VNET/P: Bridging the Cloud and High Performance Computing Through Fast Overlay Networking

Lei Xia* Zheng Cui§ John Lange‡
Yuan Tang† Peter Dinda* Patrick Bridges§

* Dept. of Electrical Engineering and Computer Science, Northwestern University
{lxia,pdinda}@northwestern.edu

§ Dept. of Computer Science, University of New Mexico
{cuizheng,bridges}@cs.unm.edu

‡ Dept. of Computer Science, University of Pittsburgh
jacklange@cs.pitt.edu

† School of Computer Science and Engineering, University of Electronic Science and Technology of China
ytang@uestc.edu.cn

ABSTRACT

It is now possible to allow VMs hosting HPC applications to seamlessly bridge distributed cloud resources and tightly-coupled supercomputing and cluster resources. However, to achieve the application performance that the tightly-coupled resources are capable of, it is important that the overlay network not introduce significant overhead relative to the native hardware, which is not the case for current user-level tools, including our own existing VNET/U system. In response, we describe the design, implementation, and evaluation of a layer 2 virtual networking system that has negligible latency and bandwidth overheads in 1–10 Gbps networks. Our system, VNET/P, is directly embedded into our publicly available Palacios virtual machine monitor (VMM). VNET/P achieves native performance on 1 Gbps Ethernet networks and very high performance on 10 Gbps Ethernet networks and InfiniBand. The NAS benchmarks generally achieve over 95% of their native performance on both 1 and 10 Gbps. These results suggest it is feasible to extend a software-based overlay network designed for computing at wide-area scales into tightly-coupled environments.

Categories and Subject Descriptors

D.4.4 [**Software**]: OPERATING SYSTEMS

General Terms

Design, Measurement, Performance, Experimentation

Keywords

Overlay Networks, Virtualization, HPC, Scalability

This project is made possible by support from the United States National Science Foundation (NSF) via grants CNS-0709168 and CNS-0707365, and by the Department of Energy (DOE) via grant DE-SC0005343. Yuan Tang's visiting scholar position at Northwestern was supported by the China Scholarship Council.

HPDC'12, June 18–22, 2012, Delft, The Netherlands.

1. INTRODUCTION

Cloud computing in the "infrastructure as a service" (IaaS) model has the potential to provide economical and effective on-demand resources for high performance computing. In this model, an application is mapped into a collection of virtual machines (VMs) that are instantiated as needed, and at the scale needed. Indeed, for loosely-coupled applications, this concept has readily moved from research [6, 38] to practice [33]. As we describe in Section 3, such systems can also be adaptive, autonomically selecting appropriate mappings of virtual components to physical components to maximize application performance or other objectives. However, *tightly-coupled* scalable high performance computing (HPC) applications currently remain the purview of resources such as clusters and supercomputers. We seek to extend the adaptive IaaS cloud computing model into these regimes, allowing an application to dynamically span both kinds of environments.

The current limitation of cloud computing systems to *loosely-coupled* applications is not due to machine virtualization limitations. Current virtual machine monitors (VMMs) and other virtualization mechanisms present negligible overhead for CPU and memory intensive workloads [14, 31]. With VMM-bypass [29] or self-virtualizing devices [35] the overhead for direct access to network devices can also be made negligible.

Considerable effort has also gone into achieving low-overhead network virtualization and traffic segregation within an individual data center through extensions or changes to the network hardware layer [32, 9, 20]. While these tools strive to provide uniform performance across a cloud data center (a critical feature for many HPC applications), they do not provide the same features once an application has migrated outside the local data center, or spans multiple data centers, or involves HPC resources. Furthermore, they lack compatibility with the more specialized interconnects present on most HPC systems. Beyond the need to support our envisioned computing model across today's and tomorrow's tightly-coupled HPC environments, we note that data center network design and cluster/supercomputer network design seems to be converging [1, 10]. This suggests that future data centers deployed for general purpose cloud computing will become an increasingly better fit for tightly-coupled parallel applications, and therefore such environments could potentially also benefit.

The current limiting factor in the adaptive cloud- and HPC-spanning model described above for tightly-coupled applications is the performance of the virtual networking system. Current adaptive cloud computing systems use software-based overlay networks to carry inter-VM traffic. For example, our VNET/U system, which is described in more detail later, combines a simple networking abstraction within the VMs with location-independence, hardware-independence, and traffic control. Specifically, it exposes a layer 2 abstraction that lets the user treat his VMs as being on a simple LAN, while allowing the VMs to be migrated seamlessly across resources by routing their traffic through the overlay. By controlling the overlay, the cloud provider or adaptation agent can control the bandwidth and the paths between VMs over which traffic flows. Such systems [42, 37] and others that expose different abstractions to the VMs [47] have been under continuous research and development for several years. Current virtual networking systems have sufficiently low overhead to effectively host loosely-coupled scalable applications [5], but their performance is insufficient for tightly-coupled applications [34].

In response to this limitation, we have designed, implemented, and evaluated VNET/P, which shares its model and vision with VNET/U, but is designed to achieve near-native performance in the 1 Gbps and 10 Gbps switched networks common in clusters today, and pave the way for even faster networks, such as InfiniBand, in the future. VNET/U is presented in more detail in Section 3.

VNET/P is implemented in the context of our publicly available, open source Palacios VMM [25], which is in part designed to support virtualized supercomputing. A detailed description of VNET/P's design and implementation is given in Section 4. As a part of Palacios, VNET/P is publicly available. VNET/P could be implemented in other VMMs, and as such provides a proof-of-concept that overlay-based virtual networking for VMs, with performance overheads low enough to be inconsequential even in a tightly-coupled computing environment, is clearly possible.

The performance evaluation of VNET/P (Section 5) shows that it is able to achieve native bandwidth on 1 Gbps Ethernet with a small increase in latency, and very high bandwidth on 10 Gbps Ethernet with a similar, small latency increase. We also demonstrate in Section 6 that VNET/P can effectively support running Ethernet-based networked programs on non-Ethernet HPC communication device, specifically InfiniBand NICs. On 10 Gbps hardware, the kernel-level VNET/P system provides on average 10 times more bandwidth and 7 times less latency than the user-level VNET/U system can.

Our contributions are as follows:

- We articulate the benefits of extending virtual networking for VMs down to clusters and supercomputers with high performance networks. These benefits are also applicable to data centers that support IaaS cloud computing.
- We describe the design and implementation of a virtual networking system, VNET/P, that does so. The design could be applied to other VMMs and virtual network systems.
- We evaluate VNET/P, finding that it provides performance with negligible overheads on 1 Gbps Ethernet networks, and manageable overheads on 10 Gbps Ethernet networks. VNET/P generally has little impact on performance for the NAS benchmarks.
- We describe how VNET/P also provides its abstraction on top of InfiniBand hardware, allowing guests to exploit such hardware without any special drivers or an InfiniBand stack.

Through the use of low-overhead overlay-based virtual networking in high-bandwidth, low-latency environments such as current clusters and supercomputers, and future data centers, we seek to make it practical to use virtual networking at all times, even when running tightly-coupled applications on such high-end environments. This would allow us to seamlessly and *practically* extend the already highly effective adaptive virtualization-based IaaS cloud computing model to such environments.

2. RELATED WORK

VNET/P is related to NIC virtualization, overlays, and virtual networks, as we describe below.

NIC virtualization: There is a wide range of work on providing VMs with fast access to networking hardware, where no overlay is involved. For example, VMware and Xen support either an emulated register-level interface [41] or a paravirtualized interface to guest operating system [30]. While purely software-based virtualized network interface has high overhead, many techniques have been proposed to support simultaneous, direct-access network I/O. For example, some work [29, 35] has demonstrated the use of self-virtualized network hardware that allows direct guest access, thus provides high performance to untrusted guests. Willmann et al have developed a software approach that also supports concurrent, direct network access by untrusted guest operating systems [39]. In addition, VPIO [48] can be applied on network virtualization to allow virtual passthrough I/O on non-self-virtualized hardware. In contrast with such work, VNET/P provides fast access to an overlay network, which includes encapsulation and routing.

Overlay networks: Overlay networks implement extended network functionality on top of physical infrastructure, for example to provide resilient routing (e.g, [2]), multicast (e.g. [13]), and distributed data structures (e.g., [40]) without any cooperation from the network core; overlay networks use end-systems to provide their functionality. VNET is an example of a specific class of overlay networks, namely virtual networks, discussed next.

