PIVOTing to a new generation of Priority Inter-VM Open Transmissions

Will D. Merges
Department of Computer Science
Rochester Institute of Technology
Rochester, NY
wdm7973@rit.edu

Kayleigh A. Kolz
Department of Computer Science
Rochester Institute of Technology
Rochester, NY
kak6927@rit.edu

Brandon N. Keller
Department of Computing and Information Sciences Ph.D.
Rochester Institute of Technology
Rochester, NY
bnk5096@rit.edu

***Abstract*—As distributed processes and systems gain in prevalence, the need for fast communication between these distributed components also grows. Virtual machines, and their corresponding inter-VM communications, promise to improve the efficiency at which hosts are utilized. The goal of this work is to improve the performance of inter-VM communications in the case of the Xen hypervisor with the proposed Priority Inter-VM Open Transmissions system. We port to modern Linux versions and improve upon an existing and currently unmaintained project and perform experimentation to measure performance. We find that, in testing, PIVOT is able to yield over double the performance for inter-VM communication compared to the default pathways employed by the Xen hypervisor. These results provide evidence that communication rates, in even widely deployed virtualization platforms, have room for improvement.**

***Index Terms*—Inter-VM Communication, Distributed Systems, Xen, Hypervisor, Operating Systems**

I. Introduction

Virtual machines are a critical aspect of many modern distributed systems, and therefore, the communications that occur between virtual machines on the same host is increasingly important. These communications, as with any others may in many cases benefit by increased speeds. While traditional networked communication schemes can work well in the case of virtualized devices on a single host, there are benefits in utilizing the shared hardware to prevent the need for such typical operations.

One approach to improving the performance of Inter-VM communications is utilizing the existing network interfaces provided by the operating systems or the hypervisor being used. One project, XenLoop [1], worked to intercept packets destined for another virtual machine located on the same physical host. This project, for the Xen hypervisor [2], improved upon both the throughput and latency provided by the standard data path [1]; however, this work has since been abandoned and this approach has not been largely explored in the context of recent iterations of the Xen hypervisor or the Linux Kernel.

We improve upon the previous XenLoop implementation [3] by updating the software to support modern systems. Further, we improve upon the design to increase performance. We do this by intercepting packets at a higher layer, saving on the unnecessary steps of processing that the prior implementation did not circumvent in an implementation we call Priority Inter-VM Open Transmissions (PIVOT). We present an analysis of both our ported implementation of XenLoop, but also its improved design in PIVOT in terms of performance not only in terms of throughput and latency as inspired by the original XenLoop publication [1], but also in terms of the overhead costs imposed by utilzing either the ported XenLoop or PIVOT.

The rest of this work is organized as follows: Section II details necessary background information for the rest of the work. Section III details our porting of XenLoop and our implementation of PIVOT. Section IV presents an analysis of the performance of both the ported version of XenLoop and Pivot when compared to the base Xen implementation of Inter-VM communications. Section V is a review of related literature. Section VI explores possibilities for future work. Conclusions are drawn and explored in Section VII.

II. Background

A. Hypervisors

Hypervisors are systems that enable the hosting of multiple virtual machines on a single physical host machine. These hypervisors come in two key types. Type 1 hypervisors are meant to run directly on the hardware without an underlying host operating system to run on top of. By contrast, type 2 hypervisors require an underlying operating system to run. These two types of hypervisors are illustrated in Fig. 1.

1) Xen: Xen is one popular type 1 hypervisor [2]. Xen is designed to operate with as many as 100 virtual machines operating on a single host at a single time. This paper also made this estimate when considering only 2003 hardware,

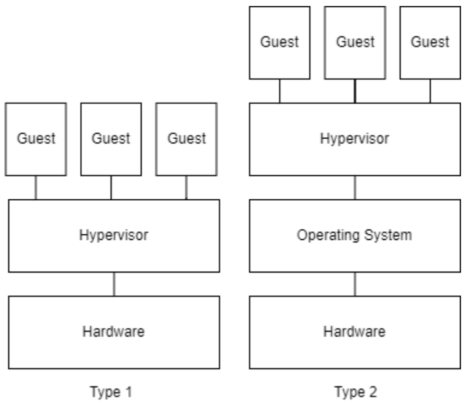

Fig. 1. Type 1 vs Type 2 Hypervisors

meaning that there is even more potential in the modern iterations of this hypervisor. This very large scale demonstrates the need for communication between individual machines located on a single host. Further, Xen uses “dom”s to describe virtual machines when compared to a host, here dom0 refers to the host system and domU refers to paravirtualized guests.

B. Inter-VM Communication

Inter-VM communication refers to the communication that occurs between VMs hosted on the same physical hardware. This type of communication is critical in the modern data center where VMs located on the same machine need to work in conjunction with one another in order to effectively carry out tasks.

