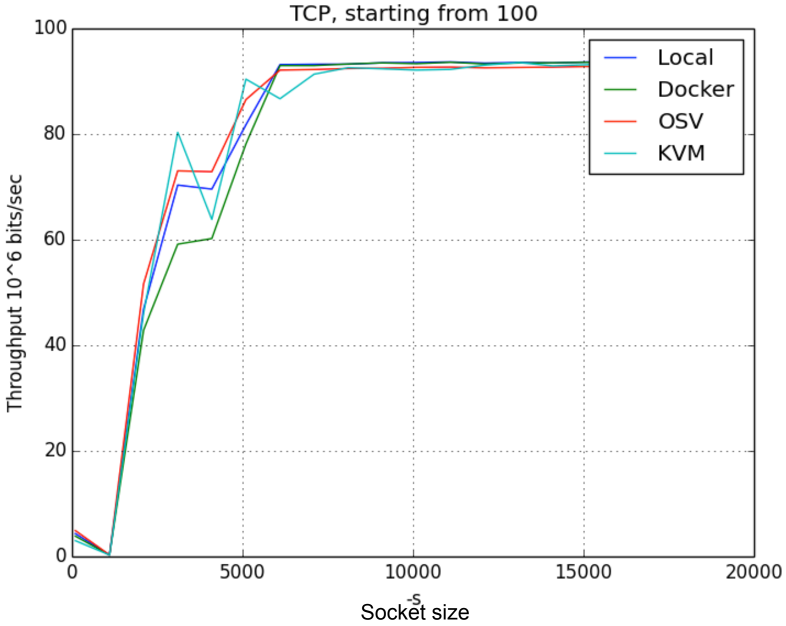
Local OS



```
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380    100 131072    10.00       4.33
./netperf -t TCP_STREAM -H 192.168.0.1 -- -P 12866 -s 1100  -m 131072
MIGRATED TCP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380   1100 131072    10.00       0.28
```

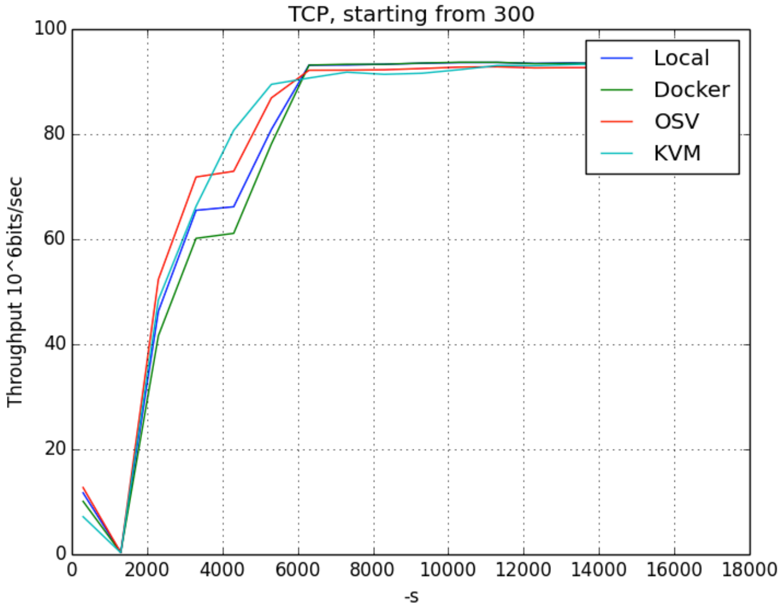
Result

Netperf result: socket size starting from 100 bytes, with increment of 1000 bytes



