An Evaluation of Software-Based Traffic Generators using Docker

Sai Man Wong

KTH Royal Institute of Technology
School of Computer Science and Communication

smwong[a]kth.se

March 12, 2018

Why...

- ... software-based traffic generators?
- ... this thesis work?

Overview

Software-Based Traffic Generator

Hardware-Based Traffic Generator

- Expensive
- Rigid
- Accurate

Software-Based Traffic Generator

- Cheap
- Flexible
- Inaccurate

Types/Metrics

(*) User Space

Default network stack,
e.g., Iperf, Mausezahn, Ostinato and Tcpdump

Kernel Space

Loadable Kernel Module,
e.g., Brute and Pktgen

External Framework

Circumvent the default network stack,
e.g., MoonGen and TRex

Types/Metrics contd.

Table: Summary of Traffic Generators Types

Replay Engines	Replay network traffic back to specified NIC from a file which contains prerecorded traffic, usually a pcap-file.
(*) Maximum Throughput Generators	Generate maximum of network traffic with the purpose to test overall network performance, for example, over a link.
Model-Based Generators	Generate network traffic based on stochastic models.
High-Level and Auto-Configurable Generators	Generate traffic from realistic network models and change the parameters accordingly.
Special Scenario Generators	Generate network traffic with a specific characteristic, for example, video streaming traffic.
Application-level Traffic generators	Generate network traffic of network applications, for example, the traffic behavior between servers and clients.
Flow-Level Traffic generators	Generate packets in a particular order that resembles a particular characteristic from source to destination, for example, Internet traffic.
(*) Packet-Level Traffic Generators	Generate and craft packets, usually, from layer 2 and up to 7.

