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

Yinghe Chen, Xi Wang, Miaoyan Zhang, Xiaoying Yang, Minjie Zhang, Abhishek Gupta
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, California, USA
ychen15@scu.edu, xwang10@scu.edu, mzhang1@scu.edu,xyang@scu.edu, mzhang2@scu.edu

Abstract —— **Virtualization is the central part of cloud computing which enables industry or academic IT resources through on-demand allocation dynamically. Hypervisor and container are the two main virtualization techniques that applied in cloud computing. Both have its own merits and demerits. Recently, there is a technology on the rise which is far more promising where security is needed: the Unikernel. Similar to Linux containers, it is a lightweight alternative to deploying distributed application based on microservices. However, the comparison between hypervisor, container and Unikernel, regarding the concurrent provisioning of instances, as in the case of microservices-based application is under controversial. In this research, we quantitatively evaluate network performance overheads for OSv Unikernel, hypervisor-based virtualization using Kernel-based Virtual Machine (KVM), and the container-based virtualization using Docker comparing the different configuration and optimization. In this study, we confirmed that Unikernel outperformed KVM and docker in term of network (TCP/UDP) performance especially when the transmission size is restricted.**

Keywords — **Cloud Computing, Docker, KVM, Network Performance, TCP, Unikernel**

I. INTRODUCTION

Cloud Computing has been envisioned as the next-generation architecture of IT enterprise, due to its long list of unprecedented advantages in the IT history: on-demand self-service, ubiquitous network access, location independent resource pooling, rapid resource elasticity, usage-based pricing and transference of risk. Virtualization is the main technology that powers today's cloud computing systems. It is widely used in public clouds and private clouds. Virtualization provides isolation and resource control which enables multiple workloads to run efficiently on a single share machine and thus allows servers that traditionally require multiple physical machines to be consolidated to a single cost effective physical machine using virtual machines or containers [3].

The two main virtualization techniques which provide multitenancy on a single host machine are hypervisor-based virtualization and container-based virtualization. Hypervisor-based virtualization allows multiple operation systems to run on a single host machine by creating isolated virtual machines (VMs), and that each have a set of virtualized CPUs, memory, and I/O devices such as networking and storage devices [7]. While the Container-based virtualization allows multiple independent userspaces on a single multitenancy through using operating system kernel namespaces and control groups. In linux system, containers provide multitenancy by utilizing operation system kernel namespaces and control groups. Docker is a high level tool which simplifies creation and management of Linux containers but still use the same namespace and cgroup mechanisms under the hood [7].

Although virtualization technique has its own merits, the performance overhead of virtualization compared to bare metal is still under controversial [1]. Both hypervisors and containers which are the two of the most popular virtualization techniques have performance overhead in areas of CPU, memory management and I/O which all contribute to the performance of network intensive applications under virtualization. While networking performance is particularly critical for cloud computing. Cloud computing workload like Memcached depend strongly on TCP/IP performance.

In this study, we evaluated the network performance of KVM, Docker and Unikernel. KVM, also known as Kernel Virtual Machine, is a full virtualization solution for Linux on x86 hardware containing virtualization extensions. In KVM, every virtual machine is treated as a Linux process. Another technique that applied in the virtualization of cloud computing is docker. Docker is an open-source project that provides a systematic way to automates the deployment of application inside portable containers (Figure 1-a). We also employ a new kind of virtualization technique: Unikernel which is single address space kernel constructed by operating system library which might be a lightweight rep[1]lacement for the work.It is built into specialized machine image that can run directly on hardware or hypervisor (Figure 1-c). Unikernel is claimed to reduce the amount of code deployed with better security; its images are orders of magnitude smaller than traditional deployment; also, the Unikernel compilation model is highly optimized. Existing Unikernels are implemented in mainly two ways: one uses POSIX API, like OSv [2], which may be defected by the old architecture but supports existing applications; the other uses completely new API seeking better performance in not so widely used language, like MirageOS [3] written in OCmal, Clive [4] written in GO, etc. One previous study

[1] GitHub Link: https://github.com/TeamBroncos/BroncoLovesCloud.git

shows that Unikernel has higher network throughput than Linux network[3] however the another study indicated the Unikernel does not outperform the other virtualization technique in term of network performance[7].

