Reducing complexity of Virtual Machine networking

Sander Vrijders[1], Vincenzo Maffione[2], Dimitri Staessens[1], Francesco Salvestrini[2]
Matteo Biancani[3], Eduard Grasa[4], Didier Colle[1], Mario Pickavet[1], John Day[5], Lou Chitkushev[5]
[1]Ghent University - iMinds, INTEC, Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium
E-mail: firstname.lastname@intec.ugent.be
[2]Nextworks s.r.l., Pisa, Italy
[3]Interoute S.p.A, Rome, Italy
[4]i2CAT Foundation, Jordi Girona, Barcelona, Spain
[5]Computer Science, Metropolitan College, Boston University, Massachusetts, USA

***Abstract*—Virtualization is an enabling technology that improves scalability, reliability and flexibility. Virtualized networking is tackled by emulating or paravirtualizing Network Interface Cards (NICs). This approach, however, leads to complexities (implementation and management) and has to conform to some limitations imposed by the Ethernet standard. The Recursive InterNetwork Architecure (RINA) turns the current approach to virtualized networking on its head: instead of emulating networks to perform inter process communication on a single processing system, it sees networking as an extension to local inter-process communication. In this article, we show how RINA can leverage a paravirtualization approach to achieve a more manageable solution for virtualized networking. We also present experimental results performed on IRATI, the reference open source implementation of RINA, which shows the potential performance that can be achieved by deploying our solution.**

I. Introduction

Virtualization technologies provide a cost-effective way of increasing the scalability, reliability and flexibility of services deployed over the internet. Virtual Machine networking, the way a VM connects to the physical network, is an aspect of high importance in the virtualization world, with network performance being paramount [1]. The traditional way that hypervisors implement VM networking is based on NIC emulation - e.g. QEMU [2], VirtualBox [3], VMWare [4] are able to emulate Intel e1000, Realtek r8169 and other NICs. This is also referred to as full NIC emulation, where the hypervisor implements a NIC hardware model in software, including the transmit and receive memory mapped rings and the Peripheral Component Interconnect (PCI) registers.

The paravirtualization approach initially proposed by Xen [5] with the netfront/netback paravitualized NIC, gained popularity over traditional emulation, leading to the advent of VMware vmxnet [6] and the VirtIO [7] standard for I/O paravirtualization. NIC paravirtualization (and I/O paravirtualization in general) is a software technique that greatly improves VM networking performance and eases implementation of VM I/O support in hypervisors. Paravirtualization removes the need to implement the emulation of hardware-related details and features, thereby exposing a simple and efficient interface for shared-memory communication between VM and hypervisor. The main advantage of the paravirtualization approach is a gain in performance. However, it is still necessary to present a NIC device in the VM, which makes it a solution that is hard to manage.

In this article we try to reduce the complexity associated with managing Virtual Machine (VM) communication by applying the paravirtualization paradigm, a cleaner and simpler interface for VM I/O, in the Recursive InterNetwork Architecture [8]. This results in a simple and clean solution for the communication between VMs and their hypervisor, without the need for the VM to even implement a (paravirtualized) NIC.

In Section II we will give a brief description of RINA. In Section III, we introduce IRATI, the open-source implementation of RINA in Linux/OS. In Section IV we introduce our main contribution, a new component, which is called the shim DIF for hypervisors, that leverages the paravirtualization approach. Some experimental results with this new component are presented in Section V. Finally, in Section VI we explore future works and in Section VII we conclude the paper.

II. Recursive InterNetwork Architecture

The Recursive InterNetwork Architecture is a network architecture *ab initio*, aiming to provide an alternative to the current TCP/IP Internet architecture. RINA extends *Inter Process Communication (IPC)*, the way processes communicate on a single *processing system*, from a local concept to the scope of an (inter)network [9].

The endpoints of all communications are *processes*, the means of communication between them is called the *IPC service*. By definition, if processes can communicate locally using shared memory (test-and-set), they reside on the same *processing system*. In this case, an *Operating System* will provide and manage the IPC service between processes. If processes can't communicate using test-and-set, they are on different processing systems. In RINA, an operating system process that provides IPC services is called an *IPC process (IPCP)*. To provide the IPC service to processes residing on multiple processing systems, IPCP instances on each system work together to form a *Distributed IPC Facility (DIF)*. A DIF

