Circuit Switched VM Networks for Zero-Copy IO

Johannes Krude, Mirko Stoffers, Klaus Wehrle
Communication and Distributed Systems
RWTH Aachen University
{krude,stoffers,wehrle}@comsys.rwth-aachen.de

ABSTRACT

Although applications are nowadays often executed in virtual machines (VMs) to isolate applications or consolidate physical machines, VM network performance is still challenging. Packetization, encapsulation, congestion control, preparations for loss, and copying of data introduce unnecessary performance degradation within a system where VMs communicate over abundant and reliable shared-memory. Although protocols like TCP are therefore not well suited for kernel network stack in VMs, preexisting applications require the kernel socket interface to keep functioning.

In eliminating the unnecessary overhead for inter-VM communication and shifting it to the host operating system for communication over a physical NIC, our approach increases performance for both cases of communicating with another VM on the same host and for communicating with external hosts. Instead of multiplexing multiple connections over a single virtual Ethernet link, we use a separate shared-memory connection for each VM application socket. Our approach improves the stream and datagram performance of existing applications over an unmodified socket interface and brings the benefits of memory-mapped zero-copy IO to modified applications without sacrificing isolation between sockets.

CCS CONCEPTS

• **Networks → Network architectures**;

KEYWORDS

VM Network Architecture, Zero-Copy IO, Interface Compatibility

ACM Reference Format:
Johannes Krude, Mirko Stoffers, Klaus Wehrle. 2018. Circuit Switched VM Networks for Zero-Copy IO. In *KBNets'18: ACM SIGCOMM 2018 Afternoon Workshop on Kernel Bypassing Networks , August 20, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3229538.3229539

1 INTRODUCTION

Executing applications in virtual machines (VMs) is a widely accepted pattern not only to consolidate server machines into the cloud but also to offload functionality into the edge, with, e.g., Cloudlets [22], to provide isolation in multi tenant container environments, and to rigidly compartmentalize applications on client machines [21]. Applications in these VMs get TCP and UDP network access through the socket interface. Data is packetized and transmitted over virtual Ethernet to the host operating system (OS) and there further forwarded to another VM or the physical NIC.

Encapsulating and decapsulating data in the TCP, IP and virtual Ethernet protocol on each VM is resource intensive and induces overhead for each individual packet in per packet forwarding and policy decisions. Additionally, processor sharing between VMs leads to anomalies in TCP congestion control [24]. Since main-memory is abundant and reliable, techniques such as multiplexing over a shared link, congestion control, and retransmissions, are unnecessary for VMs. Protocols such as TCP should not be processed by an OS Kernel in a VM. We therefore propose to replace packet switched VM networks by a novel approach to improve the communication in between VMs and to external hosts.

Existing applications depend on the functionality of the socket interface. Our goal is therefore to provide the same behavior of the socket interface although we propose a radical modification of the underlying communication. Instead of packetizing data over virtual Ethernet, we negotiate a separate shared-memory based connection for each socket. This provides performance benefits for existing applications and enables us to extend the socket interface with zero-copy functionality to be used by modified applications. Unlike other approaches, each application decides individually whether to use the new zero-copy interface, and does not get access to the sockets of other applications.

With our approach, no packetization is done when communicating between VMs on the same physical host, resulting in a factor of up to 15.4 in goodput increase. When communicating with external hosts, TCP processing is shifted from the VM to the host OS, showing potential to saturate 40 GBit NICs. The simplicity of our approach provides several benefits to VM deployment. Since every socket gets a connection separated in memory from all other connections, we call this approach circuit switched VM networks.

Our contributions are as follows: We (1) replace virtual Ethernet by a circuit switched network accessible over an unmodified socket interface with extensions to provide per socket zero-copy IO for modified applications. Further we (2) discuss several qualitative benefits of this architecture. Through our evaluation we (3) show the performance benefits of our appoach.

2 PROBLEM & REQUIREMENT ANALYSIS

Our approach is designed to be used in lieu of any datagram or stream based protocol. To this end, we need to resemble the functionality provided by such protocols. In the following, we discuss our analysis of the most common datagram and stream based protocols, namely UDP and TCP. We first analyze the socket interface, followed by a discussion of the major mechanisms the protocols provide.

2.1 The Socket Interface

From an application's perspective the interface to the network is the socket interface. To enable unmodified applications to communicate through our system, we need to closely resemble every detail of that interface. Additionally, we can provide extensions for improved performance, which is only achieved by application that use our extensions.