Unlike the previous researches, in which they pick the transmission size arbitrary, in this project, we try to quantitatively evaluate the network performance of Unikernel on KVM, Docker, Linux on KVM (Figure 1-c) and Linux native using our personal computers and local area network to simulate the cloud. We want to see whether Unikernel actually outperforms container and VM in terms of network performance.

In the project, we choose the OSv as the target Unikernel because OSv is open source, uses POSIX API, and supports a wide range of hypervisors and programming languages including C, Java, Ruby, Node.js, and Scala.

Our proposed implementation tests are:

- TCP stream quantitative test
- UDP stream quantitative test

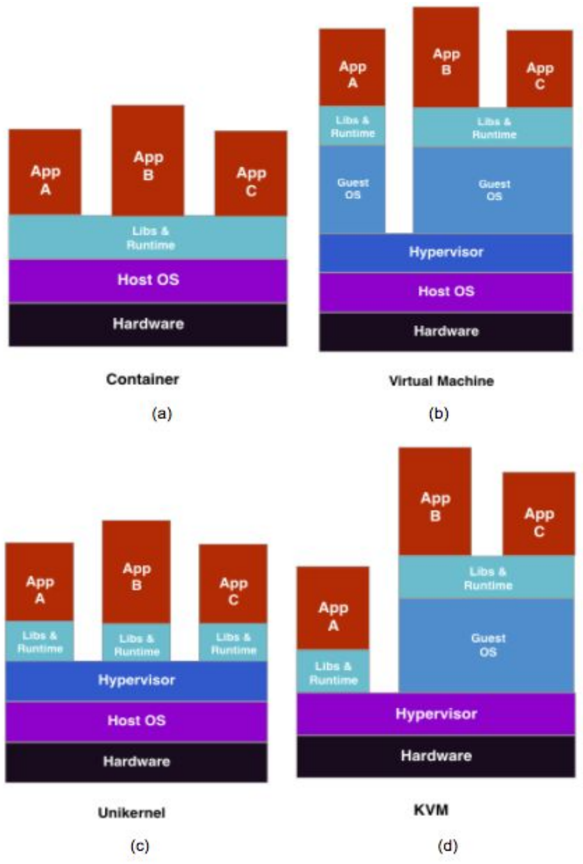

Figure 1. The architecture of a) docker container, b) virtual machine, c) unikernel, and d) KVM

The implementations will be conducted over our personal computers and local area network to evaluate TCP and UDP performance using Netperf. One computer acts as the server, another computer acts as the client, and the two computers are connected using ethernet directly, which can be shown in Figure 2.

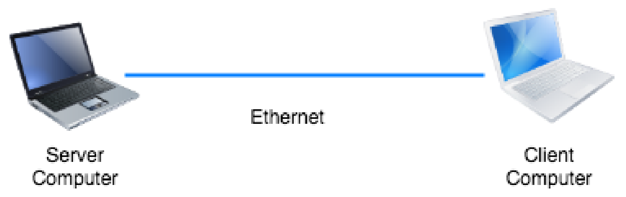

Figure 2. Evaluation setup. The system under test (server) is connected back-to-back to the load generating client to eliminate overhead from a network switch. There are no other processes running on the system and the goal is to obtain lowest possible latency.

II. PROBLEM ANALYSIS AND PROPOSAL SOLUTION

Before As our introduction described, these virtualization techniques (Linux/KVM, Unikernel, and Docker) have contributed to cloud computing. Previous study [1] examined the time provisioning evaluation toward these three technologies. However, the performance result does not show as the author proposed. The experiment result shows that Docker beats Linux/KVM and Unikernel in term of OpenStack tack communication overhead [1], and Unikernel (OSv) has the same performance as Linux/KVM in term of OpenStack full workload times. Morabito's study [6] stated there are 28% and 26% decrease in TCP stream test for Linux and OSv respectively, where another researcher [7] does not see such decrease toward TCP stream. Our goal is to fully quantitatively evaluate these three techniques toward TCP performance through AWS or remote server based on proportional access traffic, and we propose that the Unikernel technique will outperform the Container and Linux/KVM technologies.