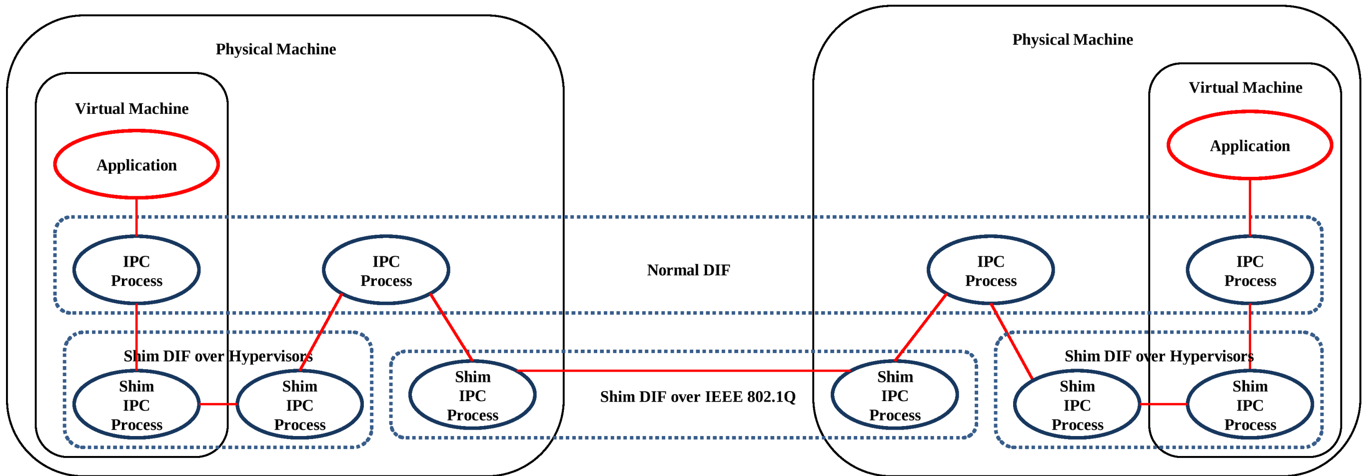


Fig. 1. An example of the Recursive InterNetwork Architecture

is the core organising structure in RINA and corresponds to what typically is referred to as a "network layer". As many DIFs can be stacked on top of each other as needed by the network designer. Most of the time at least two levels of DIFs are needed, at the scope of links and at the scope of the network, interconnecting nodes over multiple hops. DIFs of higher order can be internetworks, Virtual Private Networks (VPNs) or application-specific virtual networks. A DIF offers a fixed set of functionalities and services (the *mechanism*), but is fully configurable with suitable *policies*, in order to adapt to the environment it operates in and to fulfill the requirements of the applications (or other DIFs) it serves. DIFs are bootstrapped from a single IPCP instance and the process of an IPCP instance joining a DIF is called *enrollment*.

All IPC processes provide the same Application Programming Interface (API) to their users, which can be regular applications or other IPC processes. Through this API - referred to as the IPC API - an application can

- register with a DIF, so that it can be reached by other applications that are clients of this DIF.
- allocate a flow to a registered application.
- read and write from/to an allocated flow.
- deallocate the flow when desired.

IPCPs that both provide the IPC API northbound and make use of the IPC API southbound are called *normal IPCPs* and they form a *normal DIF*. Some internal components of the IPCP are dedicated to layer management, while others are devoted to data transfer or data transfer control. RINA has special IPCPs that are tailored to a transmission medium (possibly incorporating a Media Access Control protocol) or wrap around an existing legacy network technology such as Ethernet and provide the IPC API northbound only. Such IPCPs are called *shim IPCPs* and form *shim DIFs*.

An example scenario is shown in Figure 1. Two physical machines are interconnected over an Ethernet LAN, wrapped by a shim DIF over Ethernet with VLANs (IEEE 802.1Q) [10]. Each physical machine is running a Virtual Machine (VM). Between each VM and the physical machine it runs on, there is a shim DIF for hypervisors, which provides communication directly using shared memory. On top of these shim DIFs lays a normal DIF, which uses the (basic) IPC services provided by the shim DIFs, and which itself provides IPC services to applications running on top. The line denotes the path that Service Data Units (SDUs) sent by the applications would follow through the network in this specific scenario.

Due to space restrictions, a complete discussion of RINA is not possible here. We kindly refer the reader to [8] [9] for further details.

III. The IRATI prototype

IRATI [11] is an open source Linux/OS implementation of RINA, written in C/C++. In IRATI, the data transfer and data transfer control functionalities (the *fast path*) run in kernel space, whereas layer management functionalities (the *slow path*) run in userspace in the context of system daemons; i.e. the IPC Manager daemon and the IPC Process daemons.