Virtualization

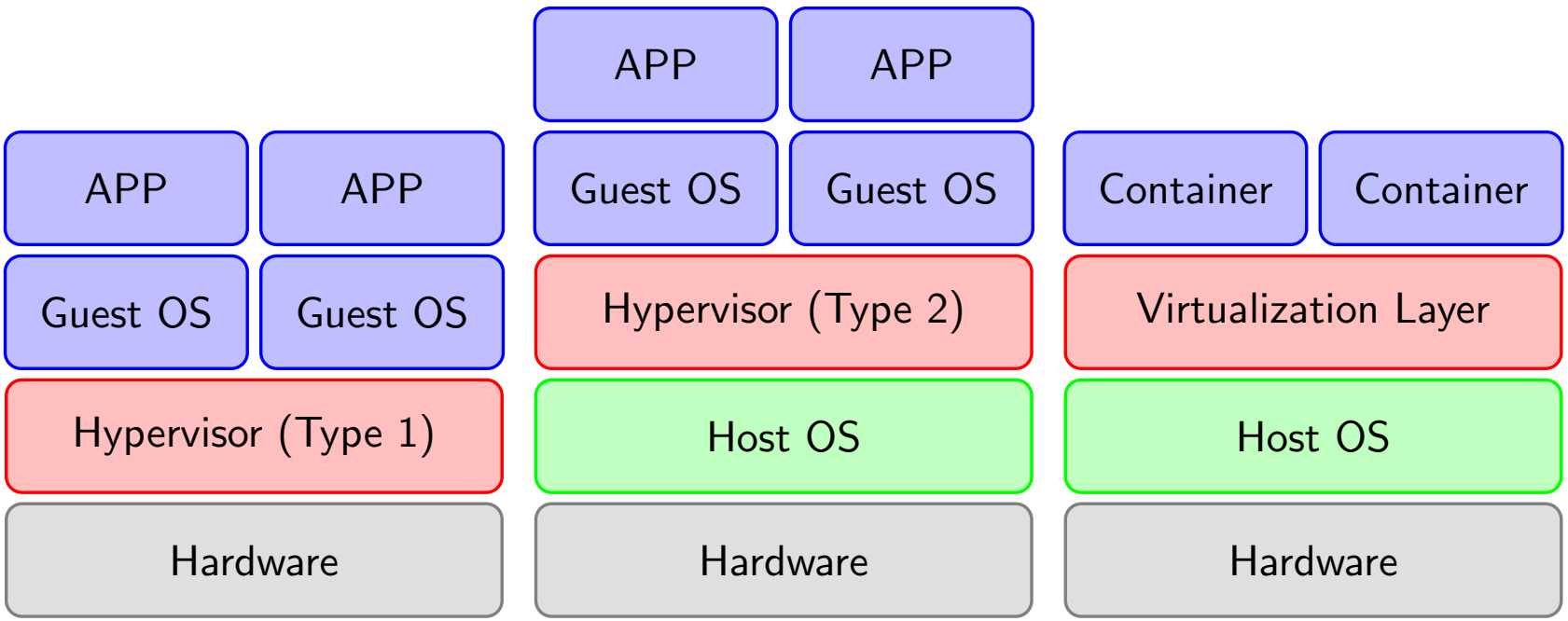


Figure: Hypervisor-Based (Type 1 and 2) and Container-Based Virtualization

Docker Architecture

- cgroups "limits how much you can use"
- namespaces "limits what you can see (and therefore use)"

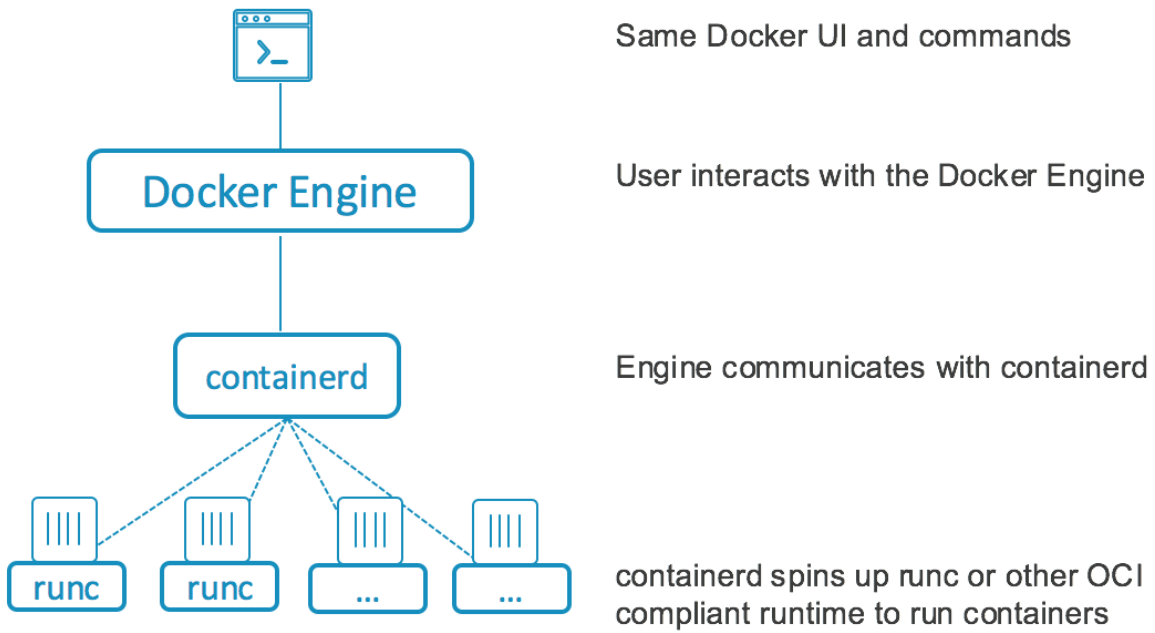


Figure: Overview of Docker Architecture

Experiments

Tools

- Iperf
  (infinite packets)
- Mausezahn
  (infinite packets)
- Ostinato
  (infinite burst packets)
- Tcpdump & Capinfos
  (capture and analysis)

Data Collection

- Source to sink
- UDP traffic
  64 – 4096 bytes packet size
- 10 seconds x 100 iterations