We analyze the POSIX interface of the most common transport protocols TCP and UDP. Fig. 1 depicts the most important states and API functions. A TCP socket is created to connect to a remote host, listen for incoming connection requests, or by accepting an incoming connection request. Communication is only possible once the socket is connected to a remote host. A UDP socket starts unbound and is either bound manually or by transmission of the first datagram. Communication is then only possible in the bound state.

We need to provide this API and internally track the socket state to provide the correct responses of the resembled socket. For example, a listening socket should create a new socket when an incoming connection request is accepted while a non-listening socket should refuse to do so. When providing new mechanisms such as zero-copy IO, we still need to provide the existing interface to allow unmodified applications to keep functioning.

2.2 Protocol Mechanisms

We identified the following mechanisms as important components of TCP, UDP, or both that need special handling.

Connection Management. Since TCP is connection oriented while UDP is not, our approach needs to support both modes. It is necessary to create shared buffers to exchange data, hence, by definition we always maintain a kind of connection. While connection oriented communication hence maps directly to our approach, we need to ensure that for connection-less communication we still provide the same interface, although a connection is set up under the hood. Since applications need to establish a socket anyway, even to use connection-less UDP, we can create the underlying connection for a socket upon suitable actions on the existing socket interface.

Connection establishment is often guarded by firewalls which enforce connection policies and drop all packets not associated to a permitted connection. To this end, the firewall needs to maintain state about existing TCP connections and previous UDP datagrams. Such configurations are much easier to realize in circuit switched networks as only the connection setup needs to be checked and the actual data packets need not be inspected. The latter only needs to be done for firewalls that inspect payload data.

Finally, we need to have an addressing system to identify communication partners in VMs and external hosts.

Addressing. TCP and UDP multiplex connections and need to be able to address each application separately. To this end, the pair of IP address and port number is used as a unique identifier. We still need addressing for connection setup and therefore use these addresses as well.

Our approach needs to handle connections to external hosts differently from connections to other VMs. We therefore decide upon connection setup whether to build a direct circuit connection or, to translate the communication to TCP and UDP, similar to a SOCKS Proxy [13].

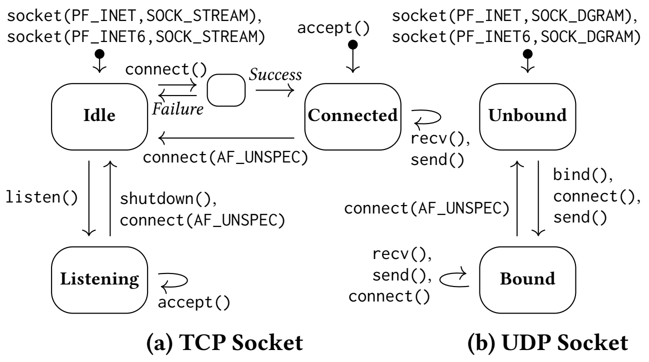

Figure 1: Most important socket states and functions.

Since IPv4 addresses are generally a scarce resource, techniques such as network-address-translation are used to share IP addresses and port numbers between VMs. Hence, we also need to support address sharing between VMs.

Flow Control. To avoid overloading the receiver with too much data TCP establishes flow control. Flow control essentially communicates the size of the receiver buffer to the transmitter and prevents the transmitter from sending more data than the receive buffer can fit. When the transmitter can directly write into the buffer of the receiver, flow control is easy to realize as the amount of available space is easily determined by the sender. However, when the communication partner resides on a remote host and the connection is translated to another TCP connection, we need to communicate the receiver window towards the sender.

Loss, Congestion, and Reordering. Other issues of communication systems are packet loss, congestion, and reordering. For direct communication via a single shared buffer, neither of those can occur: If some payload can not be stored in the receive buffer immediately, it is not transmitted at all. Hence, the packet is neither corrupted nor lost at a full queue and congestion does not occur on the network. Since main memory is reliable, payload on a single connection is always received in the order it is transmitted.

Hence, loss recovery, congestion avoidance, and reordering only needs to be handled for the proxy TCP connection established towards a remote host.

Now that we have identified the requirements we continue with a description of our approach.

3 CIRCUIT SWITCHED VM NETWORKS

In the following we introduce our general approach and discuss how we solve the issues raised in the previous section as to enable a circuit switched VM network that is compatible with existing applications.