The current implementation provides the following features:

- Enrollment, which allows IPCP instances to join an existing DIF.
- Allocation of flows with different QoS characteristics.
- Data Unit protection functionalities like checksumming, encryption and Time To Live mechanisms.
- A simple link-state routing protocol, that works on a flat (not hierarchical) addressing space.
- Extended programmability, by allowing policies to be plugged into components, both in user and kernel space.
- Two shim IPCPs: the shim IPCP over TCP/UDP, the shim IPCP over IEEE 802.1Q [10].

We extended the IRATI prototype with a new component: the shim DIF for hypervisors. This new component provides a point-to-point shim DIF over shared memory between a Virtual Machine and its hypervisor, which is achieved by leveraging the paravirtualization approach.

IV. SHIM DIF FOR HYPERVISORS

Virtual machine networking is commonly implemented by providing VMs with Network Interface Cards (NICs) that are emulated by the hypervisor, which also creates and manages the Virtual Machines (VMs). The emulated NIC forwards the VM's packets to/from the hypervisor's TCP/IP stack. The hypervisor usually connects to the emulated NIC through a special (software) network interface. As an example, Xen [5] Domain 0 uses a so-called *vif* interface (a xen-netback device), while QEMU [2] uses a *tap* interface.

In order to connect the emulated NIC with other VMs hosted by the same hypervisor or with the external network, the hypervisor's software interfaces are bridged to other host interfaces (physical interfaces and/or software interfaces associated to other emulated NICs) using software switches, e.g. OpenVSwitch [12] or the standard Linux in-kernel bridge. Each hypervisor may host many bridges in order to build arbitrary network topologies for VMs.

In TCP/IP, applications require an IP address, which must be assigned to an interface. In RINA this is not the case. Applications request IPC services through the IPC API to any DIF that can provide connectivity to the destination application. Because of this strong API, the physical layer is abstracted away and not visible to applications. In short, the DIF abstraction is at a higher level than the NIC abstraction. As a consequence, there is no need to emulate a NIC to connect the VM stack to the hypervisor's stack. Instead, VM-to-hypervisor point-to-point connectivity is provided directly using shared memory or message passing mechanisms provided by the hypervisor itself. This method is implemented in the shim DIF for hypervisors. While a physical machine will typically have one or more shim DIFs over Ethernet or TCP/UDP as lowest level network access, a VM will have one or more shim DIFs for hypervisors.

Exploring the possibilities of using hypervisor internal mechanisms other than the traditional networking subsystem for VM-to-hypervisor communication provides an easier manageable solution and may also allow for better performance. Of course, it is necessary to configure the DIFs in the network, but management is simplified, since it is always the same architectural component that has to be configured; there is no need for a different ad-hoc component. Since the shim DIF for hypervisors provides VM-to-hypervisor point-to-point connectivity directly using shared memory or message passing mechanisms provided by the hypervisor, it is not restricted by some limitations of Ethernet technology (unlike traditional VM networking).

The shim DIF for hypervisors is built on top of a new device we implemented: the Virtual Message Passing Interface (VMPI) VM-to-hypervisor shared-memory communication mechanism. The high-level architecture of the VMPI is depicted in Figure 2. A VMPI device is used to implement the point-to-point link and it is seen as a special device on both VM and hypervisor. It requires only a very small driver on the guest and hypervisor. The VMPI device implementation is almost entirely datapath since it focuses on message passing only. There is no need for all the details of Ethernet NICs, like tens of configuration registers, autonegotiation, TCP/IP offloading, checksumming, MAC addresses, MTU limitations, VLAN support, configuration of the internal modem (e.g. PHY), internal buffering, DMA configuration, EEPROM configuration.

As a concrete example of the simplicity, the e1000 driver in Linux is implemented in about 17 kilo lines of code (KLoC), the ixgbe driver in 37 KLoC. An emulated NIC such as e1000 is implemented in 3 KLoC on QEMU and in 8 KLoC on VirtualBox. The differences in KLoC are due to a different degree of emulation accuracy. For paravirtualized devices, on the guest, the virtio-net driver is 2 KLoC, whereas the VMPI driver is only 1 KLoC. Similarly, on the host, virtio-net support is about 2 KLoC, while VMPI support is implemented in about 1 KLoC. Most of the complexities of NIC drivers are due to the configuration routines as opposed to the datapath, i.e. the TX and RX rings. This explains the differences in code size among traditional NIC drivers/emulators and paravirtualized solutions. All the configuration-related complexities of an emulated NIC are overhead-only when deployed in VM environments, since there is no real hardware to drive.