Physical Environment

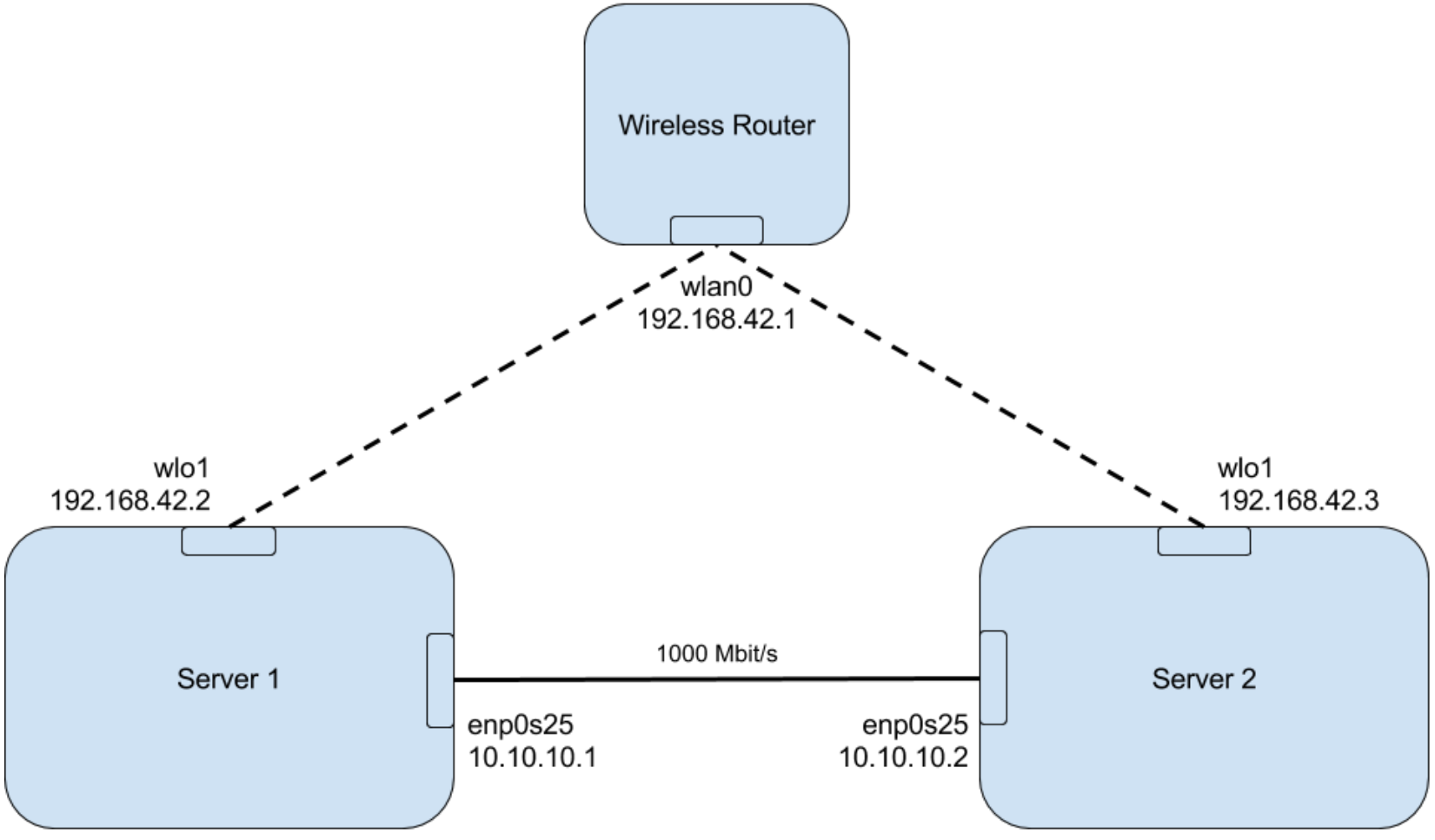


Figure: Overview of Physical Lab

Physical Environment: Results

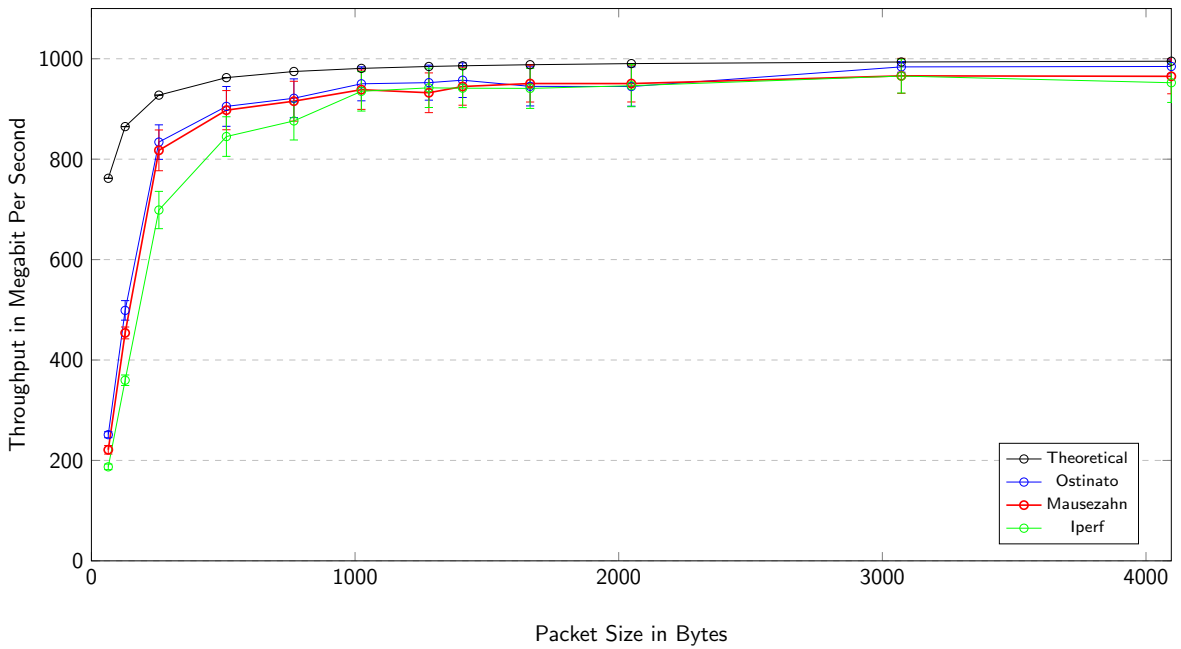


Figure: Throughput Graph Summary in Physical Environment

- Ostinato 984.61 Mbps @ 4096
- Iperf & Mausezahn 965 Mbps @ 3072
- 12% and 75% below limit @ 64 – 512