Virtual networking: Virtual networking systems provide a service model that is compatible with an existing layer 2 or 3 networking standard. Examples include VIOLIN [17], ViNe [45], VINI [3], SoftUDC VNET [19], OCALA [18] and WoW [8]. Like VNET, VIOLIN, SoftUDC, and WoW are specifically designed for use with virtual machines. Of these, VIOLIN is closest to VNET (and contemporaneous with VNET/U), in that it allows for the dynamic setup of an arbitrary private layer 2 and layer 3 virtual network among VMs. The key contribution of VNET/P is to show that this model can be made to work with minimal overhead even in extremely low latency, high bandwidth environments.

Connections: VNET/P could itself leverage some of the related work described above. For example, effective NIC virtualization might allow us to push encapsulation directly into the guest, or to accelerate encapsulation via a split scatter/gather map. Mapping unencapsulated links to VLANs would enhance performance on environments that support them. There are many options for implementing virtual networking and the appropriate choice depends on the hardware and network policies of the target environment. In VNET/P, we make the choice of minimizing these dependencies.

3. VNET MODEL AND VNET/U

The VNET model was originally designed to support adaptive computing on distributed virtualized computing resources within the Virtuoso system [4], and in particular to support the adaptive execution of a distributed or parallel computation executing in a collection of VMs potentially spread across multiple providers or supercomputing sites. The key requirements, which also hold for the present paper, were as follows.

- VNET would make within-VM network configuration the sole responsibility of the VM owner.
- VNET would provide location independence to VMs, allowing them to be migrated between networks and from site to site, while maintaining their connectivity, without requiring any within-VM configuration changes.
- VNET would provide hardware independence to VMs, allowing them to use diverse networking hardware without requiring the installation of specialized software.

- VNET would provide minimal overhead, compared to native networking, in the contexts in which it is used.

The VNET model meets these requirements by carrying the user's VMs' traffic via a configurable overlay network. The overlay presents a simple layer 2 networking abstraction: a user's VMs appear to be attached to the user's local area Ethernet network, regardless of their actual locations or the complexity of the VNET topology/properties. Further information about the model can be found elsewhere [42].

The VNET overlay is dynamically reconfigurable, and can act as a locus of activity for an an adaptive system such as Virtuoso. Focusing on parallel and distributed applications running in loosely-coupled virtualized distributed environments e.g., "IaaS Clouds", we demonstrated that the VNET "layer" can be effectively used to: (1) monitor application communication and computation behavior [11]), (2) monitor underlying network behavior [12], (3) formulate performance optimization problems [44], (4) address such problems through VM migration and overlay network control [43], scheduling [27, 28], network reservations [26], and network service interposition [22].

The VNET/P system described in this paper is compatible with, and compared to, our previous VNET implementation, VNET/U. Both support a dynamically configurable general overlay topology with dynamically configurable routing on a per MAC address basis. The topology and routing configuration is subject to global or distributed control (for example, by the VADAPT [43]) part of Virtuoso. The overlay carries Ethernet packets encapsulated in UDP packets, TCP streams with and without SSL encryption, TOR privacy-preserving streams, and others. Because Ethernet packets are used, the VNET abstraction can also easily interface directly with most commodity network devices, including virtual NICs exposed by VMMs in the host, and with fast virtual devices (e.g., Linux virtio network devices) in guests.

While VNET/P is implemented within the VMM, VNET/U is implemented as a user-level system. As a user-level system, it readily interfaces with VMMs such as VMware Server and Xen, and requires no host changes to be used, making it very easy for a provider to bring it up on a new machine. Further, it is easy to bring up VNET daemons when and where needed to act as proxies or waypoints. A VNET daemon has a control port which speaks a control language for dynamic configuration. A collection of tools allow for the wholesale construction and teardown of VNET topologies, as well as dynamic adaptation of the topology and forwarding rules to the observed traffic and conditions on the underlying network.

The last reported measurement of VNET/U showed it achieving 21.5 MB/s (172 Mbps) with a 1 ms latency overhead communicating between Linux 2.6 VMs running in VMware Server GSX 2.5 on machines with dual 2.0 GHz Xeon processors [22]. A current measurement, described in Section 5, shows 71 MB/s with a 0.88 ms latency. VNET/U's speeds are sufficient for its purpose in providing virtual networking for wide-area and/or loosely-coupled distributed computing. They are not, however, sufficient for use within a cluster at gigabit or greater speeds. Making this basic VM-to-VM path competitive with hardware is the focus of this paper. VNET/U is fundamentally limited by the kernel/user space transitions needed to handle a guest's packet send or receive. In VNET/P, we move VNET directly into the VMM to avoid such transitions.

4. DESIGN AND IMPLEMENTATION

We now describe how VNET/P has been architected and implemented in the context of Palacios as embedded in a Linux host. Section 6 describes how VNET/P is implemented in the context of a Kitten embedding. The nature of the embedding affects VNET/P

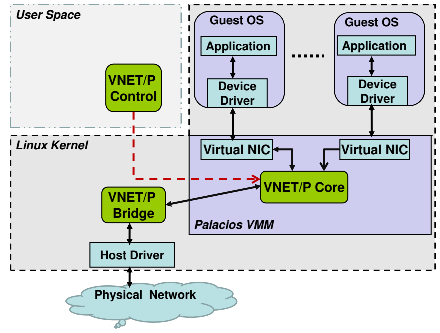


Figure 1: VNET/P architecture

primarily in how it interfaces to the underlying networking hardware and networking stack. In the Linux embedding, this interface is accomplished directly in the Linux kernel. In the Kitten embedding, the interface is done via a service VM.

4.1 Palacios VMM

VNET/P is implemented in the context of our Palacios VMM. Palacios is an OS-independent, open source, BSD-licensed, publicly available embeddable VMM designed as part of the V3VEE project (`http://v3vee.org`). The V3VEE project is a collaborative community resource development project involving Northwestern University, the University of New Mexico, Sandia National Labs, and Oak Ridge National Lab. Detailed information about Palacios can be found elsewhere [25, 23]. Palacios is capable of virtualizing large scale (4096+ nodes) with $< 5\%$ overheads [24]. Palacios's OS-agnostic design allows it to be embedded into a wide range of different OS architectures.

4.2 Architecture

Figure 1 shows the overall architecture of VNET/P, and illustrates the operation of VNET/P in the context of the Palacios VMM embedded in a Linux host. In this architecture, *guests* run in *application VMs*. Off-the-shelf guests are fully supported. Each application VM provides a virtual (Ethernet) NIC to its guest. For high performance applications, as in this paper, the virtual NIC conforms to the virtio interface, but several virtual NICs with hardware interfaces are also available in Palacios. The virtual NIC conveys Ethernet packets between the application VM and the Palacios VMM. Using the virtio virtual NIC, one or more packets can be conveyed from an application VM to Palacios with a single VM exit, and from Palacios to the application VM with a single VM exit+entry.

The *VNET/P core* is the component of VNET/P that is directly embedded into the Palacios VMM. It is responsible for routing Ethernet packets between virtual NICs on the machine and between this machine and remote VNET on other machines. The VNET/P core's routing rules are dynamically configurable, through the control interface by the utilities that can be run in user space.

The VNET/P core also provides an expanded interface that the control utilities can use to configure and manage VNET/P. The *VNET/P control* component uses this interface to do so. It in turn acts as a daemon that exposes a TCP control port that uses the same configuration language as VNET/U. Between compatible encapsulation and compatible control, the intent is that VNET/P and VNET/U be interoperable, with VNET/P providing the "fast path".

To exchange packets with a remote machine, the VNET/P core uses a *VNET/P bridge* to communicate with the physical network.

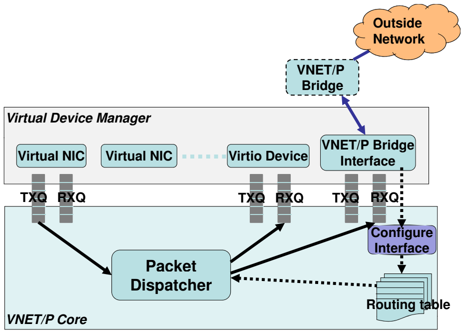


Figure 2: VNET/P core's internal logic.

The VNET/P bridge runs as a kernel module in the host kernel and uses the host's networking facilities to interact with physical network devices and with the host's networking stack. An additional responsibility of the bridge is to provide encapsulation. For performance reasons, we use UDP encapsulation, in a form compatible with that used in VNET/U. TCP encapsulation is also supported. The bridge selectively performs UDP or TCP encapsulation for packets destined for remote machines, but can also deliver an Ethernet packet without encapsulation. In our performance evaluation, we consider only encapsulated traffic.