III. IMPLEMENTATION AND EVALUATION

We use a remote computer as our server because we found it is very unlikely to use a public cloud like AWS as our cloud. Even though We have searched and found out the AWS is able to support Unikernel technology, and one easy implementation is shown by Galois [9], and this Open source tool is available to provide mechanism for uploading Unikernels to run on EC2, still, we have still faced challenges that this tool is still in alpha quality so that feasibility of running the program is not done and not clear, and this may influence our experiment result. Another challenge is that this tool requires installing Linux operating system, but the free ASW instance gives one 1GB memory and only 1 vCPU, and this will influence our experiments to result significantly. Our alternative proposed solution is to use a personal computer as a remote server which does not limit our memory.

A. Experiment Approach

We will use Netperf [8] as our evaluation tool to measure TCP stream in order to compare the network performance among three virtualization techniques as Figure 1 because Netperf shows lots of vital informations we need to compare in term of performance including the receive socket size, send socket size, send message size, and throughput [7]. We propose to conduct the TCP stream tests and UDP stream test, similar to Enberg's study [7], the TCP stream test measures the efficiency of the operating system TCP receive path [8].

The TCP test and UDP test measure the efficiency of operating system TCP transmission under different environments. we have selected OSv as instance of unikernel, Docker as instance of Container, and Linux as instance of KVM shown as Table 1.

Based on the Netperf measurement result, we will plot throughput against traffic proportion for at least 10 iterations. Different virtualization techniques (1, 2, 3) from Table 1 will be marked as Blue, Red, and Green respectively as the proposed data in Figure 1(Data are artificial). The final traffic will be at least 20 times bigger than original traffic depends on performance. Since the traffic will be increased proportionally and quantitatively, our plot will help us understand these virtualization techniques toward network performance, especially in TCP .

#	Virtualization Technique	Guest OS	Tool
1	Hypervisor	Linux	KVM
2	Hypervisor + Unikernel	OSv	KVM
3	Container	Linux	Docker

Table 1. Three Virtualization techniques

B. Environment Setup

In this section, we detail our environment setup for evaluating network performances on OSv, Linux/KVM, and Docker on the same server computer. To have a comparison, the network performances are evaluated on Linux Host OS as well. Server Computer hardware information is shown in table 2.

CPU	Intel(R) Pentium(R) CPU N3530 @ 2.16GHZ
Memory	4GB

Table 2. Server Computer Hardware

a) Host OS on server computer:

The host OS is Ubuntu 14.04.5 LTS. To have a comparison, Netperf is installed on the host OS as well. Then the test results on OSv/Docker/KVM can be compared with those on the host OS.

b) OSv on KVM on server computer:

The OSv is 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.

c) Linux on KVM on server computer:

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.

d) Docker on on server computer:

Docker is installed on the host Linux. Netperf is installed in a docker container. In all, we have these software on our server computer, as shown in table 3And the architecture of our linux server can be shown as Figure 3.

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 Image: Ubuntu 14.04.5 LTS
Netperf	Netperf-2.7.0