3.1 General Approach

The general approach is outlined and compared to virtual Ethernet in Fig. 2. Traditionally, a network stack handling TCP, UDP, IP, and Ethernet is established on every VM where data is packetized and then forwarded through the host OS. We instead use for each connection a separate *circuit* consisting of shared-memory. This already reduces overhead in eliminating the need to packetize payload in the VM and using a single buffer simultaneously as receive and transmit buffer for both sides. A separate circuit for each socket also enables zero-copy IO leading to additional overhead reduction for modified applications.

Circuits are established directly between VMs or, in the case of communicating with an external host, between a VM and a proxy stack in the host OS. The less privileged VM must always provide the buffer memory to prevent denial-of-service attacks [26]. Requests to establish a circuit are forwarded by the *switch operator* in between VMs and between VMs and the proxy stack. The switch operator is therefore in the position to enforce connection policies and mediate address sharing between VMs.

These building blocks are used to provide high performance communication capabilities for applications in VMs.

3.2 Providing Protocol Functionality

We provide the functionality of TCP and UDP as follows.

Socket Interface Compatibility. Upon a socket entering the connected or bound state (see Fig. 1), we allocate and establish a circuit. For stream connections, the communication endpoint can decide to accept or decline a connection request, such that we stay in a pending state until the decision is received. This behavior maps well to the existing TCP socket interface which already includes such a pending state.

For datagram sockets we need to distinguish between dynamically shared addresses of VMs and fixed assignments of IP addresses or port numbers to VMs. With dynamically shared addresses the decision to bind an unbound UDP socket needs to be made by the switch operator, the only instance knowing whether the port number is already used. Since the UDP Socket interface does not have a pending state, the bind operation then needs to block until an answer from the switch operator is received. However with fixed address assignment, the VM can decide by itself, which enables a substantially faster response to the bind operation.

Payload Transmission. As elaborated in Sect. 2.2, during payload transmission inside a shared-memory system we only need to ensure proper flow control. To this end, we split the shared-memory of a circuit into two ring-buffers, one for each direction of communication, and a control area containing pairs of read and write pointers. The transmitter then detects a full receive buffer when the write pointer reaches the read pointer and defers further transmission until the read pointer advances. This resembles TCP's flow control, but with minimal propagation delays of receiver window sizes. Additional communication features of TCP, e.g., shutdown flags, reset flags, and urgent pointers are placed into the control area. To differentiate the single datagrams in a circuit – which the ring-buffer essentially maps to a stream – we prepend a small header to each datagram. For datagram services, instead of delaying transmissions we can drop incoming traffic if the receive buffer is full to mimic UDP socket behavior.

For communication with external hosts, virtual Ethernet demultiplexes incoming packets twice, the host OS selects the destination VM which then needs to select the destination socket for each incoming packet. Our approach removes a demultiplexing step in directly selecting the destination socket from the proxy stack in the host OS. By using the circuit as transmit and receive buffer for the proxy connection the proxy stack can synchronize the receiver window size with the available buffer space to ensure proper congestion and flow control between the VM and an external host.

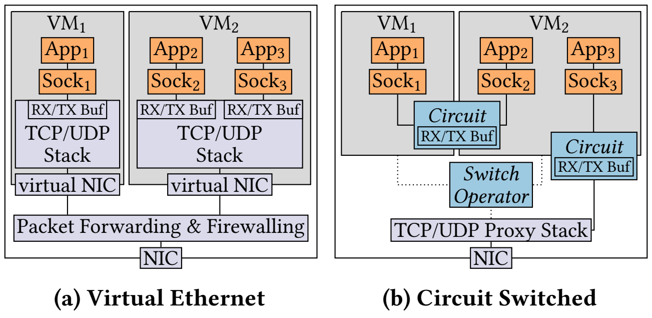

Figure 2: Comparison of the virtual Ethernet approach to our circuit-switched VM networks aprroach.

Zero-Copy Extension. We extend the socket interface by an optional mechanism for full zero-copy. Since each socket uses a separate circuit, the socket interface can be extended to map the memory of a single socket directly into user-space. Once the memory of a circuit is mapped into the address space of the application, the kernel is no longer needed to communicate on this socket except for interrupt handling. Instead it is the application's responsibility to follow the ring-buffer protocol. A modified application can therefore avoid a copy operation in directly reading payload from and writing payload to the shared memory. However, to avoid time-of-check to time-of-use race conditions, an application should not read the same payload twice, since the sending VM may have modified the memory in between both reads. Since every circuit has separate memory, zero-copy IO can be done separately for each socket and is optional to each application. The in-memory protocol remains unchanged when switching to zero-copy IO, therefore allowing an application to switch forth and back between the legacy socket interface and zero-copy IO at any time without notifying the communication partner.