Result

Netperf result: socket size starting from 300 bytes, with increment of 1000bytes



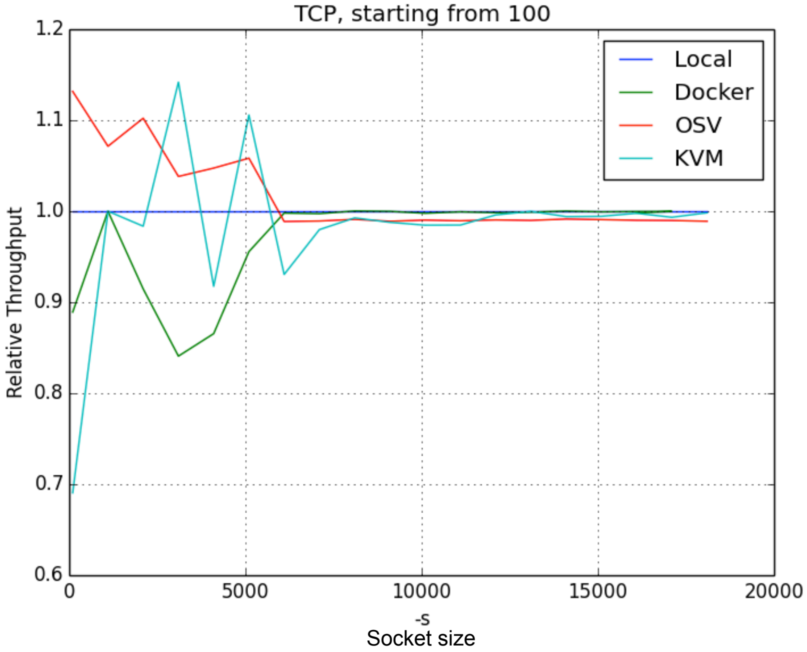
Socket size

Result

Netperf result: combined result of TCP starting from 100 and 300 bytes

Normalized with respect to

local running result