Virtual Environment

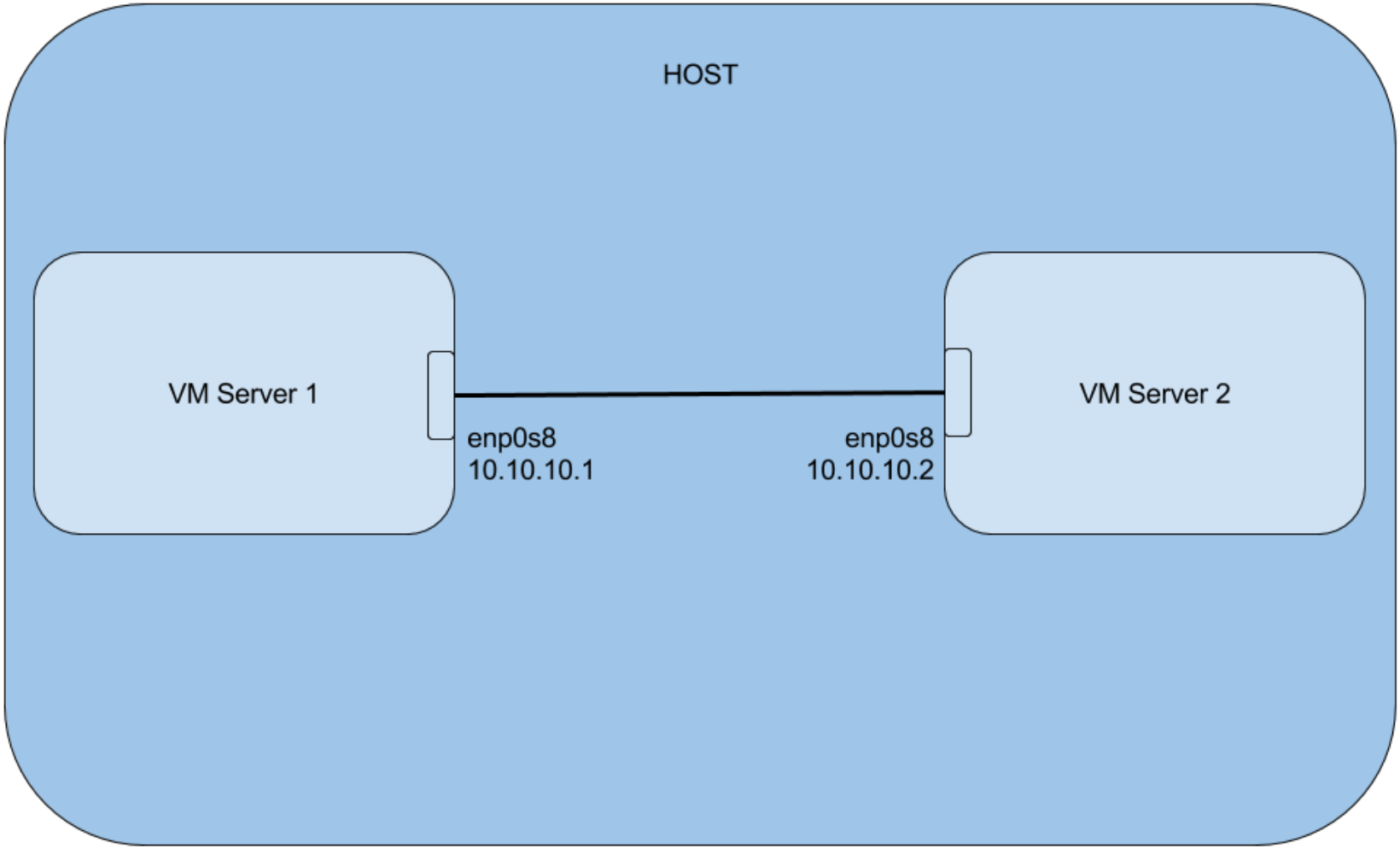
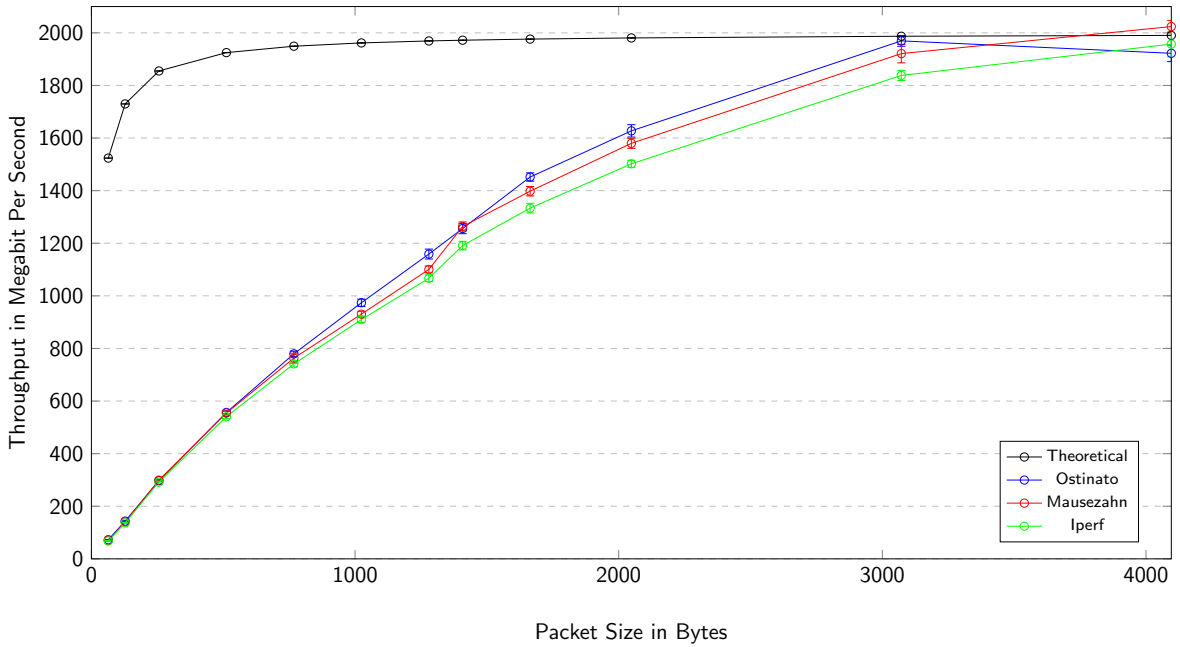