The VMPI consists of two blocks: the VMPI High Level (VMPI-HL) and the VMPI Low Level (VMPI-LL). The VMPI-LL block is hypervisor dependent, and is used to access Xen [5] I/O rings or QEMU/KVM [13][7] Virtqueues [2] with a common (internal) interface, referred to as VMPI-IMPL. VMPI-LL acts therefore as a wrapper for the shared memory communication mechanism made available by the hypervisor, effectively making use of the paravirtualization mechanisms Xen and QEMU/KVM offer. The VMPI-IMPL interface is used by the VMPI-HL block, which is hypervisor independent, to implement the VMPI device abstraction. VMPI-HL offers the VMPI API to its users, allowing data to be exchanged between the VM and the hypervisor. Each VMPI device is assigned two identifiers, one on the VM OS and the other on the hypervisor OS. The first identifier is necessary to distinguish multiple VMPI devices in the same VM, while the second one is required to distinguish between the multiple VMPI devices (assigned to possibly different VMs) on the same hypervisor. Nevertheless, the scope of those identifiers is confined to a single OS, so that the management is far easier than MAC management. The scope of MACs needs to be unique on the L2 domain in which the NICs exist (or can exist), which may be a large segment of a data center (DC) infrastructure, involving multiple hypervisors. A shim DIF for hypervisors makes use of the VMPI API offered by a VMPI device, which is a much simpler API than the one between the kernel and the NIC driver, to offer the IPC API to its users. The VMPI identifier serves as an address within the shim DIF for hypervisors. Addresses are contained within a layer in RINA, and the shim DIF for hypervisors is a layer inside the OS.

The shim DIF for hypervisors has several advantages when compared with traditional NIC emulation:

- No need to implement complex and expensive NIC

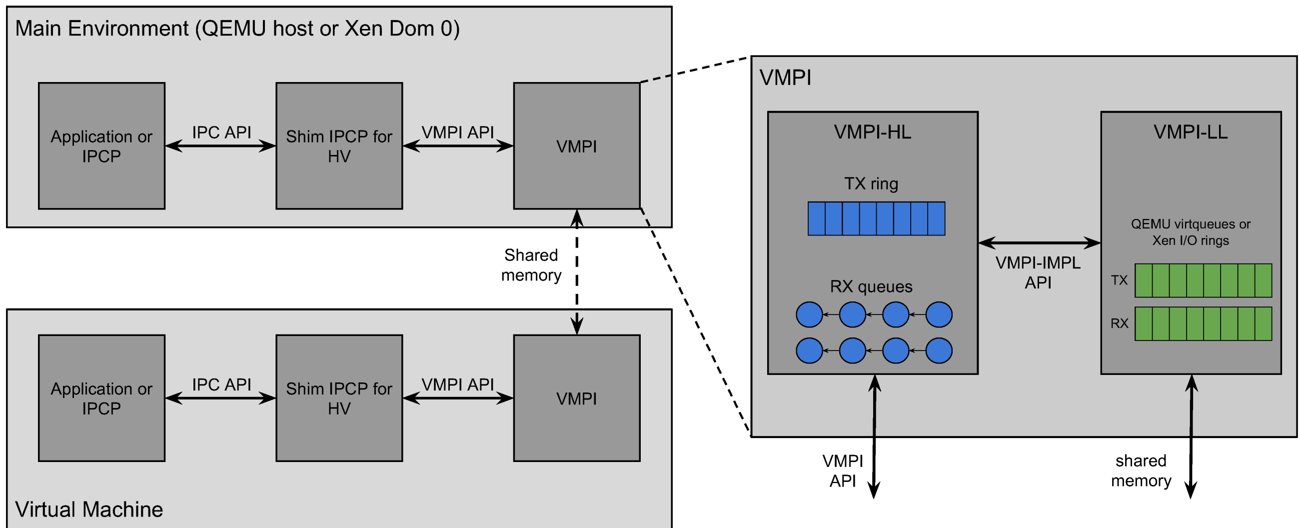


Fig. 2. Shim DIF over hypervisors in-depth

emulation.

- No need to generate and assign MAC addresses, which can become an issue at scale, especially for large DCs. Generation of VMPI-ids is needed, but it is confined to a single OS and thus they need to be unique only per hypervisor, not network-wide. The length of a VMPI-id is quite small because of this.
- No need to create and configure software L2 bridges to connect VMs and hypervisor physical NICs together.
- Users of the shim DIF are not restricted to the Ethernet MTU (maximum payload of 1500 bytes or 9000 bytes if Jumbo frames are used). This restriction is commonly bypassed in traditional VM networking by using the TCP Segmentation Offloading (TSO) features offered by emulated NICs. However, this is a workaround that adds complexity and dependencies between layers, since L4 specific operations are needed in the driver which is situated at L2. It is not needed by the shim DIF over hypervisors.
- No need to perform TCP/UDP checksumming in the emulated NIC (checksum offloading), since shared memory communication is protected from corruption by other means. Checksumming is not actually performed by modern paravirtualized NICs (e.g. virtio-net [7], xen-netfront [5]), but this is again a complex workaround that is not needed by the shim DIF over hypervisors.