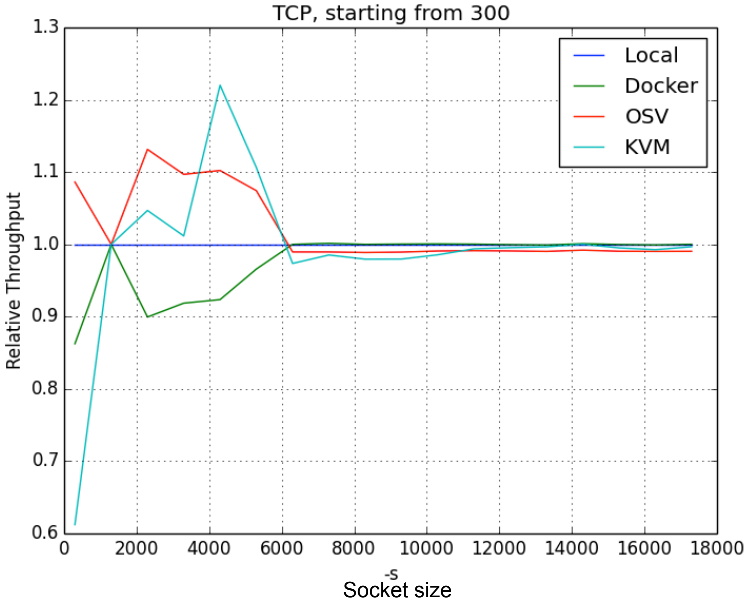


Result

Netperf result: socket size starting from 300 bytes, with increment of 1000bytes

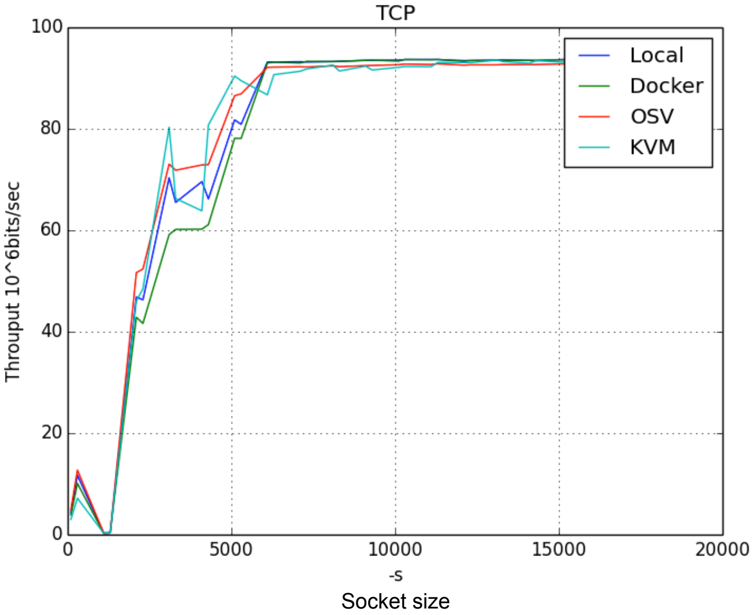
Normalized with respect to

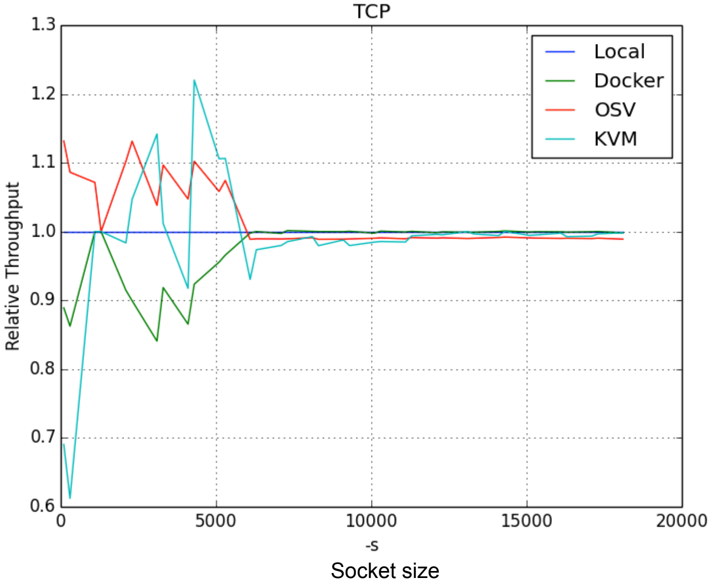
local running result



Result