Our approach resembles the functionality of state-of-the-art stream and datagram communications. In the following we elaborate on the qualitative benefits before we discuss our performance measurements.

4 QUALITATIVE BENEFITS

Besides improving the performance, circuit switched VM networks also provide a range of qualitative benefits.

Network Isolation. The execution of applications in VMs is often compared to executing them with containerization. From a network isolation standpoint, a container shields raw packet access from applications in only providing the socket interface, whereas a VM gets full access to the raw network on all layers. Raw packet access in VMs is considered a strong indicator for malicious behavior [6] and may be used to hide attack patterns from a network intrusion detection system (NIDS) through creating ambiguous and therefore hard to analyze TCP streams [7, 19]. Additionally, raw packet access may be used to gain an unfair share of the network bandwidth by using unfair congestion control [9].

Virtualization in the cloud [2] and virtualization to compartmentalize applications [21] involve VMs not trusted by the provider of the host and network. When completely removing raw packet access from VMs, none of this malicious behavior is possible. Benign applications may continue to behave as usual since we provide an interface to VMs which is similar to the interface provided by containerization.

Transparent Transport Protocol Switching. TCP does not provide optimal performance in e.g., data center networks or for mobile devices. The decoupling of VMs from packetization brings the opportunity to transparently choose a suitable transport protocol for each connection. The switch operator may therefore decide to use DCTCP [1] to hosts in the same data-center or MPFLex [18] towards a multi-homed wireless device without involving the VM in the decision. VMs may therefore benefit from improved performance over external networks without requiring software modifications.

Improved VM Deployment. New VM deployment schemes such as spawning a new VM upon a TCP connection request [14] and sharing of IP addresses as well as TCP ports between VMs are complicated in the traditional architecture by the placement of the TCP stack inside the VM. These schemes require additional TCP processing in the host OS with subsequent synchronization of TCP state between host and VM. With our approach such schemes come naturally, since we shift all TCP processing to the host OS.

High density VM deployments with up to thousands of VMs on a single machine [15] require small VM sizes. Especially for Unikernel VMs [12, 16] the TCP stack may present a significant share of memory. The Linux Implementation of our approach reduces the minimum VM size to run a single networked process by 17% from 48 MiB to 40 MiB. Circuit switched VM networks therefore increase the VM density.

Our approach provides several qualitative benefits. The next section continues with a performance evaluation.

5 EVALUATION

We evaluate our approach by means of goodput and response time benchmarks and a study on the applicability to existing applications. We describe the implementation and setup before we discuss the results.

Implementation. We implemented circuit switched VM networks for Linux as OS for both VM and host (also called dom0) on the Xen hypervisor.

To provide TCP and UDP access for applications in VMs, the VM kernel networking stack is replaced by an implementation which provides access to our circuits through the socket interface. Unmodified applications are enabled by our choice to reimplement the socket interface only in the Linux kernel without requiring any user-space modifications. We used a test-driven implementation approach, to ensure unmodified behavior of the socket interface. Our interface additionally provides means for zero-copy transmission.

Only the switch operator in the host OS is implemented purely in user-space for simplicity. The switch operator establishes control channels to VMs and connects circuits to the proxy stack through Linux kernel primitives providing Xen shared-memory and interrupt handling. For communication with external hosts, it connects each circuit to a regular Linux socket, which provides TCP and UDP anyway. We used this implementation for our performance benchmarks and our application compatibility study.

Setup. We performed all experiments on an Intel Xeon E5-4610 v4 CPU with 10 Cores, Hyper-Threading disabled, and 64 GiB of RAM. The communication with an external host was performed with a direct link over an Intel X710-T4 10 Gbit NIC. All circuit connections used ring-buffer sizes of 64 KiB. We used Xen virtual Ethernet [4] in the default configuration with a Linux bridge interface in the host OS, and additionally used direct virtual Ethernet between VMs to get a better comparison of circuits to virtual Ethernet. Measurements are shown with 99% confidence interval over 20 repetitions of goodput measurements and 200 repetitions of response time and connection setup time.

Goodput Measurements. To measure stream goodput we transmit data from a number of VMs in three scenarios to (1) an external host, (2) the host OS, (3) another VM on the same host. We varied the number of transmitting VMs from 1 to 128 and transmit a total of 1 GiB of payload. Each transmitting VM is connected to its own instance of a receiving application on a shared VM or host. We measure the time from transmitting the first byte on the first VM till receiving the acknowledgment for the last byte on the last VM.