V. Experimentation results

In order to assess the possible gains from deploying the shim DIF for hypervisors in the DC, we measured the performance of the IRATI stack against the performance of the TCP/IP stack in Linux, when deployed to support VM networking.

Note however up front that the IRATI stack is currently not optimized for performance. In particular, kernel-space components have several bottlenecks such as high per-packet locking overhead (too many locks taken for each PDU to be processed) and several unnecessary per-packet deep copies. These bottlenecks have been identified but their implementation is out of scope for a research prototype. Therefore, we expect our prototype to underperform by possibly an order of magnitude compared to its theoretical performance.

The tests reported in this section involves one or two physical machines (hosts) that act as a hypervisor for one or two VMs. We performed three different test scenarios:

- Host-to-VM tests; where a benchmarking tool (rina-tgen [14] for IRATI tests and netperf for TCP/IP tests) is used to measure the goodput between a client running in the host and a server running on VM.
- Intra host VM-to-VM tests; where a benchmarking tool is used to measure the goodput between a client running on a VM and a server running on a different VM in the same host.
- Inter host VM-to-VM tests; where a benchmarking tool is used to measure the goodput between a client running on a VM and a server running on a different VM in different hosts. The hosts are connected through a 10 Gbps Ethernet link.

The measurements were taken on a processing system with two 8 core Intel E5-2650v2 (2.6GHz) CPUs and 48GB RAM. QEMU/KVM was chosen as the hypervisor, since it is one of the two hypervisors supported by the shim DIF for hypervisors.

For the host-to-VM scenario, three test sessions were executed. The first two tests assess UDP goodput performance at variable packet size, therefore assessing the performance of traditional VM networking. The tap device corresponding to emulated NIC in the VM is bridged to the host stack through a Linux in-kernel software bridge. This setup is also depicted in Figure 3, where the setup of traditional networking in VMs is shown. The Linux bridge is accessible in the host stack through a bridge interface (e.g. br0). Once the bridge interface and the

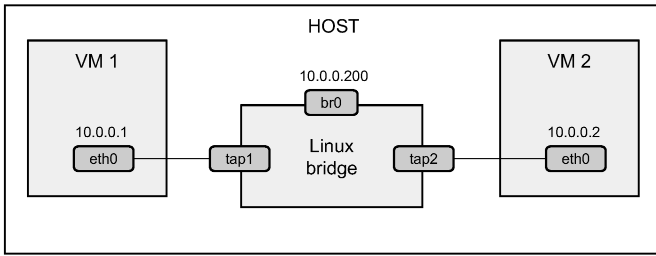


Fig. 3. Setup of traditional networking in VMs

VM NIC have been given an IP address on the same IP subnet, as they are on the same L2 domain, the netperf benchmarking tool is used to measure UDP performance between the host and the VM. In particular, the netperf server (netserver) listens on the VM's interface, while the netperf client runs in the host and uses the bridge interface.