Table 3. Software Configuration on Server Computer

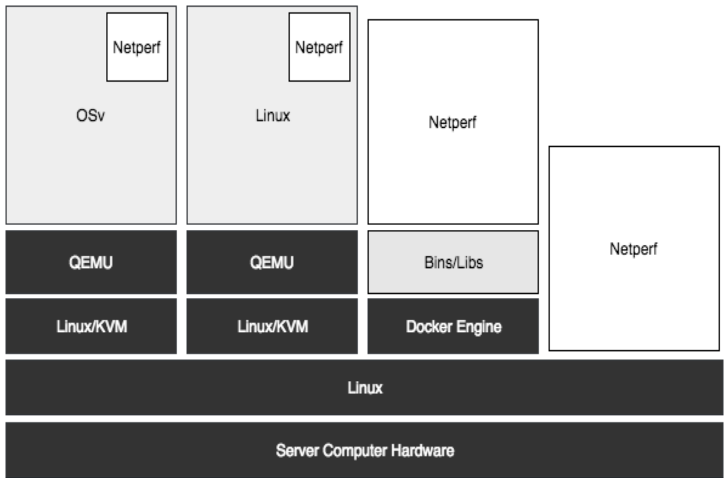

Figure 3. Architecture of the Linux Server

To map host OS physical network with hypervisor virtual network, port mapping need to be set up. Netperf server uses two ports: one control port of TCP 12865 and one TCP or UDP data port. We let ports of TCP 12865, TCP 12866 and UDP 12866 be mapped between host physical network and hypervisor virtual network. Every time we run Netperf client, the client must specify TCP 12866 or UDP 12866 as the data port. In every test, we only run one Netperf server on host OS, OSv guest, Docker container, or Linux guest on the server computer, and the Netperf client is run on the client computer. The server computer and client computer are connected directly via Ethernet cable of 100M.

C. TCP_STREAM Performance Test

The TCP_Stream test is a default test in netperf, and it is our first approach to analyze the network performance. It transfers some quantity of data from the system running netperf to the system running netserver. The vital variables in this test included the IPV(Internet Protocol Version), Port, message size to transfer, and the socket size. We set the first three variables as constant so that the netservers start with IPV4, and we ask netperf to connect with server based on port 12866, and restrict the message size to be 131072 bytes. The manipulated variable socket size (-s) starts with 100 bytes and reached up to 20000 bytes with increment of 1000 bytes per gap.

We started netserver in Docker, KVM, OSv in our server computer, and We used python to write a script to run the netperf on our client computer, which connected with server by ethernet wire, to conduct appropriate TCP_Stream test on each of the environment with 70 seconds sleeping each time to ensure that the current process will not be interfered by others before it has been finished.