1) XenLoop: XenLoop is a system developed for the Xen hypervisor that works to improve the throughput and reduce the latency of inter-VM communications [1]. This works by intercepting packets prior to proceeding with a full transmission. This allows for a drastic speed up in performance over standard network traffic; however, this project is no longer maintained, and has remained incompatible with modern operating systems and modern iterations of the Xen hypervisor.

III. Implementation

As a previously established technology, XenLoop has public source code available from which we were able to start from. Xen Virtual Machine Manager (VMM) is included in the Linux kernel source and did not need to be modified for our use. Due to the long lifespan of these systems, installation on modern hardware proved challenging. Xen hypervisor proved to be difficult to install due to bugs in the EFI boot process of the VMM. To solve this, we had to install for Legacy BIOS instead of EFI, which forced us into several re-installs before we were successful.

Further, we had to perform significant work to move XenLoop into an operable state. The XenLoop kernel modules were designed for Xen version 3.2.0, which corresponds to Linux kernel version 2.6. The last released version of XenLoop was only tested for kernel version 2.6.18, released in 2007. We ported both the module for the dom0 and the module for the domUs to Linux version 5.4.0, updating the modules for over a decade of kernel changes [4]. One concern was that the device naming scheme has changed. XenLoop utilized hard-coded network device names that were not able to be dynamically changed and did not reflect the modern naming scheme. To alleviate this issue and make it more flexible for the future, we added a parameter when loading the modules where the user can set the device name. To add to this concern, the socket buffer data structure which is heavily utilized across the network stack has changed since the original publication of XenLoop. In particular, current revisions encapsulate the previously accessible direct memory addresses with offsets to base addresses. Additionally, XenLoop requires hooking into the network stack which used unsafe functionality to be executed in the interrupt context of a nethook. In order to construct First-In-First-Out (FIFO) queues between domUs, each domU must map pages in it’s virtual address space in order to shared those pages with another domU using the Xen grant table. XenLoop assumed being able to arbitrarily remap pages within a nethook, an operation that is no longer allowed as it could cause I/O blocking if pages need to be remapped in the Memory Management Unit (MMU) device. We had to change this mapping to use contiguous pages that did not be need to remapped, essentially using the “kmalloc” kernel function rather than allocating a virtual memory area. XenLoop also notifies connected domUs when data has been written to the FIFO, this mechanism is implemented using a Xen hypercall to send a signal over the Xen event channel. When processing a packet in the nethook, XenLoop assumed it could disable interrupts and use this hypercall, which is a dangerous operation that could potentially lock the CPU. While providing the basis for our implementation, we first needed to make these modifications to XenLoop in order to achieve baseline functionality on a modern kernel. We were able to complete our baseline port of XenLoop and tested it on a Linux version 5.4.0-107 dom0 with Linux version 5.4.0-42 domUs communicating with each other.

After porting, we also sought to increase the performance of the base XenLoop design. We noticed that XenLoop only operates at the link layer, employing the use of Media Access Control (MAC) address lookups in a Netfilter [5] hook to check for Ethernet packets destined for co-resident guests on the same host. Our improvement was to hook packets at the network layer, higher in the stack, rather than at the link layer [4]. XenLoop uses the ‘NF_INET_POST_ROUTING‘ hook whereas we use the ‘NF_INET_LOCAL_OUT‘ hook, which happens earlier in the processing of an outgoing packet. Our strategy essentially moves the Netfilter hook for outgoing packets up a layer, shortening the data path that a packet sent between domUs takes by reducing the amount of work performed in the Linux networking stack before it is rerouted by our kernel module. The trade-off with moving up a layer is that we have to fix another protocol, similar to how XenLoop fixed the link layer protocol as Ethernet. We decided to fix our network layer protocol to Internet Protocol version 4 (IPv4), as Internet Protocol (IP) is the most common network protocol and IPv4 provides short 32-bit addresses that are fast and easy to hash. Our implementation hooks packets higher in the stack and does an IPv4 lookup on the destination address to determine if the packet is destined for a co-resident guest, then uses the same FIFO structures as XenLoop to route the packet if it can. Since this work is saved for every outgoing packet, we provide a constant speedup for traffic between domUs.