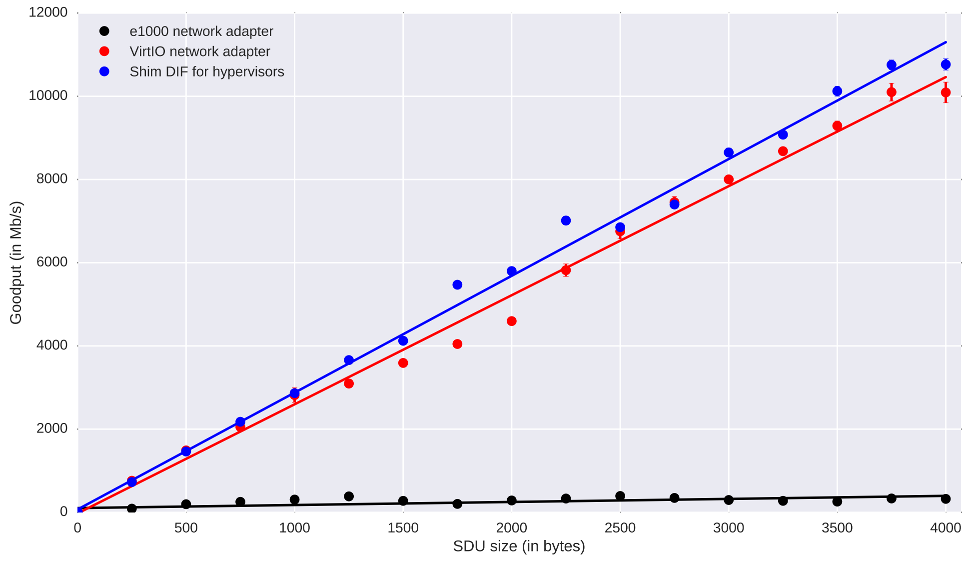
In the first session, a NIC belonging to the Intel e1000 family is used, which is implemented in QEMU by emulating the hardware behaviour (full virtualization); e.g. NIC PCI registers, DMA, packet rings, offloadings, etc. The second test session makes use of a paravirtualized NIC model, the virtio-net device. Paravirtualized devices don't correspond to real hardware, instead they are explicitly designed to be used by virtual machines, in order to save the hypervisor from the burden of emulating real hardware. Paravirtualized devices allows for better performances and code reusability. The virtio standard also provides paravirtualized disk, serial console, number generator, etc. The only difference between the first and the second test session is the model of the emulated NIC. Despite being more virtualization-friendly than e1000 (or other emulated NICs like r8169 or pcnet2000), the guest OS still sees the virtio-net adapter like a normal Ethernet interface, with all the complexities and details involved, e.g. MAC, MTU, TSO, checksum offloading, etc.

The third test session shows the performance of the shim DIF for hypervisors. A scenario comparable to the one deployed in the first and second test sessions involves a shim IPC process for hypervisors on the host and the corresponding one on the guest. No normal IPC processes are used, the applications can run directly over the shim DIF. This is a consequence of the flexibility of RINA, since the application can use the lowest level DIF whose scope is sufficient to support the intended communication (guest-to-host in this case) and that provides the required QoS. This also drastically reduces the header size. In TCP/IP the minimum header size is 84; the Ethernet header is 38 bytes and requires a payload of at least 46 bytes. With the shim DIF for hypervisors, the minimum header length is 4, the actual data is only prepended by an identifier of the VMPI channel. Zero is the control channel, other channels can be used to send data on. The host runs our rina-tgen application in server mode, while the guest runs rina-tgen in client mode. Rina-tgen is a traffic generator for RINA that also functions as a benchmarking application; it uses the IPC API to measure goodput. Each test run consists of the client sending an unidirectional stream of SDUs of a specified size to the server. Measurements have been taken varying the SDU size, ranging from 0 to 4000 (the page size of the processing systems), with a step size of 250. We repeated every measurement 20 times. The result of these goodput measurements for host-to-VM communication scenario are shown in Figure 4a. 95% confidence levels are also depicted, as well as a first degree polynomial regression line.

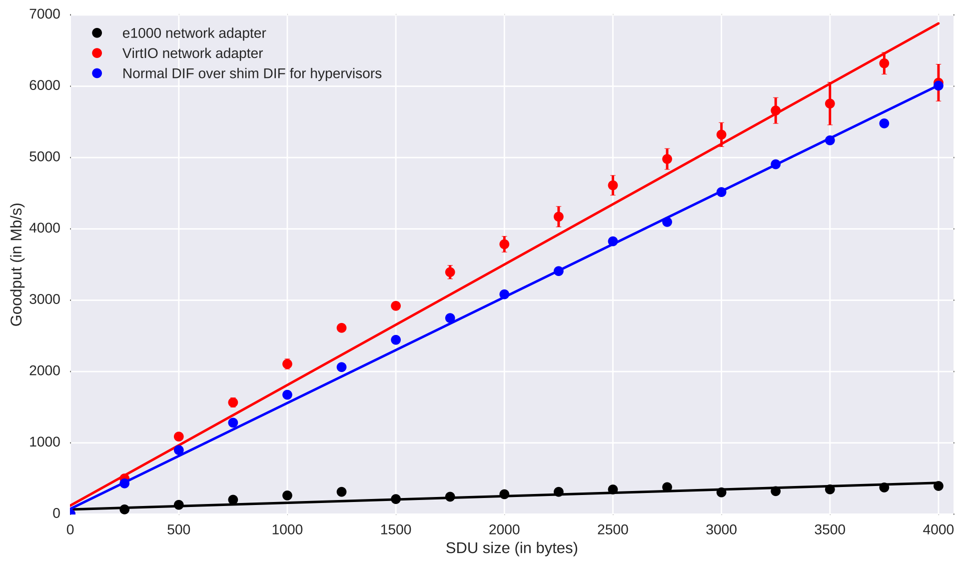
As shown in Figure 4a, the shim DIF for hypervisors outperforms both e1000 and virtio-net NIC setups, which validates that a simpler and cleaner architecture allows for better performance, even with an unoptimized prototype. Note that while the Ethernet MTU is set to 1500 in the first two sessions, it is possible to go beyond the limit because of the UDP Segmentation Offloading (UFO) feature which is supported by both e1000 and virtio-net models. This feature allows the NIC (real hardware or emulated) to accept UDP packets that do not fit into a single 1500 bytes Ethernet frame. A real NIC performs the necessary segmentation in hardware, while an emulated one (e.g. e1000) emulates the segmentation in software. It's interesting to note that in the virtio-net case, this segmentation is not really carried out, since there is no real Ethernet cable to deal with, but the oversized packet is directly forwarded to the host stack, which is able to process and deliver to the receiving application (netserver) without further segmentations. This is clearly an optimization made possible by paravirtualization, but can also be seen as a workaround that is not necessary in our solution.