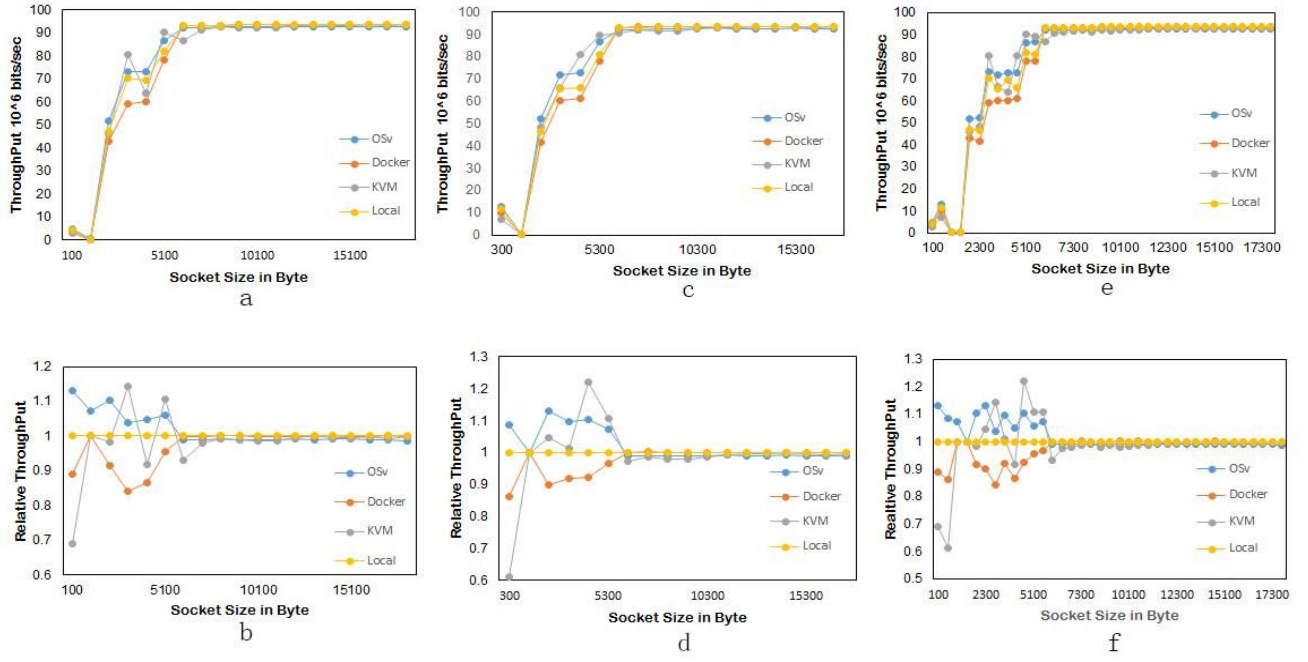

Figure 4. TCP Performance in different virtualization platform. a: Absolute throughput with socket size increasing from 100 bytes/sec with increment of 1000 bytes; b: relative throughput with socket size increasing from 100 bytes/sec with increment of 1000 bytes; c: Absolute throughput with socket size increasing from 300 bytes/sec with increment of 1000 bytes.; d: relative throughput with socket size increasing from both 300 bytes/sec with increment of 1000 bytes; e: Overall absolute throughput performance of network under all the environment setups; f : relative throughput under all the environment setups .

We conducted research at one point three times , and we don not see difference between the replicates, all throughput results are same in terms of the accuracy provided by netperf. This the error of the data are negligible ,in the quantitative results provided below.

III. BENCHMARKING TCP PERFORMANCE

We have found significant throughput differences under different operating environments as in Figure 4. As shown in Figure 4-a, the throughput experienced significant change from socket of 2000 bytes to 6000 bytes. In this period, the unikernel (OSv) have higher throughput than both local throughput and Docker result performance, the difference range from 4 to 20 * 10^6 bits/sec. While for KVM (run on Ubuntu), it experienced more various throughput change. Its performance is not gradually increasing, for example at socket size from 3000 bytes to 4000 bytes, the throughput is experiencing a decrease from 80 * 10^6 bits/sec to 62 * 10^6 bits/sec. The decrease occurs again from socket size of 5000 bytes to 6000 bytes. This shows that KVM socket sending network performance is not as stable as others in our test.

In order to study the major difference between socket size 2000 bytes to 7500 bytes, we normalized the throughput of different environments with local operating system data (The local OS data used same python script) to create Figure 4-b.

In Figure 4-b, we get a more obvious performance difference among the four operating environments. OSv network performance is better than that in local operating environment, while the later has better performance than Docker environment, at before socket size of 5000 bytes/sec. Again, network performance under KVM operating environment is experiencing significant ups and downs, not as stable as the performance under other environment. All the performance differences tend to converge to local environment performance after 5000 bytes socket size.

So far, we found the performance difference when running TCP test from socket size of 100 to 20000 bytes, with increment of 1000 bytes. To be more convinced about the trend we found, we conducted the performance experiment again, but with socket size of 300 bytes to 18000 bytes, with increment of 1000 bytes. We obtained similar results. According to the absolute throughput result in Figure 4-c and relative throughput result in Figure 4-d, the network performance of OSv is about 4 * 10 ^ 6 bits/sec better than that of Local, while the Local is also about 4 * 10 ^ 6 bits/sec better than Docker, from socket size of 2000 to 6000 bytes. In this case, KVM has a better performance than local and Docker environment in all points within the region we are interested in. However, compare to OSv, KVM has lower output right before socket size reach 4000, then is better after 4000 socket size. In the case of socket size starting from 300 bytes, OSv is still the most stable with best TCP network performance than other operating system environment.

Figure 4-e and Figure 4-f show the combination of both data points collected from socket size of 100 bytes and 300 bytes, both with increment of 1000 bytes. In Figure 5, we further compare the average TCP relative throughput in the restrict socket size (0-6000 bytes) and found that Unikernel (OSv) outperformed the other three techniques (Figure 5). The results address the better network performance of OSv than Local operating system, and better local operating system performance than that under Docker environment. The KVM again shows various odds during performance, while at some point better than all others, at some points are not as good as others, which indicates performance instability of KVM environment. The overall TCP performance of OSv is very promising, the reason of the outperformance can be because OSv, unikernel machine, runs directly on hypervisor/hardware, thus having less application space copy during the operating process.We have also done log scale and find out that the log scale is verisimilar to the raw graph.

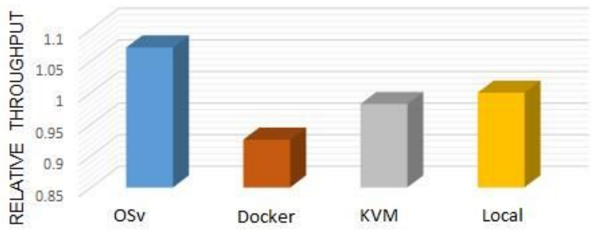

Figure 5. The average of relative throughput from socket size of 0 bytes to 6000 bytes in different condition: Unikernel (OSv), Docker, KVM and local linux.