The VNET/P core consists of approximately 2500 lines of C in Palacios, while the VNET/P bridge consists of about 2000 lines of C comprising a Linux kernel module. VNET/P is available via the V3VEE project's public git repository, as part of the "devel" branch of the Palacios VMM.

4.3 VNET/P core

The VNET/P core is primarily responsible for routing, and dispatching raw Ethernet packets. It intercepts all Ethernet packets from virtual NICs that are associated with VNET/P, and forwards them either to VMs on the same host machine or to the outside network through the VNET/P bridge. Each packet is routed based on its source and destination MAC addresses. The internal processing logic of the VNET/P core is illustrated in Figure 2.

Routing: To route Ethernet packets, VNET/P maintains routing tables indexed by source and destination MAC addresses. Although this table structure only provides linear time lookups, a hash table-based routing cache is layered on top of the table, and the common case is for lookups to hit in the cache and thus be serviced in constant time.

A routing table entry maps to a destination, which is either a *link* or an *interface*. A link is an overlay destination—it is the next UDP/IP-level (i.e., IP address and port) destination of the packet, on some other machine. A special link corresponds to the local network. The local network destination is usually used at the "exit/entry point" where the VNET overlay is attached to the user's physical LAN. A packet routed via a link is delivered to another VNET/P core, a VNET/U daemon, or the local network. An interface is a local destination for the packet, corresponding to some virtual NIC.

For an interface destination, the VNET/P core directly delivers the packet to the relevant virtual NIC. For a link destination, it injects the packet into the VNET/P bridge along with the destination link identifier. The VNET/P bridge demultiplexes based on the link and either encapsulates the packet and sends it via the corresponding UDP or TCP socket, or sends it directly as a raw packet to the local network.

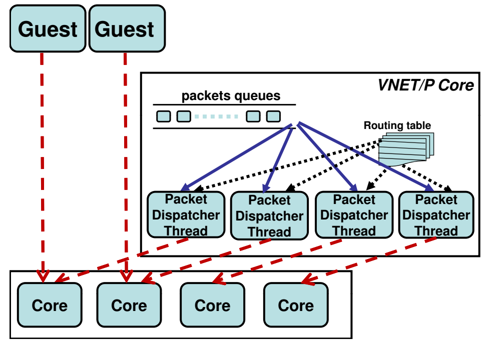


Figure 3: VNET/P running on a multicore system. The selection of how many, and which cores to use for packet dispatcher threads is made dynamically.

Packet processing: Packet forwarding in the VNET/P core is conducted by *packet dispatchers*. A packet dispatcher interacts with each virtual NIC to forward packets in one of two modes: *guest-driven mode* or *VMM-driven mode*.

The purpose of guest-driven mode is to minimize latency for small messages in a parallel application. For example, a barrier operation would be best served with guest-driven mode. In the guest-driven mode, the packet dispatcher is invoked when the guest's interaction with the NIC explicitly causes an exit. For example, the guest might queue a packet on its virtual NIC and then cause an exit to notify the VMM that a packet is ready. In guest-driven mode, a packet dispatcher runs at this point. Similarly, on receive, a packet dispatcher queues the packet to the device and then immediately notifies the device.

The purpose of VMM-driven mode is to maximize throughput for bulk data transfer in a parallel application. Unlike guest-driven mode, VMM-driven mode tries to handle multiple packets per VM exit. It does this by having VMM poll the virtual NIC. The NIC is polled in two ways. First, it is polled, and a packet dispatcher is run, if needed, in the context of the current VM exit (which is unrelated to the NIC). Even if exits are infrequent, the polling and dispatch will still make progress during the handling of timer interrupt exits.

The second manner in which the NIC can be polled is in the context of a packet dispatcher running in a kernel thread inside the VMM context, as shown in Figure 3. The packet dispatcher thread can be instantiated multiple times, with these threads running on different cores in the machine. If a packet dispatcher thread decides that a virtual NIC queue is full, it forces the NIC's VM to handle it by doing a cross-core IPI to force the core on which the VM is running to exit. The exit handler then does the needed event injection. Using this approach, it is possible, to dynamically employ idle processor cores to increase packet forwarding bandwidth.

Influenced by Sidecore [21], an additional optimization we developed was to offload in-VMM VNET/P processing, beyond packet dispatch, to an unused core or cores, thus making it possible for the guest VM to have full use of its cores (minus the exit/entry costs when packets are actually handed to/from it).

VNET/P switches between these two modes dynamically depending on the arrival rate of packets destined to or from the virtual NIC. For low rate, it enables guest-driven mode to reduce the single packet latency. On the other hand, with a high arrival rate it switches to VMM-driven mode to increase throughput. Specifically, the VMM detects whether the system is experiencing a high exit rate due to virtual NIC accesses. It recalculates the rate period-ically. If the rate is high enough when the guest transmits packets, then VNET/P switches the virtual NIC associated with that guest from guest-driven mode to VMM-driven mode. In other hand, if the rate drops low from the last recalculate period, it switches back from VMM-driven to guest-driven mode.

For a 1 Gbps network, guest-driven mode is sufficient to allow VNET/P to achieve the full native throughput. On a 10 Gbps network, VMM-driven mode is essential to move packets through the VNET/P core with near-native throughput.

4.4 Virtual NICs

VNET/P is designed to be able to support any virtual Ethernet NIC device. A virtual NIC must, however, register itself with VNET/P before it can be used. This is done during the initialization of the virtual NIC at VM configuration time. The registration provides additional callback functions for packet transmission, transmit queue polling, and packet reception. These functions essentially allow the NIC to use VNET/P as its backend, instead of using an actual hardware device driver backend.

Linux virtio virtual NIC: Virtio [36], which was recently developed for the Linux kernel, provides an efficient abstraction for VMMs. A common set of virtio device drivers are now included as standard in the Linux kernel. To maximize performance, our performance evaluation configured the application VM with Palacios's virtio-compatible virtual NIC, using the default Linux virtio network driver.

MTU: The maximum transmission unit (MTU) of a networking layer is the size of the largest protocol data unit that the layer can pass onwards. A larger MTU improves throughput because each packet carries more user data while protocol headers have a fixed size. A larger MTU also means that fewer packets need to be processed to transfer a given amount of data. Where per-packet processing costs are significant, larger MTUs are distinctly preferable. Because VNET/P adds to the per-packet processing cost, supporting large MTUs is helpful.

VNET/P presents an Ethernet abstraction to the application VM. The most common Ethernet MTU is 1500 bytes. However, 1 Gbit and 10 Gbit Ethernet can also use "jumbo frames", with an MTU of 9000 bytes. Other networking technologies support even larger MTUs. To leverage the large MTUs of underlying physical NICs, VNET/P itself supports MTU sizes of up to 64 KB.[1] The application OS can determine the virtual NIC's MTU and then transmit/receive accordingly. VNET/P advertises the appropriate MTU.

The MTU used by virtual NIC can result in encapsulated VNET/P packets that exceed the MTU of the underlying physical network. In this case, fragmentation has to occur, either in the VNET/P bridge or in the host NIC (via TCP Segmentation Offloading (TSO)). Fragmentation and reassembly is handled by VNET/P and is totally transparent to the application VM. However, performance will suffer when significant fragmentation occurs. Thus it is important that the application VM's device driver select an MTU carefully, and recognize that the desirable MTU may change over time, for example after a migration to a different host. In Section 5, we analyze throughput using different MTUs.

4.5 VNET/P Bridge

The VNET/P bridge functions as a network bridge to direct packets between the VNET/P core and the physical network through the host NIC. It operates based on the routing decisions made by the VNET/P core which are passed along with the packets to be forwarded. It is implemented as a kernel module running in the host.

When the VNET/P core hands a packet and routing directive up to the bridge, one of two transmission modes will occur, depending on the destination. In a *direct send*, the Ethernet packet is directly sent. This is common for when a packet is exiting a VNET overlay and entering the physical network, as typically happens on the user's network. It may also be useful when all VMs will remain on a common layer 2 network for their lifetime. In an *encapsulated send* the packet is encapsulated in a UDP packet and the UDP packet is sent to the directed destination IP address and port. This is the common case for traversing a VNET overlay link. Similarly, for packet reception, the bridge uses two modes, simultaneously. In a *direct receive* the host NIC is run in promiscuous mode, and packets with destination MAC addresses corresponding to those requested by the VNET/P core are handed over to it. This is used in conjunction with direct send. In an *encapsulated receive* UDP packets bound for the common VNET link port are disassembled and their encapsulated Ethernet packets are delivered to the VNET/P core. This is used in conjunction with encapsulated send. Our performance evaluation focuses solely on encapsulated send and receive.