By switching to IP lookups rather than MAC lookups, we still had the problem of determining the IPv4 addresses of co-resident guests. XenLoop utilizes the XenStore mechanism to poll the MAC addresses of each guest's virtual network device from the dom0 and send that information to each domU. Unfortunately, the IP address used by each domU can not be polled using the same mechanism, the guest can choose to configure their IP address however they would like whereas the MAC address is a unique address assigned to the virtual device that the guest OS should use without changing. Without modifying the kernel of the guest OS, we could not easily advertise the currently assigned IP address in the XenStore, and this modification would make our improvements less flexible as our mechanism would not be able to be simply loaded as a kernel module onto each domU. We instead decided to employ the use of another hook in order to intercept and read incoming Address Resolution Protocol (ARP) packets, using the 'ARP_IN' Netfilter hook. When we receive an ARP packet with a source hardware address matching a MAC address we know belongs to a domU on the same host, we can record the mapped IPv4 address in a table to use for our lookups in our hook for outgoing packets. We learn the MAC addresses of each co-resident domU by using the original XenLoop mechanism where the dom0 sends MAC address and domid pairs describing all the domUs on the system. We store these MAC addresses in the same table that XenLoop does and keep a separate table which keys the same table entries to IPv4 addresses as soon as they are learned through ARP.

We now have a way to build a table that stores connection information and is keyed with IPv4 addresses, allowing us to perform lookups on the destination IP address of outgoing packets. When a lookup succeeds, we use the same connection mechanism as XenLoop to setup two FIFO queues to copy data in to and out of, avoiding the standard netback/netfront path. Making these changes allowed us to avoid some steps of the networking stack for each packet which increases the overall performance when compared to XenLoop. Additionally, our changes only minimally increase the overhead compared to XenLoop as each nethook still only does one lookup, and in the case of IPv4 the address hashed is actually smaller. The domU kernel module utilizes slightly more memory to store the handler for the IP lookup table, but this is a fixed overhead as each entry in the table is a reference to an entry in the MAC table that XenLoop already uses, so we do not need to allocate extra memory for each connection.

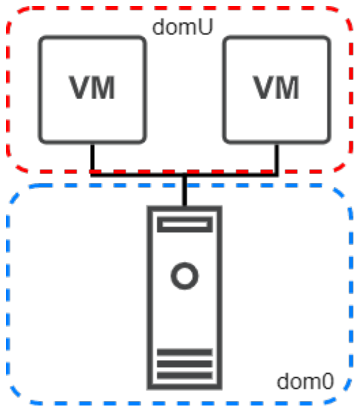

Fig. 2. Diagram of the test environment utilized in this study

IV. Performance Evaluation

A. Test Design

We wanted to compare the standard netback/netfront path used by Xen with ported XenLoop and our performance updates resulting in PIVOT. The performance metrics we used to compare were total throughput as well as round trip latency. Metrics were measured using the standard *netperf* [6] tool, we performed TCP round robin (TCP_RR) and UDP round robin (UDP_RR) tests. The tests used only IPv4 as the network layer protocol and had a duration of 120 seconds each. Five rounds of 120 second tests were performed and averaged for each test configuration. Tests were performed between two guest domUs on the same host, representing inter-VM traffic, as well as between a domU and the dom0, representing traffic sent outside of the inter-VM mechanism to measure the overhead that XenLoop and PIVOT create by hooking every outgoing packet. The tests were performed from a Xen enabled host with the Ubuntu 18.04 operating system on kernel version 5.4.0-107. Two domUs were installed on the dom0, both were Ubuntu 18.04 operating systems with kernel version 5.4.9-42. The host machine had an Intel i5-2400 clocked at 3.10 GHz and 12 GB of RAM. Each domU was allocated 1 GB of memory and given a virtual network device with a unique MAC address. The dom0 used the credit based scheduler that comes default with Project Xen. This setup is visualized in Fig. 2.

B. Results

1) Ported XenLoop: In Table I and Table II we can see the results of communicating between domUs using the base netfront/netback path and ported XenLoop. Our XenLoop port achieved 2.17 times as much throughput and 2.21 times less latency in the TCP_RR test and 1.81 times as much throughput and 1.84 times less latency in the UDP_RR test.

Table III and Table IV show that throughput and mean latency for packets not routed by XenLoop are minimally impacted despite being hooked by our kernel module to do an address lookup during their processing. A comparison of the ported XenLoop implementation to the baseline netback/netfront implementation over time can be found in Fig. 3.

2) PIVOT: Compared to the baseline netback/netfront, the performance of PIVOT shown in Tables I and II achieves 2.19 times as much throughput and 2.23 times less latency in the TCP_RR test and 1.82 times as much throughput and 1.84 times less latency in the UDP_RR test. Compared to the ported XenLoop implementation, PIVOT achieves 3.5% more throughput speedup and 4.96% more latency reduction in the TCP_RR test and 1.25% more throughput speedup and 1.28% more latency reduction in the UDP_RR test.

Similarly to the XenLoop implementation, PIVOT does not impactfully decrease throughput or increase latency for packets not going through the inter-VM mechanism despite being hooked by our kernel module for an address lookup, which is shown in Tables III and IV.