Next, similar goodput performance measurements were taken on the intra host VM-to-VM scenario. Again, three test sessions were performed, the first two for traditional VM networking and the third one for the IRATI stack. The setup of the first two sessions is very similar to the corresponding one in the host-to-VM scenario. The VMs are given an emulated NIC, whose corresponding tap device is bridged to the host stack through a Linux in-kernel software bridge (see Figure 3). Both VMs and the bridge interface are given an IP address on the same subnet. Measurements are again performed with the netperf utlity, with the netperf server running on a VM and the netperf client running on the other VM.

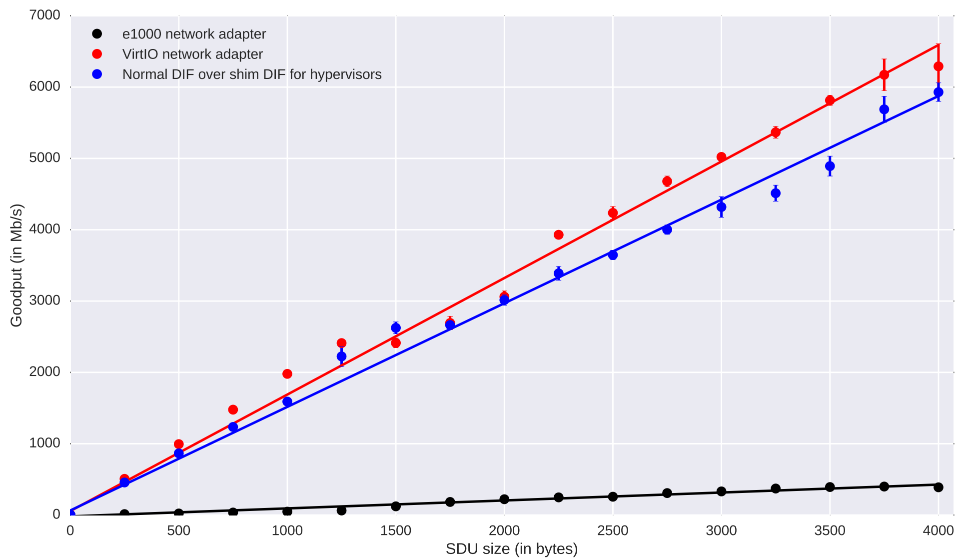
In the case of the IRATI tests, point-to-point connectivity between host and VM is provided by the shim DIF for hypervisors. A normal DIF is overlayed on these shim DIFs to provide connectivity between the two VMs. Tests are performed again with the rina-tgen application, using a flow that provides flow control without retransmission control. Flow control is used so that the receiver's resources are not abused. Retransmission control is not needed since no SDUs can be lost along the datapath - since in this very specific tests we are sure the SDU never leaves the host or is dropped in some intermediate queue. In TCP/IP, this kind of functionality, flow control without retransmission control, is not available. Hence we chose again UDP to perform the tests for the traditional networking solution, since its functionality is most similar. The result of these test sessions are depicted in Figure



(a) Host to VM communication



(b) VM to VM communication intra host



(c) VM to VM communication inter host

Fig. 4. Goodput of different network virtualization technologies

TABLE I
COMPARISON BETWEEN VM NETWORKING IN TCP/IP AND RINA

VM networking in TCP/IP	VM networking in RINA
Checksumming is always performed	Only checksumming if needed
Header length >= 84	Header length >= 4
Hardware offloading required for performance	No hardware offloading needed
MAC address to be unique in whole data center	VMPI-id to be unique per OS
SDU size restricted by Ethernet standard	No restrictions on SDU size
Complete NIC to be implemented	Simple device to be implemented
Hard to configure	Easy to configure
Ad-hoc components are needed	VM networking is part of the architecture
Code size increased due to configuration	Most code is related to the data path

4b, again with their respective 95% confidence intervals and a first degree polynomial regression line. Full virtualization again performs poorly. The paravirtualized solution currently outperforms the unoptimized IRATI stack, because a normal DIF is used to provide the connectivity between the VMs. This part of the IRATI stack is the least optimized for performance, which explains why the IRATI prototype performs worse than TCP/IP in this case.