IV. BENCHMARKING UDP PERFORMANCE

The UDP_STREAM test is also a default test in netperf, and it shows critical network performance in term of UDP. We use requested size as our manipulated variable in this test, and other control variables in this test included the IPV(Internet Protocol Version), Port, and message size to transfer. We set the first three variables as constant so that the netservers start with IPV4, and we ask netperf to connect with server based on port 12866, and restrict the message size to be 3100 bytes because the UDP message size must be restricted to relatively small. The manipulated variable requested size (-r) starts with 50 bytes and reached up to 2000 bytes with increment of 100 bytes per gap.

UDP test was conducted with request size starting from 50 bytes with increment of 100 bytes. Figure 6-a shows the absolute throughout result with increase of socket size. The absolute throughput of all four operating environment showed very similar pattern: all with gradual increase followed by a sharp drop from 97 *10 ^ 6 bytes/sec to 10 * 10 ^ 6 bytes/sec at request size of 1500 bytes. We again did a ratio normalization of the throughput with respect to local performance. For UDP performance test, in Figure 6-b, KVM shows a better performance than OSv and Docker, while none of the three environment setup has as good performance as local operating system environment.

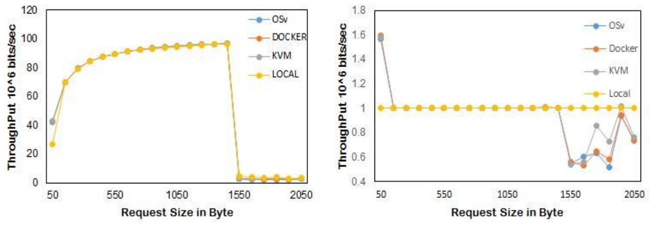

Figure 6. UDP Performance in different virtualization platform: Absolute throughput (left) and relative throughput (right) with socket size increasing from 100 bytes/sec with increment of 1000 bytes.

The figure 7 shows the okay rate of UDP running on DOCKER, KVM and OSv. The okay rate is defined as the number of okay messages divided by the total number of messages. As we can see from the graph, the okay rate of Docker and KVM as a function of request size is almost identical. However, it seems that OSv has better okay rate in range of 250 to 900 request size. But this advantage quickly diminishes as request size gets larger.

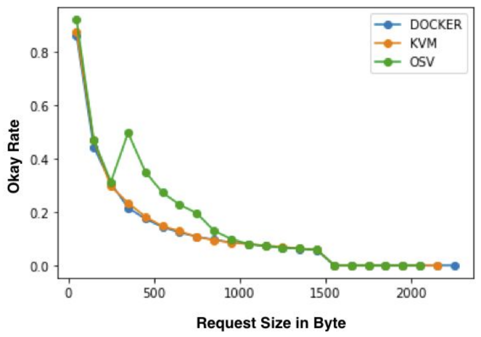

Figure 7. The okay rate of UDP running on different virtualization platform.

V. RELATED WORK AND DISCUSSION

A. Related Work

In previous study [1], the researcher conducted an experiment using Docker, Linux/kvm and Unikernel to evaluate the provisioning time within these virtualization systems on an OpenStack cloud platform. They decomposed each stage of a conventional provisioning pipeline to identify the impact of each step on the technologies. The result show that Unikernel outperforms Docker and KVM. In generally, Unikernel have smaller image size and need less start up time than docker and kvm.

Previous research [3] compared the performance of Unikernel TCPv4 stack, implementing the full connection lifecycle, fast retransmit and recovery, New Reno congestion control and window scaling, against the Linux 3.7 TCPv4 stack using iperf. They found the Unikernel receive throughput is slightly higher due to the lack of a userspace copy, while it's transmit performance is lower due to higher CPU usage. They also compare the Unikernel DNS Server application performance on Xen against Bind and NSD and found that Unikernel appliance initially performed very poorly, but dramatically improved when they introduced memorization of response to avoid repeated computation.