While the performance increase is small, since we are increasing the rate at which packets can be transmitted over large amounts of time the total number of transactions that can occur increases linearly over time. This increasing difference can be seen in Fig. 4 which shows the total number of TCP transactions that can occur over a one day period with the base netback/netfront implementation and with PIVOT. This linearly increasing difference can also be seen between base XenLoop and PIVOT in Fig. 5, although between a timeframe of 1 and 2 months as the speedup is much less than the initial gain over netback/netfront.

V. Related Works

Inter-VM communications are critical for many distributed systems, and has therefore been subject to a great deal of study. The work presented in this paper is primarily influenced by XenLoop, [1]. In this foundational work, the authors present a loopback channel for the Xen hypervisor [2]. While presenting encouraging performance results compared to the original Xen hypervisor, the work presented here seeks to improve upon [1] further by intercepting packets at a higher layer. The rest of the section outlines five additional related works and provides context as to how the work presented herein differs.

A. Shared Memory Communications

One approach that has been utilized for efficient inter-VM communications has been that of shared memory. While this approach differs from that proposed within this work, the related works in this area provide insight into larger mission of improved inter-VM communication efficiency.

One such work in this area is [7]. This work presents a shared memory communication approach for the Xen hypervisor, the same hypervisor of focus for this paper. In [7] the authors present MMNet, a kernel module that positions a VMs memory in such a way that it is within the address space of a communicating peer. They then provide signalling between peers via the Xen event channel. Of note is that in performance testing, MMNet was compared with XenLoop [1], and performance was stronger. Specifically this approach does not suffer from the same latency deterioration as XenLoop in larger request sizes. While the approach presented in this paper is based on XenLoop, we work to improve efficiency and speeds to challenge the performance of [7].

Another work which builds on top of the Xen hypervisor is [8]. This work presents YASMIN (Yet another shared memory implementation for intra-node), which builds on top of Vchan, a library for Xen that allows intra-node communication between different VMs. YASMIN exploits Xen's *grant-table* and *event channel* mechanisms to enable page sharing between co-located VMs. This creates a communication channel between location aware VMs and allows them to bypass TCP/IP stack and also does not reduce transparency.

In the case of other hypervisors, there has also been a great deal of research. One such example is [9]. In this work, the authors present a shared memory approach for the KVM hypervisor [10]. In the approach, the authors utilize a virtual communication device that exposes, at a low level, a message passing interface. For messages under 32 KB in size, shared memory buffers are utilized for transmissions. In larger cases, those messages exceeding 32 KB in size, communicate via direct copies of send a receive buffers after a process of transferring DMA requests. In testing it is found that the implementation proposed exceeds performance of peers for the larger message transfers, but not in the case of the smaller messages.

B. Transparent Socket Communications

Some works take a similar approach to XenLoop in using the existing socket abstraction and managing inter-VM communication channels below that layer. Since the socket abstraction is already used by networked applications, XenLoop and these similar approaches share the advantage of being transparent at the application layer.

One such work that provides this transparent operation is [11]. In this work a communication mechanism called XWAY is presented. XWAY is also built for the Xen hypervisor but intercepts packets at a higher layer in the network stack. XWAY intercepts TCP packets at the transport layer whereas XenLoop [1] intercepts Ethernet packets at the link layer. XWAY is able to bypass part of the TCP/IP stack that XenLoop is not able to which offers greater performance at the cost of only supporting TCP. Our approach seeks to obtain the same performance increase by intercepting packets at a higher layer, but not excluding popular transport protocols besides TCP, notably UDP and QUIC.

Another work leveraging the socket abstraction is [12] which presents a system called Fido. Fido intercepts traffic at the link layer, identically to XenLoop [1], but improves performance by having a true zero-copy data path. The XenLoop data path is copy-in copy-out, whereas Fido directly maps memory used by both domU kernels to the same physical addresses, using the MMNet system presented in [7]. This improves the primary bottleneck of XenLoop, but we believe additional performance can be gained by bypassing lower layers of the networking stack.

TABLE I
TCP PERFORMANCE ACROSS THE BASE XEN IMPLEMENTATION, PORTED XENLOOP, AND PIVOT IN A DOMU TO DOMU SCENARIO

	netback/netfront		**Port XenLoop**		**PIVOT**	
RUN	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**
0	3513.398	278.36	7407.502	130.32	8075.283	119.51
1	3500.042	279.52	8342.644	115.6	7534.403	128.19
2	3484.548	280.78	7494.407	128.81	7585.527	127.27
3	3611.481	280.78	6997.096	138.12	7406.572	130.34
4	3466.039	282.3	7894.293	122.29	7820.824	123.43
Mean	3515.1016	280.348	7627.1884	127.028	7684.5218	125.748