4.6 Control

The VNET/P control component allows for remote and local configuration of links, interfaces, and routing rules so that an overlay can be constructed and changed over time. VNET/U already has user-level tools to support VNET, and, as we described in Section 3, a range of work already exists on the configuration, monitoring, and control of a VNET overlay. In VNET/P, we reuse these tools as much as possible by having the user-space view of VNET/P conform closely to that of VNET/U. The *VNET/P configuration console* allows for local control to be provided from a file, or remote control via TCP-connected VNET/U clients (such as tools that automatically configure a topology that is appropriate for the given communication pattern among a set of VMs [43]). In both cases, the VNET/P control component is also responsible for validity checking before it transfers the new configuration to the VNET/P core.

4.7 Performance-critical data paths and flows

Figure 4 depicts how the components previously described operate during packet transmission and reception. These are the performance critical data paths and flows within VNET/P, assuming that virtio virtual NICs (Section 4.4) are used. The boxed regions of the figure indicate steps introduced by virtualization, both within the VMM and within the host OS kernel. There are also additional overheads involved in the VM exit handling for I/O port reads and writes and for interrupt injection.

Transmission: The guest OS in the VM includes the device driver for the virtual NIC. The driver initiates packet transmission by writing to a specific virtual I/O port after it puts the packet into the NIC's shared ring buffer (TXQ). The I/O port write causes an exit that gives control to the virtual NIC I/O handler in Palacios. The handler reads the packet from the buffer and writes it to VNET/P packet dispatcher. The dispatcher does a routing table lookup to determine the packet's destination. For a packet destined for a VM on some other host, the packet dispatcher puts the packet into the receive buffer of the VNET/P bridge and notify it. Meanwhile, VNET/P bridge fetches the packet from the receive buffer, determines its destination VNET/P bridge, encapsulates the packet, and transmits it to the physical network via the host's NIC.

Note that while the packet is handed off multiple times, it is copied only once inside the VMM, from the send buffer (TXQ) of the receive buffer of the VNET/P bridge. Also note that while the above description, and the diagram suggest sequentiality, packet

[1] This may be expanded in the future. Currently, it has been sized to support the largest possible IPv4 packet size.

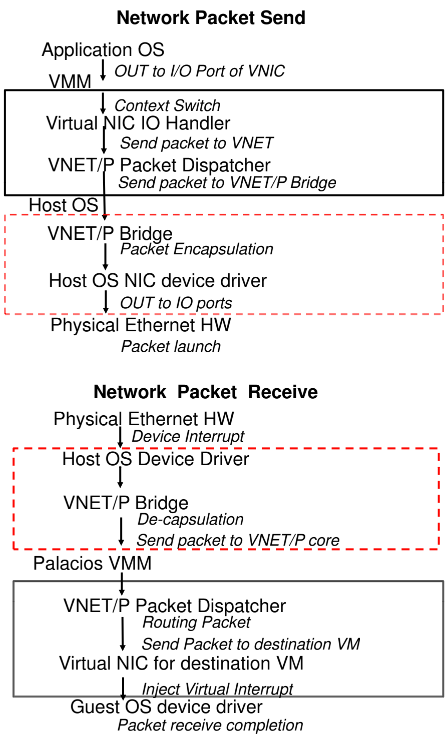


Figure 4: Performance-critical data paths and flows for packet transmission and reception. Solid boxed steps and components occur within the VMM itself, while dashed boxed steps and components occur in the host OS.

dispatch can occur on a separate kernel thread running on a separate core, and the VNET/P bridge itself introduces additional concurrency. From the guest's perspective, the I/O port write that initiated transmission returns essentially within a VM exit/entry time.

Reception: The path for packet reception is essentially symmetric to that of transmission. The host NIC in the host machine receives a packet using its standard driver and delivers it to the VNET/P bridge. The bridge unencapsulates the packet and sends the payload (the raw Ethernet packet) to the VNET/P core. The packet dispatcher in VNET/P core determines its destination VM and puts the packet into the receive buffer (RXQ) of its virtual NIC.

Similar to transmission, there is considerably concurrency in the reception process. In particular, packet dispatch can occur in parallel with the reception of the next packet.

5. PERFORMANCE EVALUATION

The purpose of our performance evaluation is to determine how close VNET/P comes to native throughput and latency in the most demanding (lowest latency, highest throughput) hardware environments. We consider communication between two machines whose NICs are directly connected in most of our detailed benchmarks. In the virtualized configuration the guests and performance testing tools run on top of Palacios with VNET/P carrying all traffic between them using encapsulation. In the native configuration, the same guest environments run directly on the hardware.

Our evaluation of communication performance in this environment occurs at three levels. First, we benchmark the TCP and UDP bandwidth and latency. Second, we benchmark MPI using a widely used benchmark. Finally, we evaluated the performance of the HPCC and NAS application benchmarks in a cluster to see how VNET/P's impact on the performance and scalability of parallel applications.

5.1 Testbed and configurations

Most of our microbenchmark tests are focused on the end-to-end performance of VNET/P. Therefore our testbed consists of two physical machines, which we call host machines. Each machine has a quadcore 2.4 GHz X3430 Intel Xeon(tm) processor, 8 GB RAM, a Broadcom NetXtreme II 1 Gbps Ethernet NIC (1000BASE-T), and a NetEffect NE020 10 Gbps Ethernet fiber optic NIC (10GBASE-SR) in a PCI-e slot. The Ethernet NICs of these machines are directly connected with twisted pair and fiber patch cables.

All microbenchmarks included in the performance section are run in the testbed described above. The HPCC and NAS application benchmarks are run on a 6-node test cluster described in Section 5.4.

We considered the following two software configurations:

- *Native:* In the native configuration, neither Palacios nor VNET/P is used. A minimal BusyBox-based Linux environment based on an unmodified 2.6.30 kernel runs directly on the host machines. We refer to the 1 and 10 Gbps results in this configuration as *Native-1G* and *Native-10G*, respectively.
- *VNET/P:* The VNET/P configuration corresponds to the architectural diagram given in Figure 1, with a single guest VM running on Palacios. The guest VM is configured with one virtio network device, 2 cores, and 1 GB of RAM. The guest VM runs a minimal BusyBox-based Linux environment, based on the 2.6.30 kernel. The kernel used in the VM is identical to that in the Native configuration, with the exception that the virtio NIC drivers are loaded. The virtio MTU is configured as 9000 Bytes. We refer to the 1 and 10 Gbps results in this configuration as *VNET/P-1G* and *VNET/P-10G*, respectively.

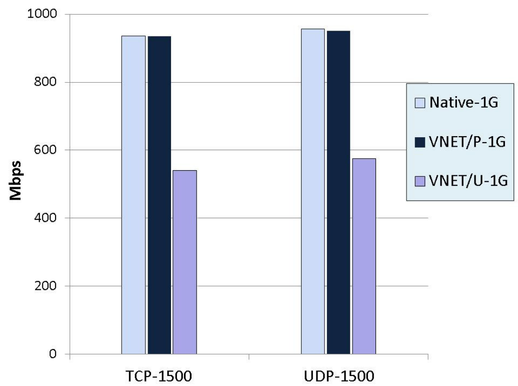
To assure accurate time measurements both natively and in the virtualized case, our guest is configured to use the CPU's cycle counter, and Palacios is configured to allow the guest direct access to the underlying hardware cycle counter. Our 1 Gbps NIC only supports MTUs up to 1500 bytes, while our 10 Gbps NIC can support MTUs of up to 9000 bytes. We use these maximum sizes unless otherwise specified.

5.2 TCP and UDP microbenchmarks

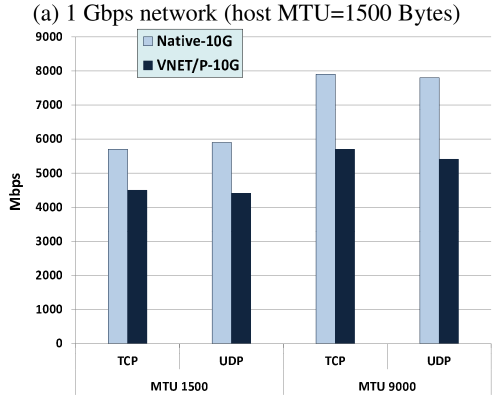
Latency and throughput are the fundamental measurements we use to evaluate the VNET/P system performance. First, we consider these at the IP level, measuring the round-trip latency, the UDP goodput, and the TCP throughput between two nodes. We measure round-trip latency using *ping* by sending ICMP packets of different sizes. UDP and TCP throughput are measured using *ttcp-1.10*.