Another research used TCP stream test to measure the efficiency of the Unikernel TCP receive path [7]. They found that docker has the least overhead comparing with virtual machine and Unikernel; specifically, docker used bridged networking mode is the fastest having less overhead than Docker using NAT. And when Unikernel run with vhost-techniques, its performance is very close to Docker. However, when they test TCP reserves stream, which is similar to TCP stream except that the data flows in the reverse direction, they found that Unikernel has the least overhead of all the virtualization techniques; Docker and linux running on QEMU/KVM do not have significant different. They also compare UDP stream and found that Docker has more overhead than Linux and the Unikernel's performance is pretty close to Docker.

B. Discussion

For the TCP stream test, since the shape of the curve is very unpredictable, and it is very hard to compare the performance based on human eyes so that we calculated the performance based on the plot area that the plot covered. As for Figure 4, the area above the local operating system is considered as positive while the area below the local operating system is considered as negative. We find that the OSv outperforms KVM, and KVM outperforms Docker under socket restriction from 0 to 6000 bytes. Since OSv is an instance of the unikernel, and this test demonstrates that unikernel (running on hypervisor) outperforms other environments in term of TCP stream performance under the socket restriction.

For the UDP stream test, based on Figure 6, we can see that from range 0 to 1400 bytes requested size, all three different environments shared the same performance, and overperformed the local OS. However, local OS becomes dominant after 1450 bytes of requested size, and from range 1450 to 2000 bytes of requested size, the KVM outperforms Docker, and Unikernel during the UDP stream test. Another thing is that based on Firgue X, OSv has better Okay Rate in range of 250 to 900 bytes, and OSv, KVM, and Docker share the same performance throughout so that Unikernel generate more Okay transmission during the range 250 to 900 bytes, and this shows Unikernel outperforms KVM, and Container in term of Okay transmission of UDP stream test when requested size is in range of 250 to 900 bytes.

VI. CONCLUSION AND FUTURE WORK

A. Conclusions

In this project, we conducted TCP and UDP network performance test with Netperf under local environment, Docker environment, KVM environment, and OSv environment. This is the first study which quantitatively compare the throughput results with change of socket size(TCP) and requested size(UDP). We then have fully discussed the results of TCP stream test, and results of UDP stream test, and we confirmed from similar studies [7] that Unikernel has become more powerful environment for cloud computing environment than other traditional environments especially in network performance. Unikernel has been in a dominate position in TCP stream test in terms of throughput, and Unikernel create more successful transmissions than the transmission that other environment produced in UDP stream test.

B. Contributions and Summary

In this project, we quantitatively studied the network performance of virtualization technology OSv, Docker and KVM and compare them as well as performance at local operating system environment. Base on our first hand experiment, we solve real dilemma between two previous researchers[1][7], one of whom believe no performance difference among these techniques in term of TCP stream performance, while the other one of who believe so. We concluded from our experiment data that OSv, an unikernel platform, out perform other virtualization technology at certain socket size range(requested size in terms of UDP test). As we read through papers throughout related research we did not found any quantitative comparison between these virtualization methods. Some papers claims that OSv better performed than other virtualization technology, and in our experiment the claim was confirmed. Thus, we provide a alternative way to resolve the controversial of this issue which means that it comes different conclusion might be caused by the different transmission size.

The experiment results shows a promising result for unikernel implementation in cloud computing,which generate higher throughputs than other virtualization methods, and which is not yet largely used in market. More study on unikernel could be done to explore the features of the new virtualization technology. Moreover,the result shows that unikernel can be used as a promising virtualization tool in cloud computing, for demonstrating a better network performance.

C. Future work

In this experiment, we conducted network performance research using netprf. We choose four operating environments to conduct the network performance and compare their throughputs. 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.

D. Experimental artefacts

We wish to explore the possibilities of the current result by the other groups. To that end we have made available all the code used in this paper: all the code and set up documents are available on GitHub (https://github.com/TeamBroncos/BroncoLovesCloud.git)

VII. LIST OF CONTRIBUTION

- Yinghe Chen was mainly responsible for studying netperf, and designing and executing the test cases.
- Xiaoying Yang was mainly responsible for analysing the result of UDP_STEAM test and data analysis.
- Miaoyan Zhang was mainly responsible for environment setup on linux server.
- Xi Wang was mainly responsible for analysing the result of TCP_STREAM test, and visualization of the results.
- Minjie Zhang was mainly responsible for analysing the result of UDP_STREAM test.
- All members in the team were responsible for the conclusions.