TABLE II
UDP PERFORMANCE ACROSS THE BASE XEN IMPLEMENTATION, PORTED XENLOOP, AND PIVOT IN A DOMU TO DOMU SCENARIO

	netback/netfront		**Port XenLoop**		**PIVOT**	
RUN	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**
0	4532.116	215.22	7752.987	124.29	7630.043	126.43
1	4178.612	233.5	7950.527	121.21	7978.024	120.79
2	4377.313	222.93	7784.927	123.86	7697.703	125.27
3	4160.946	234.43	7987.684	120.73	8115.252	118.71
4	4210.67	231.73	7454.394	129.4	7657.278	125.94
Mean	4291.9314	227.562	7786.1038	123.898	7815.66	123.428

TABLE III
TCP PERFORMANCE ACROSS THE BASE XEN IMPLEMENTATION, PORTED XENLOOP, AND PIVOT IN A DOMU TO DOMO SCENARIO

	netback/netfront		**Port XenLoop**		**PIVOT**	
RUN	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**
0	7059.771	137.65	6553.41	148.46	7470.856	129.96
1	7542.899	128.72	7351.927	123.86	7310.48	132.84
2	6977.182	139.3	7088.82	137.1	6945.429	139.95
3	6966.152	139.51	7235.566	134.24	7205.411	134.78
4	6724.484	144.65	6782.372	143.36	6617.443	147.01
Mean	7054.0976	137.966	7002.419	137.404	7109.9238	136.908

TABLE IV
UDP PERFORMANCE ACROSS THE BASE XEN IMPLEMENTATION, PORTED XENLOOP, AND PIVOT FOR DOMU TO DOM0 TRAFFIC

	netback/netfront		**Port XenLoop**		**PIVOT**	
RUN	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**	**Transactions / s**	**Mean Latency (μs)**
0	6729.44	144.41	6800.738	142.9	6729.481	144.4
1	6849.129	141.88	6759.464	143.75	7014.038	138.5
2	6972.755	139.35	7377.502	131.6	7038.763	138
3	6903.296	140.72	6596.114	147.36	6788.496	143.12
4	6491.321	149.76	6963.478	139.48	6830.996	142.26
Mean	6789.1882	143.224	6899.4592	141.018	6880.3548	141.256

C. Other Xen Applications

Both the Xen hypervisor and XenLoop have been highly influential. This is partially why we chose to look into XenLoop for our experiments. One work [13] demonstrates the inter-VM communication performance gains from using XenLoop. First, these experiments demonstrated that co-resident inter-VM communication methods do not cause any sort of performance degradation. In addition, even when only part of a workload is done between two co-resident VMs, the network throughput is gained in orders of magnitude. Also, this work demonstrated that XenLoop improves performance for both UDP and TCP workloads. These results conclude that when the co-resident inter-VM communication method XenLoop is added, the network will benefit from it in terms of performance.

An area in which Xen was used to gain performance was for mechanically steerable sensor networks. [14] In this experiment, Xen was used to build a program called MultiSense, which multiplexes the resource of controlling a sensor's actuators. This addresses the problem that mechanical sensors have that only one person or application can control them at a time. MultiSense utilizes Xen to implement proportional-share scheduling as well as state restoration, request batching and merging, and anticipatory scheduling for the sensors. In doing so, this reduces the very high cost of operating the sensors. This differs from our approach since it seeks to improve a physical medium utilizing Xen. Whereas, our approach did not handle any sort of physical medium. This further demonstrates the utility of Xen.

In addition, another work that boosted performance with

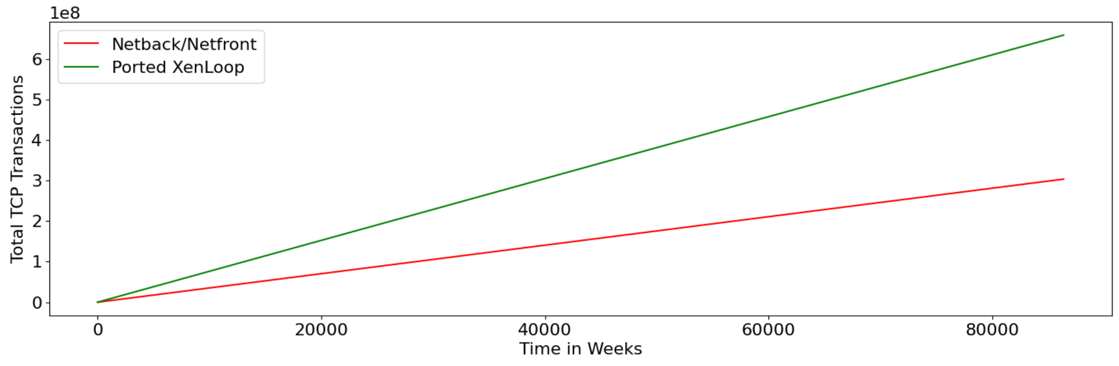

Fig. 3. The total TCP transactions processed by both the ported XenLoop implementation and the base netback/netfront implementation over the course of a day based on average performance.