UDP and TCP with a standard MTU: Figure 5 shows the TCP throughput and UDP goodput achieved in each of our configurations on each NIC. For the 1 Gbps network, host MTU is set to 1500 bytes, and for the 10 Gbps network, host MTUs of 1500 bytes and 9000 bytes are both tested. For 1 Gbps, we also compare with VNET/U running on the same hardware with Palacios. Compared to previously reported results (21.5 MB/s, 1 ms), the combination of the faster hardware we use here, and Palacios, leads to VNET/U increasing its bandwidth by 330%, to 71 MB/s, with a 12% reduction in latency, to 0.88 ms. We also tested VNET/U with VMware, finding that bandwidth increased by 63% to 35 MB/s, with no change in latency. The difference in performance of VNET/U on the two VMMs is due to a custom tap interface in Palacios, while on VMware, the standard host-only tap is used. Even with this optimization, VNET/U cannot saturate a 1 Gbps link.

We begin by considering UDP goodput when a standard host MTU size is used. For UDP measurements, ttcp was configured to use 64000 byte writes sent as fast as possible over 60 seconds. For



(a) 1 Gbps network (host MTU=1500 Bytes)



(b) 10 Gbps network (host MTU=1500, 9000 Bytes)

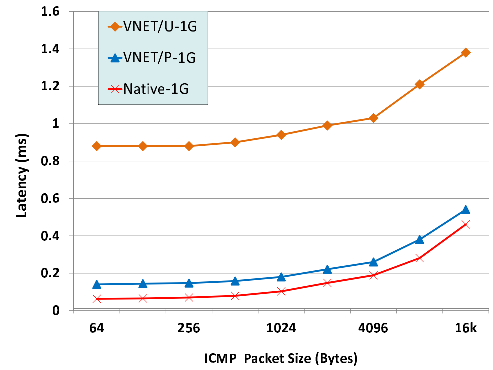
Figure 5: End-to-end TCP throughput and UDP goodput of VNET/P on 1 and 10 Gbps network. VNET/P performs identically to the native case for the 1 Gbps network and achieves 74–78% of native throughput for the 10 Gbps network.

the 1 Gbps network, VNET/P easily matches the native goodput. For the 10 Gbps network, VNET/P achieves 74% of the native UDP goodput.

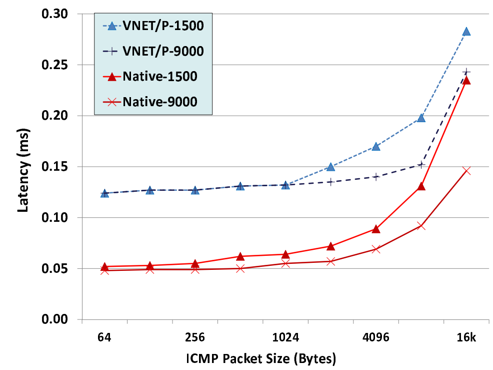
For TCP throughput, ttcp was configured to use a 256 KB socket buffer, and to communicate 40 MB writes were made. Similar to the UDP results, VNET/P has no difficulty achieving native throughput on the 1 Gbps network. On the 10 Gbps network, using a standard Ethernet MTU, it achieves 78% of the native throughput. The UDP goodput and TCP throughput that VNET/P is capable of, using a standard Ethernet MTU, are approximately 8 times those we would expect from VNET/U given the 1 Gbps results.

UDP and TCP with a large MTU: We now consider TCP and UDP performance with 9000 byte jumbo frames our 10 Gbps NICs support. We adjusted the VNET/P MTU so that the ultimate encapsulated packets will fit into these frames without fragmentation. For TCP we configure ttcp to use writes of corresponding size, maximize the socket buffer size, and do 4 million writes. For UDP, we configure ttcp to use commensurately large packets sent as fast as possible for 60 seconds. The results are also shown in the Figure 5. We can see that performance increases across the board compared to the 1500 byte MTU results. Compared to the VNET/U performance we would expect in this configuration, the UDP goodput and TCP throughput of VNET/P are over 10 times higher.

Latency: Figure 6 shows the round-trip latency for different packet sizes, as measured by ping. The latencies are the average of 100 measurements. While the increase in latency of VNET/P over Native is significant in relative terms (2x for 1 Gbps, 3x for



(a) 1 Gbps network (Host MTU=1500 Bytes)



(b) 10 Gbps network (Host MTU=1500, 9000 Bytes)

Figure 6: End-to-end round-trip latency of VNET/P as a function of ICMP packet size. Small packet latencies on a 10 Gbps network in VNET/P are ∼130 μs.

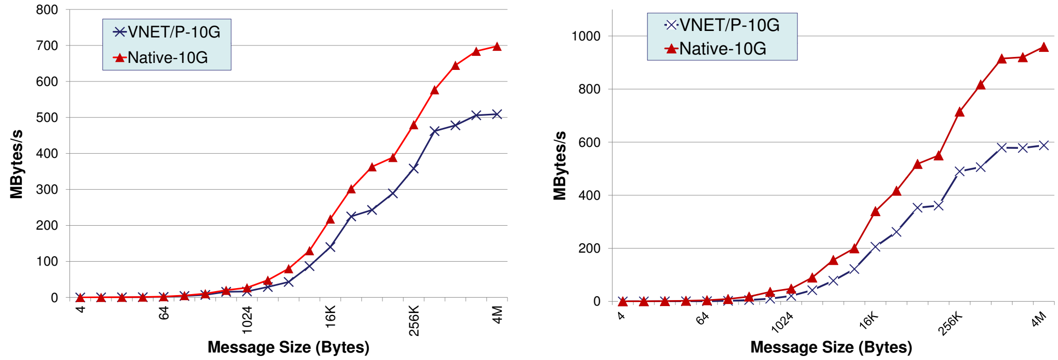
10 Gbps), it is important to keep in mind the absolute performance. On a 10 Gbps network, VNET/P achieves a 130 μs round-trip, end-to-end latency. The latency of VNET/P is almost seven times lower than that of VNET/U.

5.3 MPI microbenchmarks

Parallel programs for distributed memory computers are typically written to the MPI interface standard. We used the OpenMPI 1.3 [7] implementation in our evaluations. We measured the performance of MPI over VNET/P by employing the widely-used Intel MPI Benchmark Suite (IMB 3.2.2) [16], focusing on the point-to-point messaging performance. We compared the basic MPI latency and bandwidth achieved by VNET/P and natively.

Figures 7 and 8(a) illustrate the latency and bandwidth reported by Intel MPI PingPong benchmark for our 10 Gbps configuration. Here the latency measured is the one-way, end-to-end, application-level latency. That is, it is the time from when an MPI send starts on one machine to when its matching MPI receive call completes on the other machine. For both Native and VNET/P, the host MTU is set to 9000 bytes.

VNET/P's small message MPI latency is about 55 μs, about 2.5 times worse than the native case. However, as the message size increases, the latency difference decreases. The measurements of end-to-end bandwidth as a function of message size show that native MPI bandwidth is slightly lower than raw UDP or TCP throughput, and VNET/P performance tracks it similarly. The bottom line is that the current VNET/P implementation can deliver an MPI latency of 55 μs and bandwidth of 510 MB/s on 10 Gbps Ethernet hardware.



(a) One-way bandwidth (b) SendRecv Bandwidth

Figure 8: Intel MPI PingPong microbenchmark showing (a) one-way bandwidth and (b) bidirectional bandwidth as a function of message size on the 10 Gbps hardware.

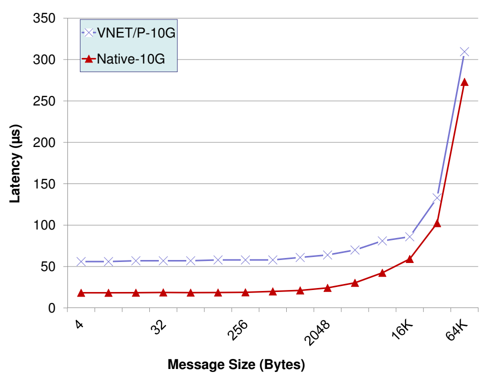


Figure 7: One-way latency on 10 Gbps hardware from Intel MPI PingPong microbenchmark

Figure 8(b) shows the results of the MPI SendRecv microbenchmark in which each node simultaneously sends and receives. There is no reduction in performance between the bidirectional case and the unidirectional case.