For each scenario we employ three different approaches: First, the *legacy approach* establishes a TCP connection via the virtual Ethernet devices. Second, a legacy application uses the POSIX socket interface provided by our circuit switched approach (*circuit + legacy app*). Third, an application uses the zero-copy capabilities to directly access the shared buffer provided by our circuit switched approach (*circuit + zero-copy*).

Fig. 3 shows the results. In *scenario (1)* the goodput of our approach is limited by the 10 Gbit NIC, whereas virtual Ethernet underutilizes the NIC.

This bottleneck is removed in *scenario (2)*. However, in this scenario our approach still needs to use the TCP stack in the host OS, i.e., the data is transmitted over a circuit to the host OS and then packetized over the host's loopback interface. As depicted in the results, our approach not only speeds up the performance with a single VM but even benefits significantly from the opportunity to parallelize the computations of multiple VMs. With 32 VMs the circuit switched network with a legacy application achieves 4.2 times the throughput that can be achieved by TCP over virtual Ethernet. Interestingly, the highest throughput of our approach is reached with VM counts of 16-32, therefore having more VMs than the 10 CPU cores, whereas virtual Ethernet reaches the highest goodput int the range of 2-4 VMs. The zero-copy capabilities of our socket interface bring another increase of up to 7 Gbit/s in goodput. We attribute this to reduced processing in the VMs leaving more CPU time for the host OS to perform TCP processing.

We conclude that circuit switched VM networks are suitable for faster NICs, e.g., 40 Gbit, while virtual Ethernet barely reaches 10 Gbit. Furthermore, our approach scales well with an increasing number of VMs and benefits significantly from the opportunity to parallelize workload.

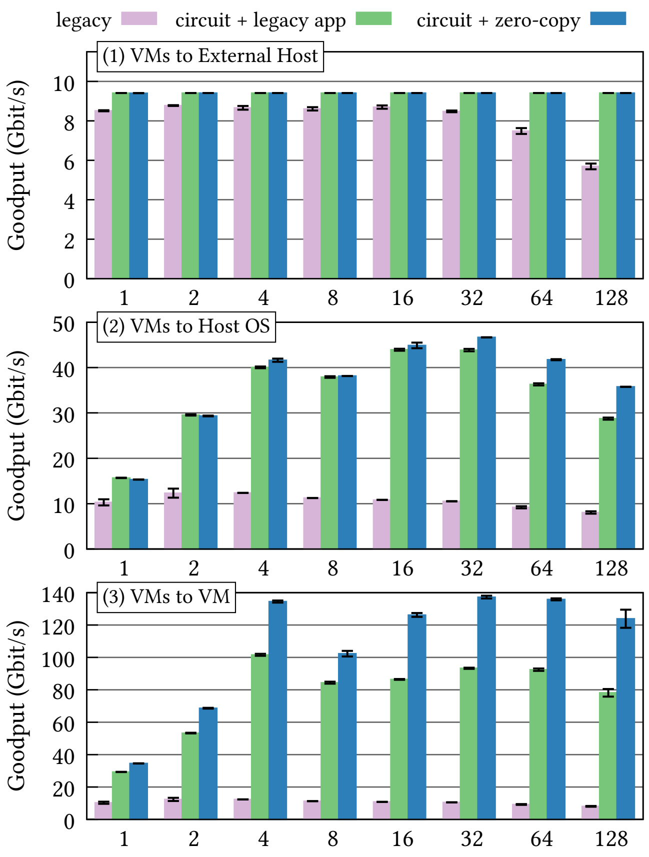

Figure 3: Goodput for different numbers of VMs.