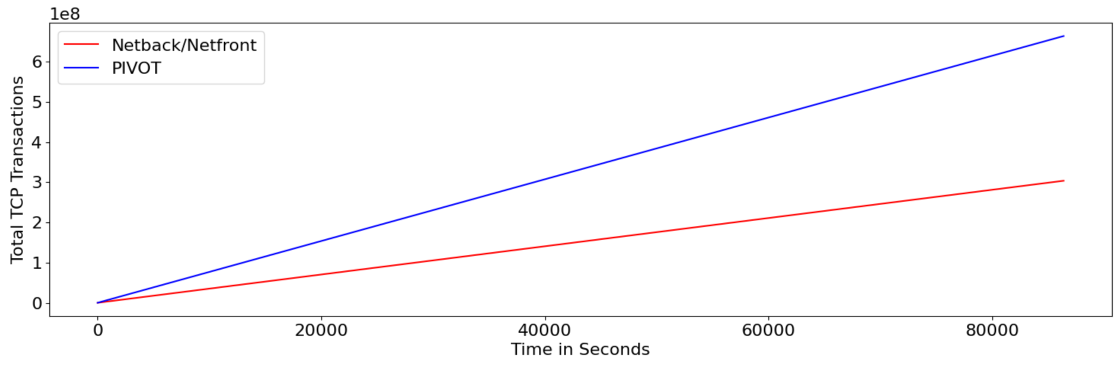

Fig. 4. The total TCP transactions processed by both PIVOT and the base implementation over the course of a day based on average performance.

Xen, aimed to build upon Xen by implementing a performance aware power capping orchestrator for Xen. [15] The paper presented XeMPUPiL, which extended the current implementation of PUPiL, to make it compatible with a virtualized environment without modifying guest workloads. XeMPUPiL exploits Intel RAPL hardware interface to set a strict limit on the processors' power consumption. Then, a software level ODA loop explores all of the available resource allocations, to find the most power efficient for the running workload. XeMPUPiL is able to achieve a very high performance under a variety of power caps. Our research does not take into account power of any sort.

A final work, [16] deconstructs three binaries in the Xen hypervisor to address over approximation of memory used by assembly code and formal verification of memory usage in compiled code. This is an important topic because it is needed to make sure functions do not interfere with each others stack frames. Also, it plays a role in security issues such as validating the integrity of a return address. This experiment demonstrated that a verified memory usage certificate could be generated on Xen hypervisor binaries. For 71% of the functions within these binaries, a verified certificate could be generated, meaning the memory approximation was more accurate and not overestimated as much. This in turn also boosted performance. Our research does not take memory into account and only sought to improve performance.

D. *Other Inter-VM Communication*

Research into inter-VM communications more generally is widespread. One paper in this area is [17]. In this work, explores an approach to these communications that merges the two common approaches, networked datapaths and shared memory, to create a solution capable of supporting VM migrations across hosts. While this work does work to improve the performance of inter-VM communication within a single host, it also, unlike the work we present in this paper, works to provide functionality across an entire data centers in targeting use cases where virtual machine migrations are common. In the case of networked hosts, they utilized normal network communications, but constantly search for new co-located VMs to then swap to a shared memory approach. They find that their approach achieves 50% higher performance than observed when utilizing the standard TCP/IP stack.

Another work exploring inter-VM communication is [18]. This work's focus is on the security of cloud computing, specifically in the context of inter-VM traffic. The proposed approach is network based and is centered upon making communications attributable to a specific VM. This is achieved by modifying IP packets to include additional information. This information includes tags that indicate the source of the

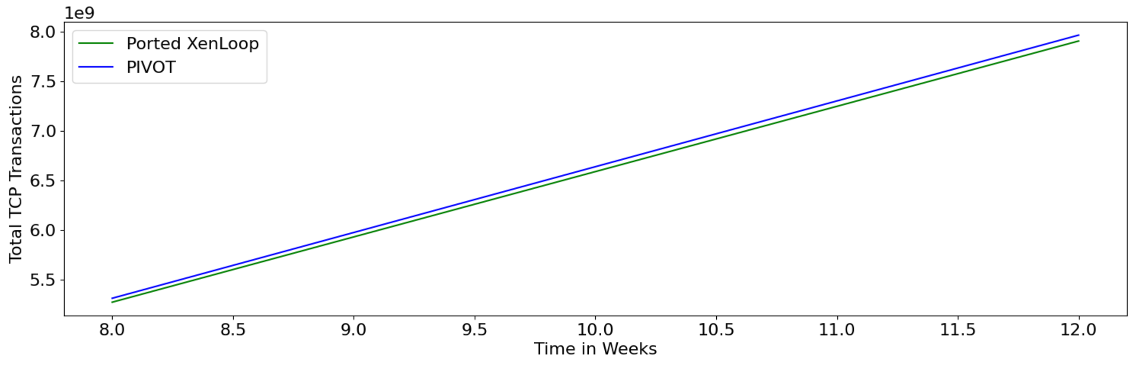

Fig. 5. The total TCP transactions processed by the ported XenLoop implementation and PIVOT over the course of several weeks based on measured average performance.