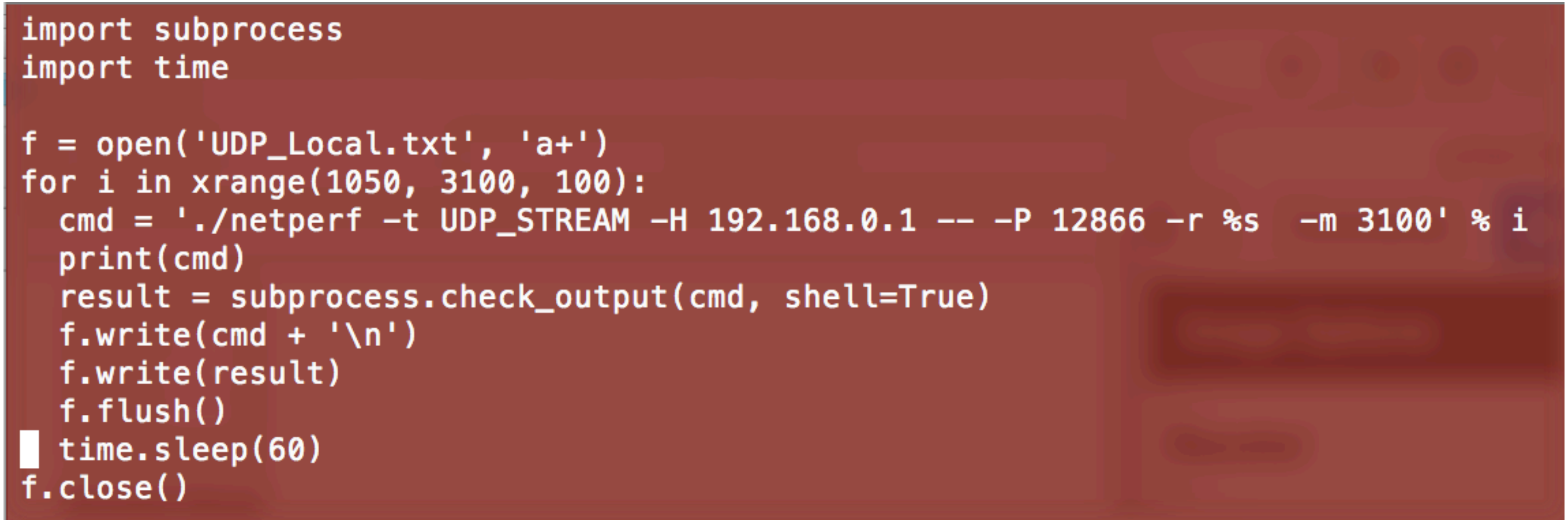
Combination of result TCP running test





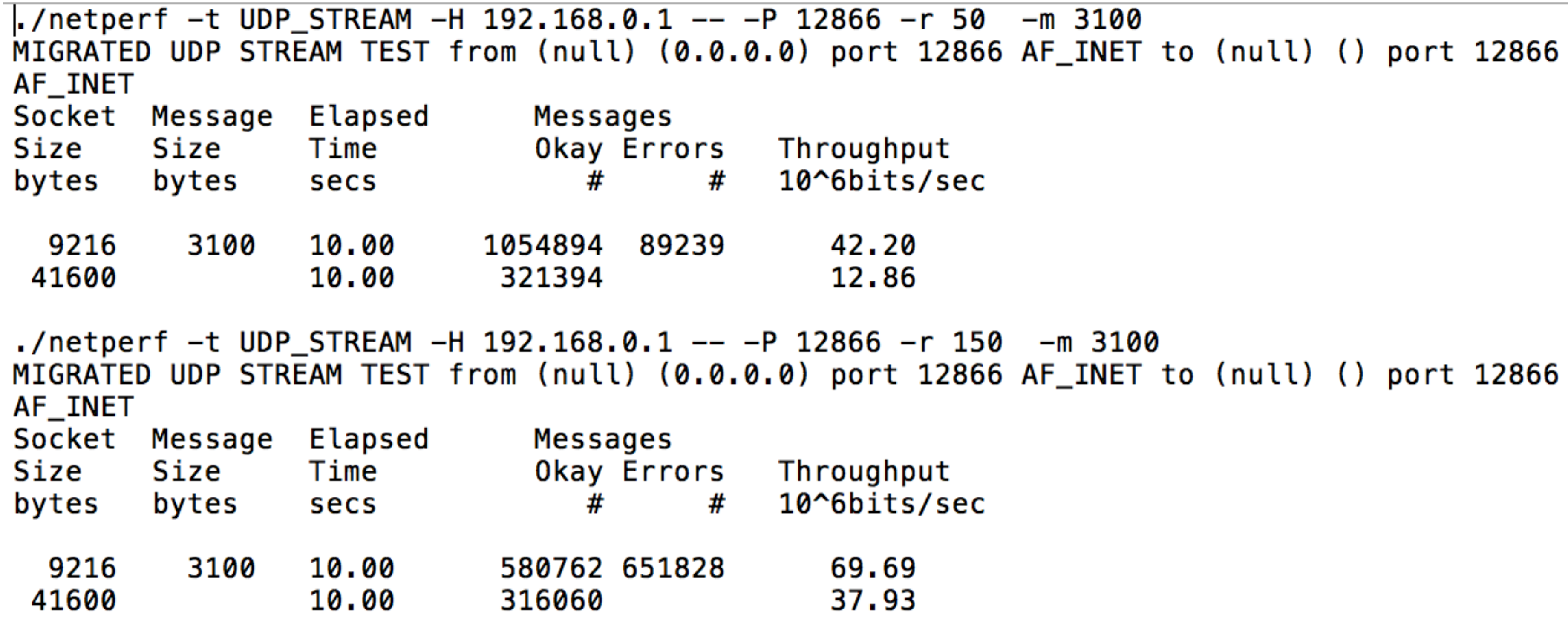
Netperf Experiment: UDP_Stream Test

Similar to the TCP_Stream, we select -r requested size in our test as our manipulated variable, and python script to run this experiment.