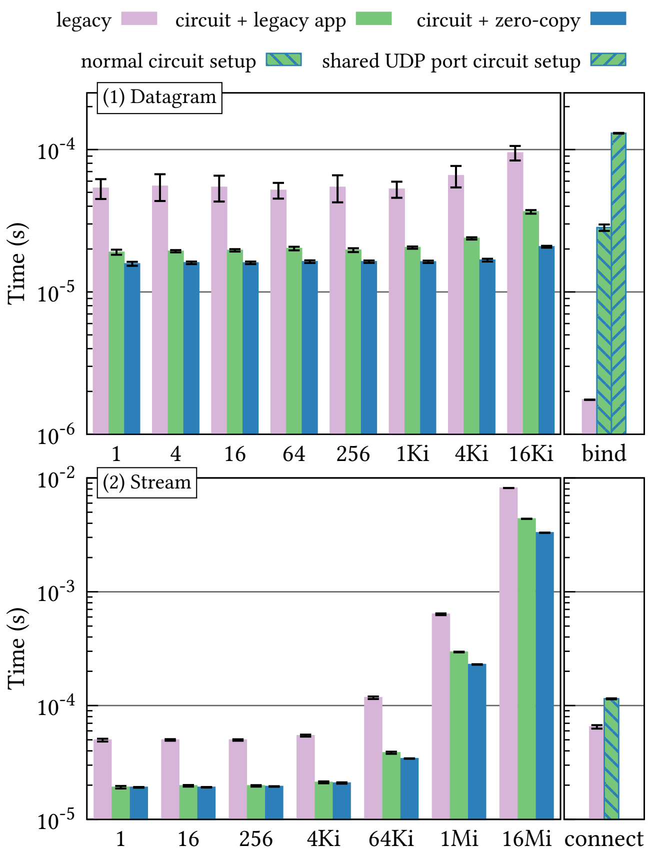

Figure 4: RTT for different response sizes (byte) and the response time of the bind and connect operation.

In *scenario (3)* (VM to VM communication) our approach can unleash its full potential as no TCP packetizing is required anymore, neither with zero-copy nor with legacy applications. On the contrary, virtual Ethernet still requires TCP connections to be established and achieves the same performance as in scenario (2). Circuit switched VM networks, on the other hand, achieve goodput of up to 100 Gbit/s even with an unmodified legacy application running in the VM. If the application uses the zero-copy capabilities, goodput easily exceeds 100 Gbit/s with 4 to 128 parallel VMs and reaches up to 137.2 Gbit/s. This is an improvement by a factor of 15.4 compared to the 10.5 Gbit/s achieved via virtual Ethernet.

We conclude that data transfers can be speeded up by more than an order of magnitude when great amounts of data are to be transmitted. To get a better understanding of the implication on small payloads, we investigate round-trip times (RTTs) in the following.

Response Times. To determine the influence of our approach on delay, we measured VM to VM RTTs and connection setup time for both (1) datagram and (2) stream sockets. All measured round-trips consisted of a single byte request followed by a response of varying size. Datagram responses are fragmented into payload sizes of at maximum 1472 bytes. Fig. 4 shows a decrease in RTT when using circuits for all response sizes and an additional decrease in RTT when performing zero-copy for all cases except stream responses with a size below 64 KiB. Datagram RTT decreases by at least 61% when using circuits with legacy sockets and decrease by at least an additional 16% when performing zero-copy with up to an additional decrease of 43% at 16 KiB. Stream RTT decreases by at least 45% when using circuits with legacy sockets and additionally up to 19% when performing zero-copy with a response size of at least 64 KiB.

Establishing a circuit takes more time than performing the TCP handshake with another VM or binding a datagram socket. Fig. 4 shows an increase in connecting a stream socket from 65.0 μs to 114.7 μs. When summing up connection time and RTT, stream circuits with large response sizes decrease delay for the first response. However, for request sizes below 64 KiB, stream circuits are only beneficial when performing at least 2 round-trips. For datagram circuits, we measured the time of two different variants of the `bind` operation, the normal case with returning before receiving a response from the operator, and the dynamic port sharing case with waiting for the switch operator to acknowledge circuit establishment. Binding a datagram socket increases from 1.8 μs to 28.3 μs under normal circumstances and to 130.2 μs when dynamically sharing UDP ports between VMs. Datagram circuits therefore under normal circumstances decrease response time already for the first round-trip, whereas with dynamic port sharing, the benefit is delayed until the second to fourth round-trip depending on response size. We suspect, that the high connection setup times are caused by our choice to implement the switch operator in user-space.

Circuit switched VM networks reduce delays in communicating. Zero-copy brings in most cases an additional decrease in RTTs. However the increase in connection setup time does not always lead to a decreased delay for the first round-trip.

Application Compatibility. Additionally, we investigate the compatibility of unmodified applications with our approach. Since we aimed to provide the original Linux socket interface, we expect any application to run out of the box. In fact, we did not experience any incompatibilities with any of the applications we tested. These applications included the web browser Firefox, the online game Quake 3, the Transmission BitTorrent client, the web server nginx, the DNS server BIND, the Tor anonymity client, the Mutt email client, and the command line tools openssh, git, aptitude, and wget.

Our evaluation shows an increased performance for most cases while providing existing applications with a compatible socket interface.