message with a signature. This work differs greatly from our own in that it is centered upon network based traffic where our work strives to improve performance of VMs on the same host that may not require communications to occur over the network. Results are not explored within [18], but they note that there are several stages of encryption and checks, suggesting that speed is sacrificed in favor of yielding a more secure set of transmissions.

Further work was conducted in [19]. This work, in a divergence from our own, investigates asynchronous inter-VM communications in place of the synchronous communications that we explore in this paper. Further, while we build on the Xen Hypervisor [2], [19] instead builds on OKL4 [20]. They find that they can deliver highly performant data transfer through their asynchronous method with a decrease in required data transfer time by a measure of about 20%. With those results encouraging on their own, this work further emphasizes that this work serves primarily of a check of feasibility for further exploration also among these lines.

One paper that differs from those previously discussed is [21]. In this work, instead of investigating direct communications between virtual machines in an effort to improve the speed and efficiency of these messages, the authors opt to develop an intrusion detection system analysis system called ICAS. This ICAS system leverages inter-VM communications to enhance the performance of these analyses so they are more fit for purpose in the context of data centers. They utilized Open vSwitch [22] in achieving the strong performance metrics that they observed.

VI. Future work

There are several improvements to our implementation [4] that we believe can still be made. Firstly, we believe we may be able to get marginally more performance by changing where received packets are introduced into the networking stack. Currently, when a packet is sent over the FIFO and the domU is notified on the Xen event channel, the received socket buffer is copied and then sent up the stack using the same ‘netif_rx‘ function that a device driver would use. However, since PIVOT does not use the link layer, it can introduce packets at the IP layer of the stack, using a kernel function like ‘ip_local_deliver’ or similar. Unfortunately, in kernel version 5.4.0 that we worked with, these functions are not exported for linking with kernel modules, so without modifying the kernel we can not make this improvement. We believe there would be a similar small gain to what we saw as we are shortening the data path further, however modifying the kernel would make our implementation less flexible as PIVOT can currently be simply loaded as a module into unmodified Linux.

We also believe there could be major improvements to the FIFO mechanism. We did not modify the mechanism that XenLoop used where the sending domU copies data into a FIFO and the receiving domU copies it out, requiring two copy operations per transfer. If a new strategy were implemented that used a single copy we believe this would offer a major improvement. Implementations like Fido [12] use this strategy with good results. Our improvements do not conflict with the improvements made by implementing single copy transfer, so a system that uses both improvements would see compounding improvements.

One of the other improvements we would like to make is support for Internet Protocol version 6 (IPV6). Our implementation only reroutes IPv4 traffic currently, but in the future there may be a need to extend this to IPv6. The hash table that maps IP addresses to connection information would need to be updated for storing the larger 128-bit keys, which would make hashing marginally slower. Our implementation also relies on the ARP protocol to map MAC addresses to IPv4 addresses, however IPv6 does not use ARP and instead uses the Neighbor Discovery Protocol (NDP). In order to map IPv6 addresses the MAC addresses, a Netfilter hook for Internet Control Message Protocol version 6 (ICMPv6) would need to be added that looks for NDP control messages that map MAC addresses to IPv6 addresses.

VII. Conclusions

This paper investigated inter-VM communications in the context of the Xen Hypervisor and worked to improve communication rates compared to previous efforts. We presented an updated version of XenLoop and further improved upon the design with our PIVOT system. Additionally, we suggest areas for future improvements to further the performance of communications between virtual machines hosted on the same instance of the Xen hypervisor. The results indicate that this is a fruitful area of research with performance more than two times better than that observed in the netback/netfront implementation of the Xen hypevisor without utilizing either XenLoop or PIVOT. The results also show that overhead is minimal, suggesting that there is great benefit and little cost in implementing these tools in production environments. These tools also do not require modification of applications in order to see the performance benefits as the normal socket system calls are still used and the packets are transparently routed through a faster channel without the user even needing to know if the destination is a VM guest on the same physical host. Together, these results and the related works show the need for and benefit that can be gained from the implementation of the PIVOT system for inter-VM communications.