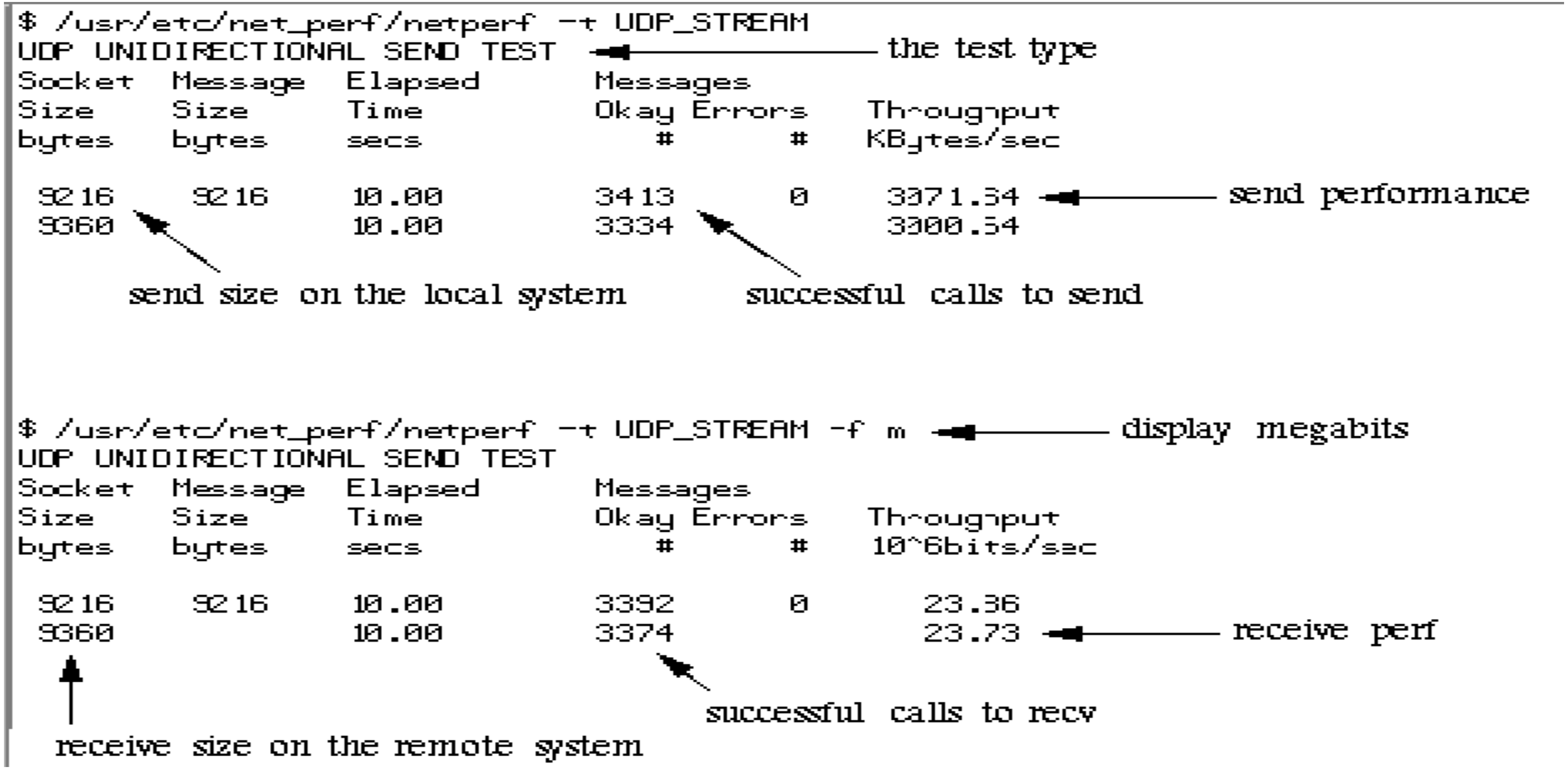
```
import subprocess
import time

f = open('UDP_Local.txt', 'a+')
for i in xrange(1050, 3100, 100):
  cmd = './netperf -t UDP_STREAM -H 192.168.0.1 -- -P 12866 -r %s  -m 3100' % i
  print(cmd)
  result = subprocess.check_output(cmd, shell=True)
  f.write(cmd + '\n')
  f.write(result)
  f.flush()
  time.sleep(60)
f.close()
```

Netperf Experiment: UDP_Stream Test



```
./netperf -t UDP_STREAM -H 192.168.0.1 -- -P 12866 -r 50  -m 3100
MIGRATED UDP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

  9216    3100   10.00     1054894  89239      42.20
 41600           10.00      321394             12.86

./netperf -t UDP_STREAM -H 192.168.0.1 -- -P 12866 -r 150  -m 3100
MIGRATED UDP STREAM TEST from (null) (0.0.0.0) port 12866 AF_INET to (null) () port 12866
AF_INET
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

  9216    3100   10.00      580762 651828      69.69
 41600           10.00      316060             37.93
```