Figure: Overview of Virtual Lab

Virtual Environment: Results



1. Around 2000 Mbps
2. "internal-networking mode"
3. CPU depended

Figure: Throughput Graph Summary in Virtual Environment

Recommendation & Conclusion

Recommendation

1. Craft arbitrary packets: Mausezahn and Ostinato
2. Automation: Ostinato
3. Bandwidth test: Iperf

Conclusion

1. Smaller end-to-end system tests
2. Suitable with container technology for automation capabilities and reproducibility

Run a container (CLI)

Example (test.py)

```
import time
while True:
    print("hello there")
    time.sleep(3)
```

```
$ docker run --rm -it
    --name demo1
    -v $PWD/test.py:/test.py
    python:latest python test.py
```

Run a container (Dockerfile)

Example (Dockerfile)

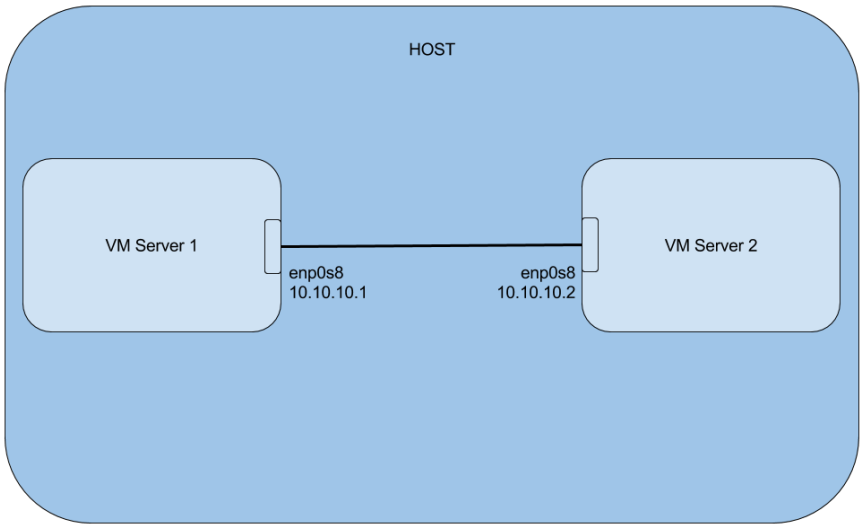
```
FROM python

COPY test.py test.py

ENTRYPOINT ["python", "test.py"]
```

```
$ docker build -t demo2:1.0 .
$ docker run --rm -it demo2:1.0
```

Experiment in virtual environment




```
$ ./send_receive_packets.sh
    vm
    ostinato
    1
    64
    10
    ./tmp
```

Figure: Overview of Virtual Lab

Thank you & Questions