6 RELATED WORK

Offloading parts of TCP processing to the host has been proposed in different variants. Menon et al. [17] suggested providing TCP segmentation offloading to the VM virtual network interface. TCP segmentation offloading is part of current Xen virtual Ethernet and was enabled during our evaluation. The vSnoop [10] and vFlood [5] approaches, offload TCP acknowledgment generation and congestion control to the host. We go further in removing TCP processing when possible and moving it to the host when necessary.

Direct inter-VM communication has been proposed in different variants. XenSockets [26] deviate from the TCP stream semantics and use a different socket interface, thereby requiring modified applications. XenLoop [25] proposes direct virtual Ethernet between VMs which we included into our performance evaluation. XWAY [11] discovers collocation of two VMs on the same host to use direct shared-memory communication, but only for streams. Unlike our approach, none of these consider extending the benefits to the communication with external hosts or providing zero-copy IO.

Different proposals exist to change the VM networking architecture. NetVM [8] shares a single memory-area between the NIC and all VMs on a host, thereby allowing zero-copy VM service chains. Since each application gets full access to the network communication of all VMs, this approach is only applicable if all VMs are considered to be in the same trust domain. The ClickOS [16] Unikernel extends the netmap [20] kernel bypass mechanism to provide high packet throughput for Unikernel VMs and Linux. Both NetVM and ClickOS focus on network function virtualization which requires packet access, we, however, focus on VMs as network endpoints which allows us to provide per socket zero-copy IO without sacrificing socket isolation.

Using the existing TCP stack of the host OS for VMs exists in different variants. The socket-outsourcing approach [3] provides VMs with remote system-calls into the socket interface of the host OS, thereby heavily weakening the isolation between the VM and the host OS. The pvcalls [23] mechanism which is still under development for the Linux kernel, uses the TCP stack of a Linux host OS from a Xen VM. In comparison to pvcalls, circuit switched VM networks bring additional benefits through zero-copy IO, direct VM communication, and datagram circuits.

We propose to use separate shared-memory connection for each socket, in order to provide per socket zero-copy IO. We also propose to unify stream and datagram communication as well as direct VM and external host communication through a mechanism of separate circuits. To our knowledge, neither such socket zero-copy IO nor such a unified communication mechanism upon separate circuits has been previously proposed.

7 CONCLUSION

The processing of protocols like TCP in the Kernel of a VM is challenging for network performance. We therefore remove packetization, encapsulation, congestion control, and unnecessary copying of data whenever it is possible and move it to the host OS when still necessary. This is achieved through replacing the single virtual Ethernet link of each VM by a separate shared-memory connection for each application socket. Separate memory for each socket brings the possibility to perform memory mapped zero-copy IO without breaking network isolation between applications. Our approach reduces overhead not only for communication between VMs, but also when communicating with external hosts through using a proxy stack in the host OS.

Existing applications depend on the behavior of the socket interface which provides TCP and UDP access. We therefore provide our approach over an unmodified socket interface for legacy applications and extend the socket interface with zero-copy IO functionality for additional performance benefits available to modified applications.

Our approach provides the opportunity to remove raw packet access from VMs. Untrusted VMs can therefore be prevented from using the raw packet access to forge malicious packets. The decoupling of the VM from the packet network stack brings deployment benefits through smaller VM network stack sizes and the possibility to transparently choose a suitable transport protocol in the proxy stack.

We show significant performance improvement for both communication between VMs and when communicating with external hosts. The evaluation shows goodput increase up to a factor of 15.4 and RTT decrease up to 78%.

Circuit switched VM networks increase performance without sacrificing socket isolation and bring the possibility to better isolate the network from malicious VMs.

ACKNOWLEDGMENTS

This work has been performed in the context of the German Research Foundation (DFG) funded Collaborative Research Center (CRC) 1053 "MAKI - Multi-Mechanism-Adaptation for the Future Internet".

REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010.

Data Center TCP (DCTEP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM'10)*. ACM, 63–74. https://doi.org/10.1145/1851182.1851192

[2] Patrick Colp, Mihir Nanavati, Jun Zhu, William Aiello, George Coker, Tim Deegan, Peter Loscocco, and Andrew Warfield. 2011. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP'11)*. ACM, 189–202. https://doi.org/10.1145/2043556.2043575

[3] Hideki Eiraku, Yasushi Shinjo, Calton Pu, Younggyun Koh, and Kazuhiko Kato. 2009. Fast Networking with Socket-Outsourcing in Hosted Virtual Machine Environments. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC'09)*. ACM, 310–317. https://doi.org/10.1145/1529282.1529350

[4] Keir Fraser, Steven Hand, Rolf Neugebauer, Ian Pratt, Andrew Warfield, and Mark Williamson. 2004. Safe Hardware Access with the Xen Virtual Machine Monitor. In *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*.