Netperf Experiment: UDP_Stream Test

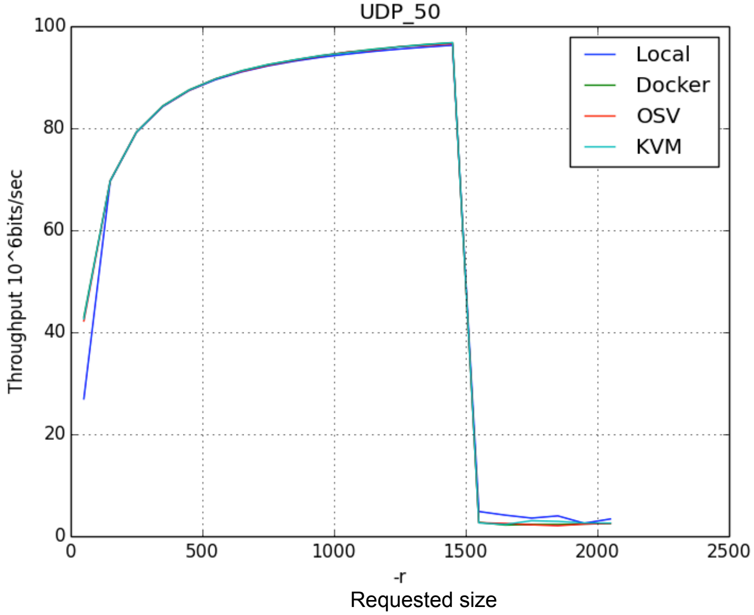


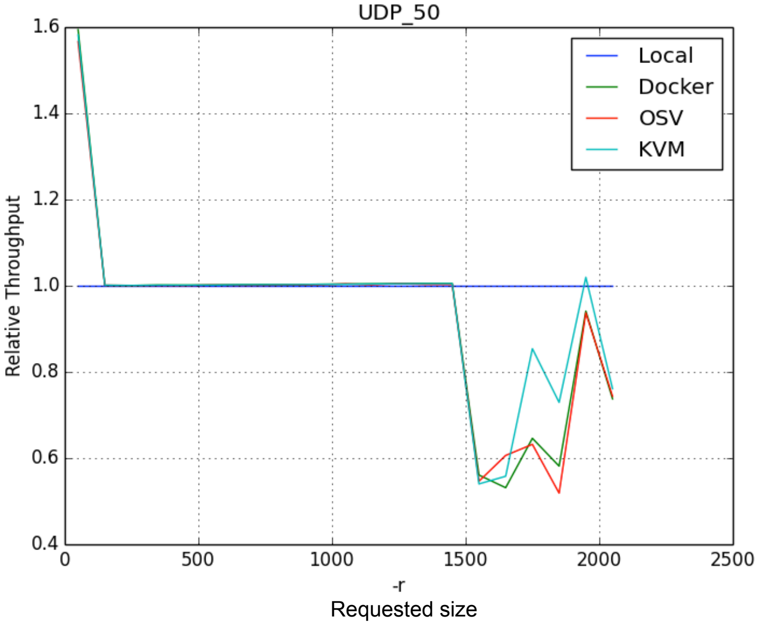
```
$ /usr/etc/net_perf/netperf -t UDP_STREAM
UDP UNIDIRECTIONAL SEND TEST
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   KBytes/sec

 9216    9216     10.00       3413      0    3371.54
 9360             10.00       3334           3300.54
```

```
$ /usr/etc/net_perf/netperf -t UDP_STREAM -f m
UDP UNIDIRECTIONAL SEND TEST
Socket  Message  Elapsed      Messages
Size    Size     Time         Okay Errors   Throughput
bytes   bytes    secs            #      #   10^6bits/sec

 9216    9216     10.00       3392      0      23.36
 9360             10.00       3374             23.73
```

Netperf Experiment: UDP_Stream Test Result

Netperf result: requested size starting from 50 bytes, with increment of 100 bytes