5.4 HPCC benchmarks on more nodes

To test VNET/P performance on more nodes, we ran the HPCC benchmark [15] suite on a 6 node cluster with 1 Gbps and 10 Gbps Ethernet. Each node was equipped with two quad-core 2.3 GHz 2376 AMD Opterons, 32 GB of RAM, an nVidia MCP55 Forthdeth 1 Gbps Ethernet NIC and a NetEffect NE020 10 Gbps Ethernet NIC. The nodes were connected via a Fujitsu XG2000 10Gb Ethernet Switch.

The VMs were all configured exactly as in previous tests, with 4 virtual cores, 1 GB RAM, and a virtio NIC. For the VNET/P test case, each host ran one VM. We executed tests with 2, 3, 4, 5, and 6 VMs, with 4 HPCC processes per VM (one per virtual core). Thus, our performance results are based on HPCC with 8, 12, 16, 20 and 24 processes for both VNET/P and Native tests. In the native cases, no VMs were used, and the processes ran directly on the host. For 1 Gbps testing, the host MTU was set to 1500, while for the 10 Gbps cases, the host MTU was set to 9000.

Latency-bandwidth benchmark: This benchmark consists of the ping-pong test and the ring-based tests. The ping-pong test measures the latency and bandwidth between all distinct pairs of processes. The ring-based tests arrange the processes into a ring topology and then engage in collective communication among neighbors in the ring, measuring bandwidth and latency. The ring-based tests model the communication behavior of multi-dimensional domain-decomposition applications. Both naturally ordered rings and randomly ordered rings are evaluated. Communication is done with MPI non-blocking sends and receives, and MPI SendRecv. Here, the bandwidth per process is defined as total message volume divided by the number of processes and the maximum time needed in all processes. We reported the ring-based bandwidths by multiplying them with the number of processes in the test.

Figure 9 shows the results for different numbers of test processes. The ping-pong latency and bandwidth results are consistent with what we saw in the previous microbenchmarks: in the 1 Gbps network, bandwidth are nearly identical to those in the native cases while latencies are 1.2–2 times higher. In the 10 Gbps network, bandwidths are within 60-75% of native while latencies are about 2 to 3 times higher. Both latency and bandwidth under VNET/P exhibit the same good scaling behavior of the native case.

5.5 Application benchmarks

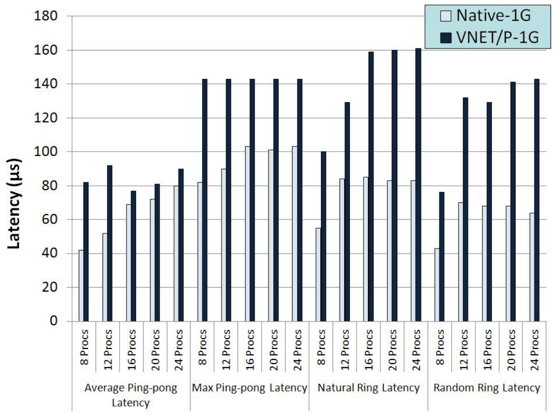
We evaluated the effect of a VNET/P overlay on application performance by running two HPCC application benchmarks and the whole NAS benchmark suite on the cluster described in Section 5.4. Overall, the performance results from the HPCC and NAS benchmarks suggest that VNET/P can achieve high performance for many parallel applications.

HPCC application benchmarks: We considered the two application benchmarks from the HPCC suite that exhibit the large volume and complexity of communication: MPIRandomAcceess and MPIFFT. For 1 Gbps networks, the difference in performance is negligible so we focus here on 10 Gbps networks.

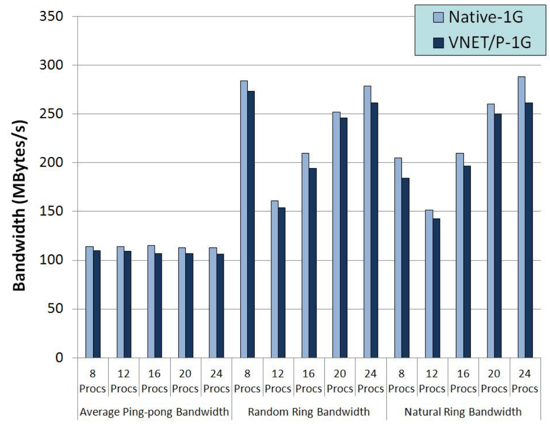
In MPIRandomAccess, random numbers are generated and written to a distributed table, with local buffering. Performance is measured by the billions of updates per second (GUPs) that are performed. Figure 10(a) shows the results of MPIRandomAccess, comparing the VNET/P and Native cases. VNET/P achieves 65-70% application performance compared to the native cases, and performance scales similarly.

MPIFFT implements a double precision complex one-dimensional Discrete Fourier Transform (DFT). Figure 10(b) shows the results of MPIFFT, comparing the VNET/P and Native cases. It shows that VNET/P's application performance is within 60-70% of native performance, with performance scaling similarly.

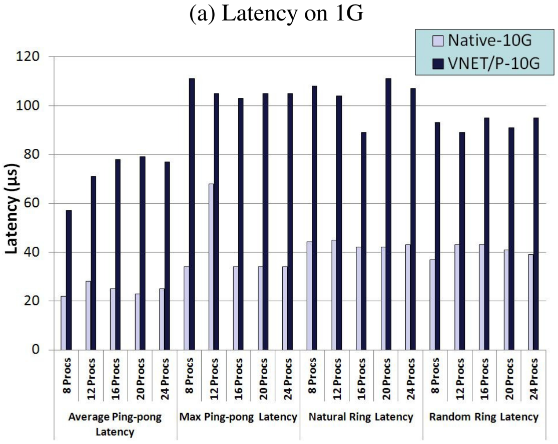
NAS parallel benchmarks: The NAS Parallel Benchmark (NPB) suite [46] is a set of five kernels and three pseudo-applications that is widely used in parallel performance evaluation. We specif-



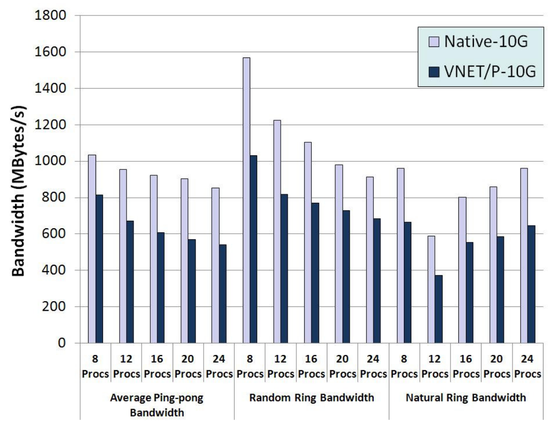
(a) Latency on 1G



(b) Bandwidth on 1G

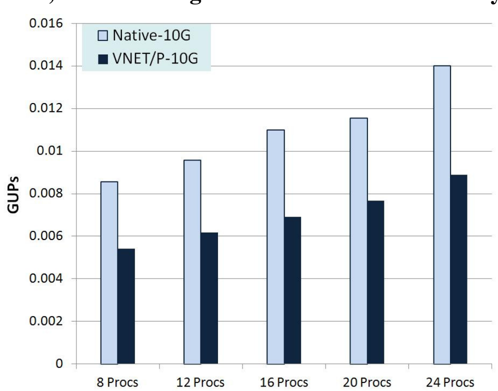


(c) Latency on 10G

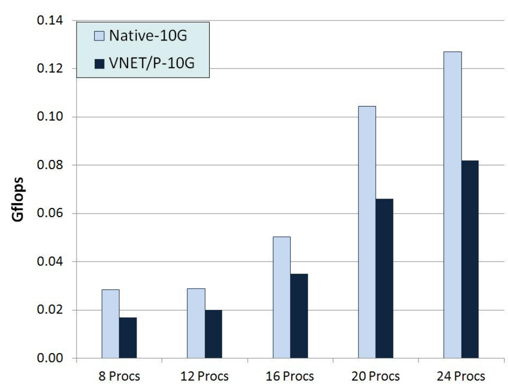


(d) Bandwidth on 10G

Figure 9: HPCC Latency-bandwidth benchmark for both 1 Gbps and 10 Gbps. Ring-based bandwidths are multiplied by the total number of processes in the test. The ping-pong latency and bandwidth tests show results that are consistent with the previous microbenchmarks, while the ring-based tests show that latency and bandwidth of VNET/P scale similarly to the native cases.



(a) MPIRandomAccess



(b)MPIFFT

Figure 10: HPCC application benchmark results. VNET/P achieves reasonable and scalable application performance when supporting communication-intensive parallel application workloads on 10 Gbps networks. On 1 Gbps networks, the difference is negligible.

ically use NPB-MPI 2.4 in our evaluation. In our description, we name executions with the format "name.class.procs". For example, *bt.B.16* means to run the BT benchmark on 16 processes with a class B problem size.