Finally, inter host VM to VM tests were performed. The hosts are connected through a 10 Gbps link. The setup of the first two sessions now differs in the fact that both hosts have to setup a Linux bridge. Each VM is given an emulated NIC, whose tap device is bridged to the host stack through a Linux in-kernel software bridge. On both hosts, the interface connecting the host to the other host is also added as an interface to the bridge. In this way, the VMs are connected to each other. The VMs are assigned an IP address in the same subnet, and all NICs are enabled to use jumbo frames. Once more we used netperf to test the goodput between the VMs.

For the IRATI prototype, the setup is identical to what is depicted in Figure 1. Point-to-point connectivity between host and VM is provided by the shim DIF for hypervisors. Point-to-point connectivity between the hosts is provided by the shim DIF for IEEE 802.1Q. The NICs on the hosts have jumbo frames enabled. On top of these shim DIFs, a normal DIF is overlayed. Tests are performed using the rina-tgen application. These test sessions' results are shown in Figure 4c, with their respective 95% confidence intervals and a first degree polynomial regression line. Full virtualization performs similar to the previous test sessions. Both the paravirtualized solution and the IRATI prototype solution achieve a performance that is similar to the one in the previous test session. The paravirtualized solution again outperforms the unoptimized IRATI stack, since the IRATI prototype also uses a normal DIF in this test session, which is the bottleneck for performance.

VI. FUTURE WORKS

We plan to optimize the IRATI stack to achieve better performances by reducing buffer copies and allocations to the bare minimum to improve the performance when communicating between VMs.

VII. CONCLUSION

In this paper, we illustrated how paravirtualization can be leveraged by the Recursive InterNetwork Architecture (RINA), a network architecture *ab initio*. We presented the shim DIF for hypervisors as an alternative to traditional networking solutions in Virtual Machines, which has been implemented in the IRATI prototype.

We explained how the shim DIF for hypervisors is a more manageable solution for Virtual Machine networking. A summary of the main differences between VM networking in TCP/IP and VM networking in RINA can be found in Table I. We also showed how it already allows for good performances in some reference scenarios, host-to-VM and VM-to-VM, despite being unoptimized.

ACKNOWLEDGMENT

This work is partly funded by the European Commission's Seventh Framework Programme (FP7/2007-2013) through the projects IRATI (Grant 317814), part of the Future Internet Research and Experimentation (FIRE) objective and IRINA, part of the GN3plus Open Calls.

REFERENCES

[1] L. Rizzo, G. Lettieri, and V. Maffione, "Speeding up packet i/o in virtual machines," in *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*. IEEE Press, 2013, pp. 47–58.

[2] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

[3] (2015, Nov.) The VirtualBox website. [Online]. Available: https://www.virtualbox.org/

[4] (2015, Nov.) The VMware website. [Online]. Available: http://www.vmware.com/

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[6] "Performance Evaluation of VMXNET3 Virtual Network Device." VMware, inc., Tech. Rep., 2009.

[7] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[8] J. Day, *Patterns in network architecture: a return to fundamentals*. Prentice Hall, 2008.

[9] J. Day, I. Matta, and K. Mattar, "Networking is IPC: a guiding principle to a better internet," in *Proceedings of the 2008 ACM CoNEXT Conference*. ACM, 2008, p. 67.

[10] S. Vrijders, E. Trouva, J. Day, E. Grasa, D. Staessens, D. Colle, M. Pickavet, and L. Chitkushev, "Unreliable inter process communication in Ethernet: migrating to RINA with the shim DIF," in *5th International Workshop on Reliable Networks Design and Modeling (RNDM-2013)*, 2013, pp. 97–102.

[11] S. Vrijders, D. Staessens, D. Colle, F. Salvestrini, E. Grasa, M. Tarzan, and L. Bergesio, "Prototyping the Recursive Internet Architecture: the IRATI project approach," *Network, IEEE*, vol. 28, no. 2, pp. 20–25, 2014.

[12] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer." in *Hotnets*, 2009.

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

[14] (2015, Nov.) RINA traffic generator. [Online]. Available: http://www.github.com/IRATI/traffic-generator