ACKNOWLEDGES

We thank Professor Abhishek Gupta's impatience advising of this study and the teaching assistant Lakshmimanasa Velaga ‘s useful suggestions.

References

[1] B. Xavier, T. Ferreto and L. Jersak, "Time Provisioning Evaluation of KVM, Docker and Unikernels in a Cloud Platform," 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid),Cartagena,2016, pp. 277-280. doi: 10.1109/CCGrid.2016.86

[2] Kivity, D. Laor, G. Costa, P. Enberg, and N. Har’El. OSv Optimizing the Operating System for Virtual Machines. 2014 USENIX Annual Technical Conference, 2014.

[3] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library operating systems for the cloud. In Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’13, pages 461–472, New York, NY, USA, 2013. ACM.

[4] F. J. Ballesteros. The Clive Operating System. Oct, 2014.

[5] Galoisln, "ec2-unikernel: Tool for uploading unikernels into EC2", https://github.com/GaloisInc/ec2-unikernel, Accessed on Jan, 30th, 2017

[6] D. Bernstein, "Containers and cloud: From LXC to Docker to Kubernetes," IEEE Cloud Computing, vol. 1, no. 3, pp. 81 –84, Sep. 2014.

[7] P.Enberg, "A Performance Evaluation of Hypervisor, Unikernel,and Container Network I/O Virtualization ".Master’s Thesis University of Helsinki Department of Computer Science Helsinki, May 17, 2016.

[8] J. Rick: Care and Feeding of Netperf 2.6.X. www.netperf.org/svn/netperf2/tags/netperf-2.6.0/doc/ netperf. html. 2012 (accessed July 30, 2015).

[9] A.Bratterud, A.Walla, H.Haugerud, P.l Engelstad, Kyrre Begnum. IncludeOS: A minimal, resource efficient unikernel for cloud services. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science.

[10] I.Briggs, M.Day, Y.Guo, P.Marheine, E.Eide, “A Performance Evaluation of Unikernels”. School of Computing, University of Utah.

[11] Y. Ueda and T. Nakatani, “Performance variations of two open-source cloud platforms,” Workload Characterization (IISWC), 2010 IEEE International Symposium on, pp. 1–10, 2010.

[12] K. Razavi, S. Costache, A. Gardiman, K. Verstoep, and T. Kielmann, “Scaling VM Deployment in an Open Source Cloud Stack,” in ScienceCloud ’15: Proceedings of the 6thWorkshop on Scientific Cloud Computing. ACM Request Permissions, Jun. 2015, pp. 3–10.

[13] Y. Ueda and T. Nakatani, “Performance variations of two Open-source cloud platforms,” Workload Characterization (IISWC), 2010 IEEE International Symposium on, pp. 1–10, 2010.

[14] K. Razavi, S. Costache, A. Gardiman, K. Verstoep, and T. Kielmann, “Scaling VM Deployment in an Open Source Cloud Stack,” in ScienceCloud’15: Proceedings of the 6th Workshop on Scientific CloudComputing. ACM Request Permissions, Jun. 2015, pp. 3–10.

[15] Morabito, Roberto, Kjällman, Jimmy, and Komu, Miika: Hypervisors vs. lightweight virtualization: A performance comparison. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering, IC2E’15, pages 386–393, Washington, DC, USA, 2015. IEEE Computer Society, ISBN 978-1-4799-8218-9. http://dx.doi.org/10.1109/IC2E. 2015.74

[16] S. Lankes, S. Pickartz, J. Breitbart, “HermitCore—A Unikernel for Extreme Scale Computing”, June 01 - 01, 2016 ROSS '16 Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers ArticleNo. 4 Kyoto, Japan.

[17] M. Mao and M. Humphrey, “A Performance Study on the VM Startup Time in the Cloud,” in Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. IEEE, 2012, pp. 423–430.