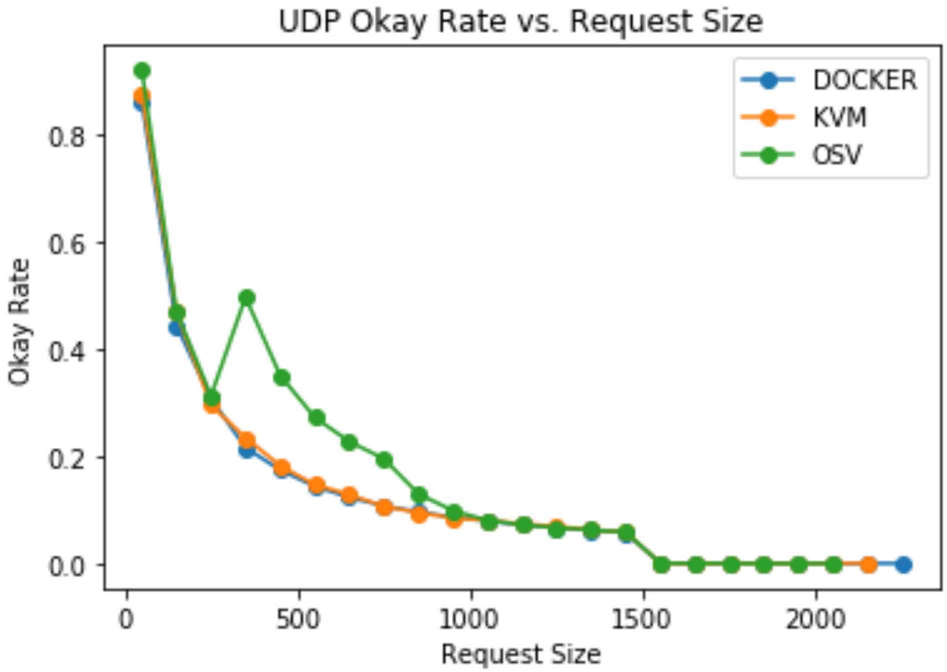


Netperf Experiment: UDP Error Comparison

It seems like the OSV has better

Okay rate in range of 250 to 900.

Okay rate = # okay / #(error + okay)