References

[1] J. Wang, K.-L. Wright, and K. Gopalan, "XenLoop: a transparent high performance inter-vm network loopback," in *Proceedings of the 17th international symposium on High performance distributed computing - HPDC '08*, (Boston, MA, USA), p. 109, ACM Press, 2008.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 164–177, Dec. 2003.

[3] "Binghamton Operating Systems and Networks Lab - Projects - XenLoop," 2022.

[4] "WillMerges/XenLoop-ResearchProject: XenLoop source modifications," 2022.

[5] R. Rosen, "Netfilter," in *Linux Kernel Networking: implementation and theory*, The expert's voice in open source, pp. 247–278, New York, NY: Apress, 2014.

[6] "Netperf."

[7] P. Radhakrishnan and K. Srinivasan, "MMNet: An Efficient Inter-VM Communication Mechanism," p. 3.

[8] M. Rozis, S. Gerangelos, and N. Koziris, "Yasmin: Efficient intranode communication using generic sockets," *Lecture Notes in Computer Science*, pp. 617–628.

[9] F. Diakhaté, M. Perache, R. Namyst, and H. Jourdren, "Efficient Shared Memory Message Passing for Inter-VM Communications," in *Euro-Par 2008 Workshops - Parallel Processing* (E. César, M. Alexander, A. Streit, J. L. Träff, C. Cérin, A. Knüpfer, D. Kranzlmüller, and S. Jha, eds.), vol. 5415, pp. 53–62, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[10] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, pp. 225–230, Dttawa, Dntorio, Canada, 2007.

[11] K. Kim, C. Kim, S.-I. Jung, H.-S. Shin, and J.-S. Kim, "Interdomain socket communications supporting high performance and full binary compatibility on Xen," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments - VEE '08*, (Seattle, WA, USA), p. 11, ACM Press, 2008.

[12] A. Burtsev, K. Srinivasan, P. Radhakrishnan, L. N. Bairavasundaram, K. Voruganti, and G. R. Goodson, "Fido: Fast inter-virtual-machine communication for enterprise appliances," in *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, (USA), p. 25, USENIX Association, 2009.

[13] Qi Zhang, Ling Liu, Yi Ren, Kisung Lee, Yuzhe Tang, Xu Zhao, and Yang Zhou, "Residency Aware Inter-VM Communication in Virtualized Cloud: Performance Measurement and Analysis," in *2013 IEEE Sixth International Conference on Cloud Computing*, (Santa Clara, CA), pp. 204–211, IEEE, June 2013.

[14] N. Sharma, D. Irwin, M. Zink, and P. Shenoy, "MultiSense: proportional-share for mechanically steerable sensor networks," *Multimedia Systems*, vol. 18, pp. 425–444, Oct. 2012.

[15] M. Arnaboldi, M. Ferroni, and M. D. Santambrogio, "Towards a performance-aware power capping orchestrator for the Xen hypervisor," *ACM SIGBED Review*, vol. 15, pp. 8–14, Mar. 2018.

[16] F. Verbeek, J. A. Bockenek, and B. Ravindran, "Highly Automated Formal Proofs over Memory Usage of Assembly Code," in *Tools and Algorithms for the Construction and Analysis of Systems* (A. Biere and D. Parker, eds.), vol. 12079, pp. 98–117, Cham: Springer International Publishing, 2020. Series Title: Lecture Notes in Computer Science.

[17] M. Kurtadikar, A. Patil, P. Toshniwal, and J. Abraham, "An Inter-VM Communication Model Supporting Live Migration," in *2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies*, (Pune, India), pp. 63–68, IEEE, Nov. 2013.

[18] K. Benzidane, S. Khoudali, and A. Sekkaki, "Secured architecture for inter-VM traffic in a Cloud environment," in *2nd IEEE Latin American Conference on Cloud Computing and Communications*, (Maceio, Brazil), pp. 23–28, IEEE, Dec. 2013.

[19] R. Wang, L. Xu, Y. Bai, K. Cheng, Z. Wang, G. Luan, H. Yang, and W. Wang, "Research on Asynchronous Inter-VM Communication Mechanism Based on Embedded Hypervisor," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, (Tokyo, Japan), pp. 695–700, IEEE, July 2018.

[20] G. Heiser and B. Leslie, "The OKL4 microvisor: convergence point of microkernels and hypervisors," in *Proceedings of the first ACM asia-pacific workshop on Workshop on systems - APSys '10*, (New Delhi, India), p. 19, ACM Press, 2010.

[21] S.-F. Yang, W.-Y. Chen, and Y.-T. Wang, "ICAS: An inter-VM IDS Log Cloud Analysis System," in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, (Beijing, China), pp. 285–289, IEEE, Sept. 2011.

[22] "Open vSwitch," 2022.