[5] Sahan Gamage, Ardalan Kangarlou, Ramana Rao Kompella, and Dongyan Xu. 2011. Opportunistic Flooding to Improve TCP Transmit Performance in Virtualized Clouds. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC'11)*. ACM, 24:1–24:14. https://doi.org/10.1145/2038916.2038940

[6] Tal Garfinkel and Mendel Rosenblum. 2003. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2003)*. The Internet Society.

[7] Mark Handley, Vern Paxson, and Christian Kreibich. 2001. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *Proceedings of the 10th USENIX Security Symposium (Security'01)*. USENIX Association.

[8] Jinho Hwang, K.K. Ramakrishnan, and Timothy Wood. 2015. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. *IEEE Transactions on Network and Service Management* 12, 1 (Feb. 2015), 34–37. https://doi.org/10.1109/TNSM.2015.2401568

[9] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. 2017. Taking a Long Look at QUIC. In *Proceedings of the 2017 Internet Measurement Conference (IMC'17)*. ACM, 290–303. https://doi.org/10.1145/3131365.3131368

[10] Ardalan Kangarlou, Sahan Gamage, Ramana Rao Kompella, and Dongyan Xu. 2010. vSnoop: Improving TCP Throughput in Virtualized Environments via Acknowledgement Offload. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*. IEEE, 1–11. https://doi.org/10.1109/SC.2010.57

[11] Kangho Kim, Cheiyol Kim, Sung-In Jung, Hyun-Sup Shin, and Jin-Soo Kim. 2008. Inter-domain 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)*. ACM, 11–20. https://doi.org/10.1145/1346256.1346259

[12] Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. 2014. OSv—Optimizing the Operating System for Virtual Machines. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX Association, 61–72.

[13] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblar, and L. Jones. 1996. *SOCKS Protocol Version 5*. RFC 1928.

[14] Anil Madhavapeddy, Thomas Leonard, Magnus Skjegstad, Thomas Gazagnaire, David Sheets, Dave Scott, Richard Mortier, Amir Chaudhry, Balraj Singh, Jon Ludlam, Jon Crowcroft, and Ian Leslie. 2015. Jitsu: Just-In-Time Summoning of Unikernels. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, 559–573.

[15] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17)*. ACM, 218–233. https://doi.org/10.1145/3132747.3132763

[16] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the Art of Network Function Virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, 459–473.

[17] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. 2006. Optimizing Network Virtualization in Xen. In *Proceedings of the 2006 USENIX Annual Technical Conference (USENIX ATC'06)*. USENIX Association, 15–28.

[18] Ashkan Nikravesh, Yihua Guo, Feng Quian, Z. Morley Mao, and Subhabrata Sen. 2016. An In-depth Understanding of Multipath TCP on Mobile Devices: Measurement and System Desgin. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking (MobiCom'16)*. ACM, 189–201. https://doi.org/10.1145/2973750.2973769

[19] Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer Networks* 31, 23-24 (Dec. 1999), 2435–2463. https://doi.org/10.1016/S1389-1286(99)00112-7

[20] Luigi Rizzo. 2012. netmap: a novel framework for fast packet I/O. In *Proceedings of the 2012 USENIX Annual Technical Conference (ATC'12)*. USENIX Association.

[21] Joanna Rutkowska and Rafal Wojtczuk. 2010. *The Qubes OS Architecture*. Technical Report.

[22] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Cácares, and Nigel Davis. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 14–23. https://doi.org/10.1109/MPRV.2009.82

[23] Stefano Stabellini. 2017. introduce the Xen PV Calls frontend. https://lwn.net/Articles/728622/

[24] Guohui Wang and T. S. Eugene Ng. 2010. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *Proceedings of the IEEE INFOCOM 2010 (INFOCOM 2010)*. IEEE. https://doi.org/10.1109/INFCOM.2010.5461931

[25] Jian Wang, Kwame-Lante Wright, and Kartik Gopalan. 2008. XenLoop: A Transparent High Performance Inter-VM Network Loopback. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC'08)*. ACM, 109–118. https://doi.org/10.1145/1383422.1383437

[26] Xiaolan Zhang, Suzanne McIntosh, Pankaj Rohatgi, and John Linwood Griffin. 2007. XenSocket: A High-Throughput Interdomain Transport for Virtual Machines. In *Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2007)*. Springer. https://doi.org/10.1007/978-3-540-76778-7_10