We run each benchmark with at least two different scales and one problem size, except FT, which is only run with 16 processes. One VM is run on each physical machine, and it is configured as described in Section 5.4. The test cases with 8 processes are running within 2 VMs and 4 processes started in each VM. The test cases with 9 processes are run with 4 VMs and 2 or 3 processes per VM. Test cases with with 16 processes have 4 VMs with 4 processes per

Mop/s	Native-1G	VNET/P-1G	$\frac{VNET/P-1G}{Native-1G}$(%)		Native-10G	VNET/P-10G	$\frac{VNET/P-10G}{Native-10G}$(%)
ep.B.8	103.15	101.94	98.8%		102.18	102.12	99.9%
ep.B.16	204.88	203.9	99.5%		208	206.52	99.3%
ep.C.8	103.12	102.1	99.0%		103.13	102.14	99.0%
ep.C.16	206.24	204.14	99.0%		206.22	203.98	98.9%
mg.B.8	4400.52	3840.47	87.3%		5110.29	3796.03	74.3%
mg.B.16	1506.77	1498.65	99.5%		9137.26	7405	81.0%
cg.B.8	1542.79	1319.43	85.5%		2096.64	1806.57	86.2%
cg.B.16	160.64	159.69	99.4%		592.08	554.91	93.7%
ft.B.16	1575.83	1290.78	81.9%		1432.3	1228.39	85.8%
is.B.8	78.88	74.61	94.6%		59.15	59.04	99.8%
is.B.16	35.99	35.78	99.4%		23.09	23	99.6%
is.C.8	89.54	82.15	91.7%		132.08	131.87	99.8%
is.C.16	84.76	82.22	97.0%		77.77	76.94	98.9%
lu.B.8	6818.52	5495.23	80.6%		7173.65	6021.78	83.9%
lu.B.16	7847.99	6694.12	85.3%		12981.86	9643.21	74.3%
sp.B.9	1361.38	1215.85	89.3%		2634.53	2421.98	91.9%
sp.B.16	1489.32	1399.6	94.0%		3010.71	2916.81	96.9%
bt.B.9	3423.52	3297.04	96.3%		5229.01	4076.52	78.0%
bt.B.16	4599.38	4348.99	94.6%		6315.11	6105.11	96.7%

Figure 11: NAS Parallel Benchmark performance with VNET/P on 1 Gbps and 10 Gbps networks. VNET/P can achieve native performance on many applications, while it can get reasonable and scalable performance when supporting highly communication-intensive parallel application workloads.

VM. We report each benchmark's *Mop/s total* result for both native and with VNET/P.

Figure 11 shows the NPB performance results, comparing the VNET/P and Native cases on both 1 Gbps and 10 Gbps networks. The upshot of the results is that for most of the NAS benchmarks, VNET/P is able to achieve in excess of 95% of the native performance even on 10 Gbps networks. We now describe the results for each benchmark.

EP is an "embarrassingly parallel" kernel that estimates the upper achievable limits for floating point performance, It does not require a significant interprocessor communication. VNET/P achieves native performance in all cases.

MG is a simplified multigrid kernel that requires highly structured long distance communication and tests both short and long distance data communication. With 16 processes, MG achieves native performance on the 1 Gbps network, and 81% of native performance on the 10 Gbps network.

CG implements the conjugate gradient method to compute an approximation to the smallest eigenvalue of a large sparse symmetric positive definite matrix. It is typical of unstructured grid computations in that it tests irregular long distance communication, employing unstructured matrix vector multiplication. With 16 processes, CG achieves native performance on the 1 Gbps network and 94% of native performance on the 10 Gbps network.

FT implements the solution of partial differential equations using FFTs, and captures the essence of many spectral codes. It is a rigorous test of long-distance communication performance. With 16 nodes, it achieves 82% of native performance on 1 Gbps and 86% of native performance on 10 Gbps.

IS implements a large integer sort of the kind that is important in particle method codes and tests both integer computation speed and communication performance. Here VNET/P achieves native performance in all cases.

LU solves a regular-sparse, block (5×5) lower and upper triangular system, a problem associated with implicit computational fluid dynamics algorithms. VNET/P achieves 75%-85% of native performance on this benchmark, and there is no significant difference between the 1 Gbps and 10 Gbps network.

SP and BT implement solutions of multiple, independent systems of non diagonally dominant, scalar, pentadiagonal equations, also common in computational fluid dynamics. The salient difference between the two is the communication to computation ratio. For SP with 16 processes, VNET/P achieves 94% of native performance on 1 Gbps around 97% of native on 10 Gbps. For BT at the same scale, 95% of native at 1 Gbps and 97% of native at 10 Gbps are achieved.

6. VNET/P FOR INFINIBAND

In support of hardware independence, the 3rd goal of VNET articulated in Section 3, we have developed an implementation of VNET/P that allows guests that only support Ethernet NICs to be seamlessly run on top of an InfiniBand network, or to span InfiniBand networks and other networks. Regardless of the underlying networking hardware, the guests see a simple Ethernet LAN.

For the current Infiniband implementation, the host OS that is used is Sandia National Labs' Kitten lightweight kernel. Kitten has, by design, a minimal set of in-kernel services. For this reason, the VNET/P Bridge functionality is not implemented in the kernel, but rather in a privileged service VM called the Bridge VM that has direct access to the physical Infiniband device.

In place of encapsulating Ethernet packets in UDP packets for transmission to a remote VNET/P core, VNET/P's InfiniBand support simply maps Ethernet packets to InfiniBand frames. These frames are then transmitted through an InfiniBand queue pair accessed via the Linux IPoIB framework.

We conducted preliminary performance tests of VNET/P on InfiniBand using 8900 byte TCP payloads running on ttcp on a testbed similar to the one described in Section 5.1. Here, each node was a dual quad-core 2.3 GHz 2376 AMD Opteron machine with 32 GB of RAM and a Mellanox MT26428 InfiniBand NIC in a PCI-e slot. The Infiniband NICs were connected via a Mellanox MTS 3600 36-port 20/40Gbps InfiniBand switch.

It is important to point out that VNET/P over Infiniband is a work in progress and we present it here as a proof of concept. Nonetheless, on this testbed it achieved 4.0 Gbps end-to-end TCP throughput, compared to 6.5 Gbps when run natively on top of IP-over-InfiniBand in Reliable Connected (RC) mode.

7. CONCLUSION AND FUTURE WORK

We have described the VNET model of overlay networking in a distributed virtualized computing environment and our efforts in extending this simple and flexible model to support tightly-coupled high performance computing applications running on high-performance networking hardware in current supercomputing environments, future data centers, and future clouds. VNET/P is our design and implementation of VNET for such environments. Its design goal is to achieve near-native throughput and latency on 1 and 10 Gbps Ethernet, InfiniBand, and other high performance interconnects.

To achieve performance, VNET/P relies on several key techniques and systems, including lightweight virtualization in the Palacios virtual machine monitor, high-performance I/O, and multicore overlay routing support. Together, these techniques enable VNET/P to provide a simple and flexible level 2 Ethernet network abstraction in a large range of systems no matter the actual underlying networking technology is. While our VNET/P implementation is tightly integrated into our Palacios virtual machine monitor, the principles involved could be used in other environments as well.

We are currently working to further enhance VNET/P's performance through its guarded privileged execution directly in the guest, including an uncooperative guest. We are also enhancing its functionality through broader support on InfiniBand and on the Cray SeaStar interconnect.

8. REFERENCES

[1] ABU-LIBDEH, H., COSTA, P., ROWSTRON, A., O'SHEA, G., AND DONNELLY, A. Symbiotic routing in future data centers. In *Proceedings of SIGCOMM* (August 2010).

[2] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient overlay networks. In *Proceedings of SOSP* (March 2001).

[3] BAVIER, A. C., FEAMSTER, N., HUANG, M., PETERSON, L. L., AND REXFORD, J. In vini veritas: realistic and controlled network experimentation. In *Proceedings of SIGCOMM* (September 2006).

[4] DINDA, P., SUNDARARAJ, A., LANGE, J., GUPTA, A., AND LIN, B. Methods and systems for automatic inference and adaptation of virtualized computing environments, March 2012. United States Patent Number 8,145,760.

[5] EVANGELINOS, C., AND HILL, C. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. In *Proceedings of Cloud Computing and its Applications (CCA)* (October 2008).

[6] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003).