Conclusion

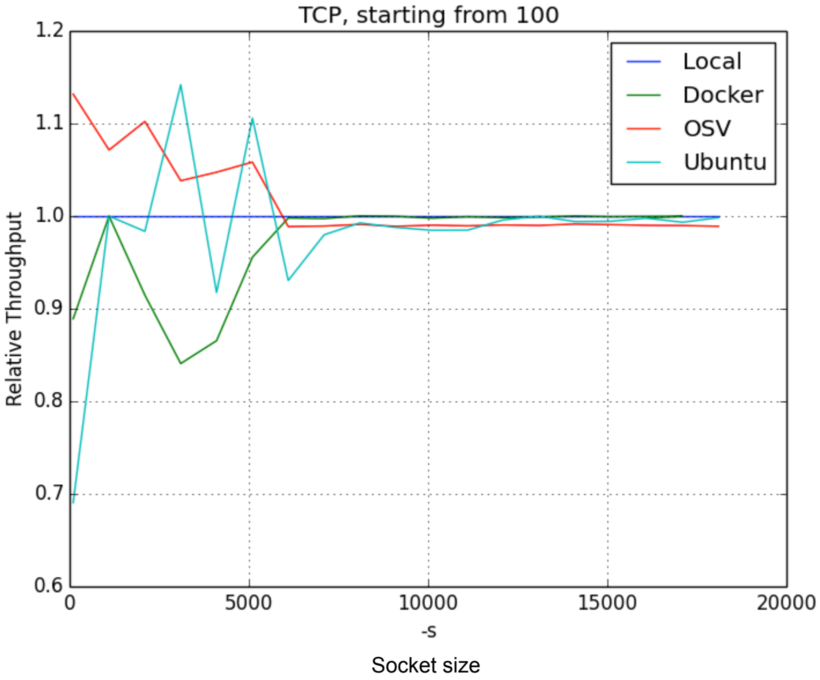
We calculate the performance based on

The area that curve covered. The area

Above Local OS is considered as positive, an

Otherwise, negative. The performance is

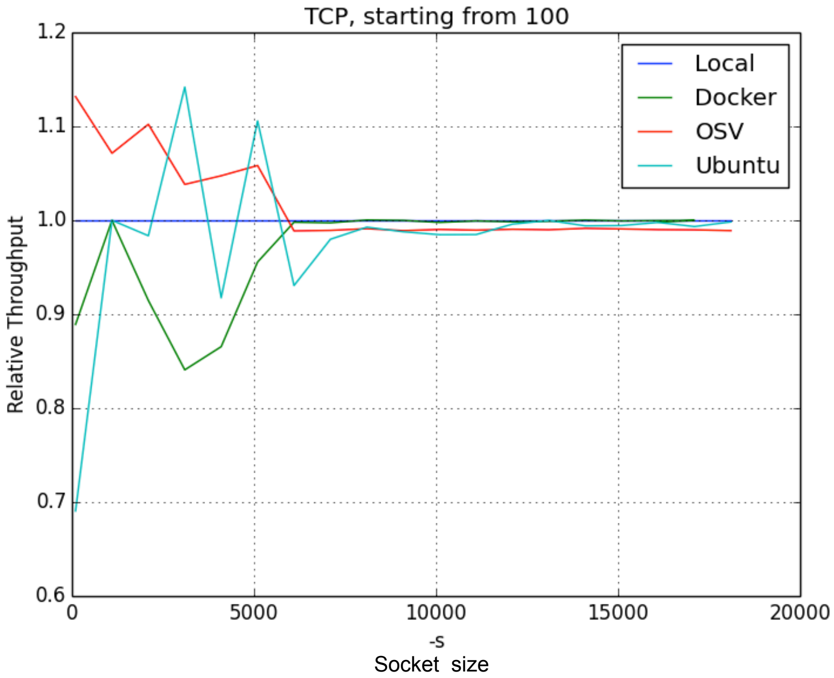
Determined by the total area.



Conclusion

Thus under socket size restriction (0 - 5000), the TCP stream performance ranking is OSV, Ubuntu(KVM), and Docker

That unikernel outperformed others in This case.

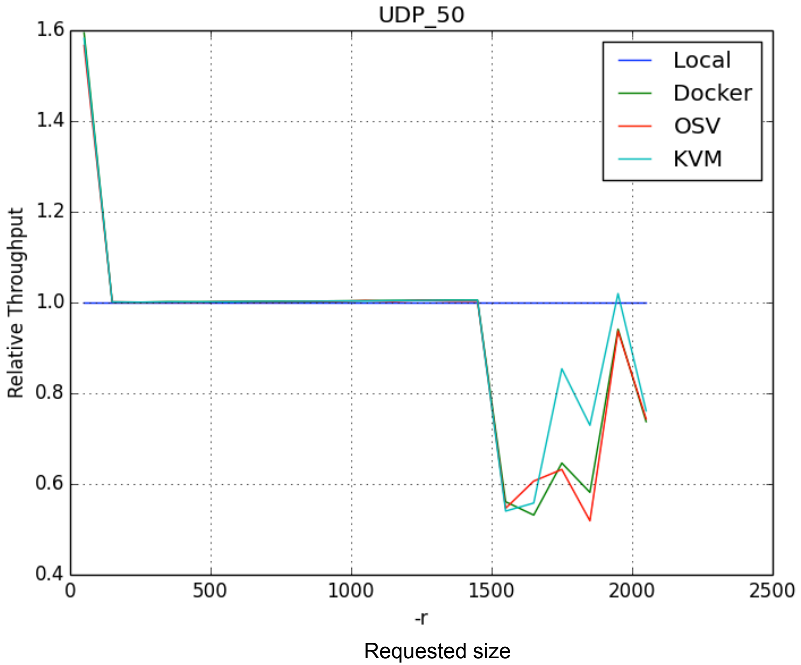


Conclusion

Under request size restriction(1500 to

2000), KVM outperforms others in term

of UDP stream performance.



Contribution and Summary

We studied the three virtualization methods and did quantitatively analysis of the network performance under OSv, KVM, and Docker environment;

Quantitatively confirm the outperformance of OSv as claimed only qualitatively in papers.

Future work

In the data collection process, we incremented socket size with increment of 1000 bytes, and the result of network performance in some environment experience bumping results.

This could be reduced by collecting data point with smaller increments, which will take longer time to finish. If more time allowed, the research could be rerun with more socket size point, thus will have more accurate results.

Questions ?