[7] GABRIEL, E., FAGG, G. E., BOSILCA, G., ANGSKUN, T., DONGARRA, J. J., SQUYRES, J. M., SAHAY, V., KAMBADUR, P., BARRETT, B., LUMSDAINE, A., CASTAIN, R. H., DANIEL, D. J., GRAHAM, R. L., AND WOODALL, T. S. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings of the 11th European PVM/MPI Users' Group Meeting* (September 2004).

[8] GANGULY, A., AGRAWAL, A., BOYKIN, P. O., AND FIGUEIREDO, R. IP over P2P: Enabling self-configuring virtual ip networks for grid computing. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (April 2006).

[9] GREENBERG, A., HAMILTON, J. R., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D. A., PATEL, P., AND SENGUPTA, S. VL2: A scalable and flexible data center network. In *Proceedings of SIGCOMM* (August 2009).

[10] GUO, C., LU, G., LI, D., WU, H., ZHANG, X., SHI, Y., TIAN, C., ZHANG, Y., AND LU, S. Bcube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of SIGCOMM* (August 2009).

[11] GUPTA, A., AND DINDA, P. A. Inferring the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)* (June 2004).

[12] GUPTA, A., ZANGRILLI, M., SUNDARARAJ, A., HUANG, A., DINDA, P., AND LOWEKAMP, B. Free network measurement for virtual machine distributed computing. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2006).

[13] HUA CHU, Y., RAO, S., SHESHAN, S., AND ZHANG, H. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM* (August 2001).

[14] HUANG, W., LIU, J., ABALI, B., AND PANDA, D. A case for high performance computing with virtual machines. In *Proceedings of the 20th ACM International Conference on Supercomputing (ICS)* (June–July 2006).

[15] INNOVATIVE COMPUTING LABORATORY. Hpc challenge benchmark. http://icl.cs.utk.edu/hpcc/.

[16] INTEL. Intel cluster toolkit 3.0 for linux. http://software.intel.com/en-us/articles/intel-mpi-benchmarks/.

[17] JIANG, X., AND XU, D. Violin: Virtual internetworking on overlay infrastructure. Tech. Rep. CSD TR 03-027, Department of Computer Sciences, Purdue University, July 2003.

[18] JOSEPH, D. A., KANNAN, J., KUBOTA, A., LAKSHMINARAYANAN, K., STOICA, I., AND WEHRLE, K. Ocala: An architecture for supporting legacy applications over overlays. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI)* (May 2006).

[19] KALLAHALLA, M., UYSAL, M., SWAMINATHAN, R., LOWELL, D. E., WRAY, M., CHRISTIAN, T., EDWARDS, N., DALTON, C. I., AND GITTLER, F. Softudc: A software-based data center for utility computing. *IEEE Computer 37*, 11 (2004), 38–46.

[20] KIM, C., CAESAR, M., AND REXFORD, J. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *Proceedings of SIGCOMM* (August 2008).

[21] KUMAR, S., RAJ, H., SCHWAN, K., AND GANEV, I. Re-architecting vmms for multicore systems: The sidecore approach. In *Proceedings of the 2007 Workshop on the Interaction between Operating Systems and Computer Architecture* (June 2007).

[22] LANGE, J., AND DINDA, P. Transparent network services via a virtual traffic layer for virtual machines. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (June 2007).

[23] LANGE, J., DINDA, P., HALE, K., AND XIA, L. An introduction to the palacios virtual machine monitor—release 1.3. Tech. Rep. NWU-EECS-11-10, Department of Electrical Engineering and Computer Science, Northwestern University, October 2011.

[24] LANGE, J., PEDRETTI, K., DINDA, P., BAE, C., BRIDGES, P., SOLTERO, P., AND MERRITT, A. Minimal-overhead virtualization of a large scale supercomputer. In *Proceedings of the 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)* (March 2011).

[25] LANGE, J., PEDRETTI, K., HUDSON, T., DINDA, P., CUI, Z., XIA, L., BRIDGES, P., GOCKE, A., JACONETTE, S., LEVENHAGEN, M., AND BRIGHTWELL, R. Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (April 2010).

[26] LANGE, J., SUNDARARAJ, A., AND DINDA, P. Automatic dynamic run-time optical network reservations. In *Proceedings of the 14th International Symposium on High Performance Distributed Computing (HPDC)* (July 2005).

[27] LIN, B., AND DINDA, P. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of ACM/IEEE SC (Supercomputing)* (November 2005).

[28] LIN, B., SUNDARARAJ, A., AND DINDA, P. Time-sharing parallel applications with performance isolation and control. In *Proceedings of the 4th IEEE International Conference on Autonomic Computing (ICAC)* (June 2007).

[29] LIU, J., HUANG, W., ABALI, B., AND PANDA, D. High performance vmm-bypass i/o in virtual machines. In *Proceedings of the USENIX Annual Technical Conference* (May 2006).

[30] MENON, A., COX, A. L., AND ZWAENEPOEL, W. Optimizing network virtualization in xen. In *Proceedings of the USENIX Annual Technical Conference (USENIX)* (May 2006).

[31] MERGEN, M. F., UHLIG, V., KRIEGER, O., AND XENIDIS, J. Virtualization for high-performance computing. *Operating Systems Review 40*, 2 (2006), 8–11.

[32] MYSORE, R. N., PAMBORIS, A., FARRINGTON, N., HUANG, N., MIRI, P., RADHAKRISHNAN, S., SUBRAMANYA, V., AND VAHDAT, A. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of SIGCOMM* (August 2009).

[33] NURMI, D., WOLSKI, R., GRZEGORZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L., AND ZAGORODNOV, D. The eucalyptus open-source cloud-computing system. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)* (May 2009).

[34] OSTERMANN, S., IOSUP, A., YIGITBASI, N., PRODAN, R., FAHRINGER, T., AND EPEMA, D. An early performance analysis of cloud computing services for scientific computing. Tech. Rep. PDS2008-006, Delft University of Technology, Parallel and Distributed Systems Report Series, December 2008.

[35] RAJ, H., AND SCHWAN, K. High performance and scalable i/o virtualization via self-virtualized devices. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 2007).

[36] RUSSELL, R. virtio: towards a de-facto standard for virtual i/o devices. *Operating Systems Review 42*, 5 (2008), 95–103.

[37] RUTH, P., JIANG, X., XU, D., AND GOASGUEN, S. Towards virtual distributed environments in a shared infrastructure. *IEEE Computer* (May 2005).

[38] RUTH, P., MCGACHEY, P., JIANG, X., AND XU, D. Viocluster: Virtualization for dynamic computational domains. In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster)* (September 2005).

[39] SHAFER, J., CARR, D., MENON, A., RIXNER, S., COX, A. L., ZWAENEPOEL, W., AND WILLMANN, P. Concurrent direct network access for virtual machine monitors. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture (HPCA)* (February 2007).

[40] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001* (2001), pp. 149–160.

[41] SUGERMAN, J., VENKITACHALAN, G., AND LIM, B.-H. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference* (June 2001).

[42] SUNDARARAJ, A., AND DINDA, P. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd USENIX Virtual Machine Research And Technology Symposium (VM 2004)* (May 2004). Earlier version available as Technical Report NWU-CS-03-27, Department of Computer Science, Northwestern University.

[43] SUNDARARAJ, A., GUPTA, A., , AND DINDA, P. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 2005).

[44] SUNDARARAJ, A., SANGHI, M., LANGE, J., AND DINDA, P. An optimization problem in adaptive virtual environmnets. In *Proceedings of the seventh Workshop on Mathematical Performance Modeling and Analysis (MAMA)* (June 2005).

[45] TSUGAWA, M. O., AND FORTES, J. A. B. A virtual network (vine) architecture for grid computing. In *20th International Parallel and Distributed Processing Symposium (IPDPS)* (April 2006).

[46] VAN DER WIJNGAART, R. NAS parallel benchmarks version 2.4. Tech. Rep. NAS-02-007, NASA Advanced Supercomputing (NAS Division), NASA Ames Research Center, October 2002.

[47] WOLINSKY, D., LIU, Y., JUSTE, P. S., VENKATASUBRAMANIAN, G., AND FIGUEIREDO, R. On the design of scalable, self-configuring virtual networks. In *Proceedings of 21st ACM/IEEE International Conference of High Performance Computing, Networking, Storage, and Analysis (SuperComputing / SC)* (November 2009).

[48] XIA, L., LANGE, J., DINDA, P., AND BAE, C. Investigating Virtual Passthrough I/O on Commodity Devices. *Operating Systems Review 43*, 3 (July 2009). Initial version appeared at WIOV 2008.