# Diverter: A New Approach to Networking Within Virtualized Infrastructures

Aled Edwards, Anna Fischer, Antonio Lain

HP Laboratories
HPL-2009-231

**Keyword(s):**
Network virtualization, distributed overlays, packet filtering, routing

**Abstract:**

As virtualized data-centres become the back-end platforms behind a new generation of utility and cloud computing infrastructures (such as AmazonAWS [1]) their multi-tenancy, scale and complexity introduce new challenges that especially affect the networking layer. Multiple customers' requirements for varying logical network topologies must be simultaneously accommodated on the shared, underlying network fabric in a secure manner. Diverter is a new approach to network virtualization that targets these highly flexible, large-scale, multi-tenanted environments and advances the current state-of-the-art by implementing an efficient, fully distributed virtualized routing system that allows end-to-end communication between any endpoint with just a single network "hop". We have implemented a prototype of this solution that, in certain network configurations, achieves a throughput improvement of at least 66 % compared to alternative approaches.

External Posting Date: September 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: September 6, 2009 [Fulltext]

Published and presented at Workshop: Research on Enterprise Networking (WREN'09/SIGCOMM'09), Barcelona, Spain, August 17-21, 2009.

# Diverter: A New Approach to Networking Within Virtualized Infrastructures

Aled Edwards, Anna Fischer and Antonio Lain
HP Laboratories, Long Down Avenue, Stoke Gifford, Bristol BS34 8QZ, UK
{aled.edwards,anna.fischer,antonio.lain}@hp.com

## ABSTRACT

As virtualized data-centres become the back-end platforms behind a new generation of utility and cloud computing infrastructures (such as AmazonAWS [1]) their multi-tenancy, scale and complexity introduce new challenges that especially affect the networking layer. Multiple customers' requirements for varying logical network topologies must be simultaneously accommodated on the shared, underlying network fabric in a secure manner.

Diverter is a new approach to network virtualization that targets these highly flexible, large-scale, multi-tenanted environments and advances the current state-of-the-art by implementing an efficient, fully distributed virtualized routing system that allows end-to-end communication between any endpoint with just a single network "hop". We have implemented a prototype of this solution that, in certain network configurations, achieves a throughput improvement of at least 66 % compared to alternative approaches.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*distributed networks*

## General Terms

Design, Management

## Keywords

Network virtualization, distributed overlays, packet filtering, routing

## 1. INTRODUCTION

Over the past years data-centres have become the main infrastructures behind large-scale Internet and enterprise computing, and as customers embrace utility and cloud computing models (such as AmazonAWS [1]), the scale of data-centre platforms is set to increase further. In order to provide better resource utilization and more flexible services to such customers, many data-centres are moving towards multi-tenanted, virtualized environments. For example, Figure 1 shows potentially competing customers that can safely manage the topology of their (virtual) networks while sharing physical infrastructure provided by the data-centre owner. While these virtualized environments offer many benefits, they also represent very complex and dynamic systems, and so pose significant management and control challenges.

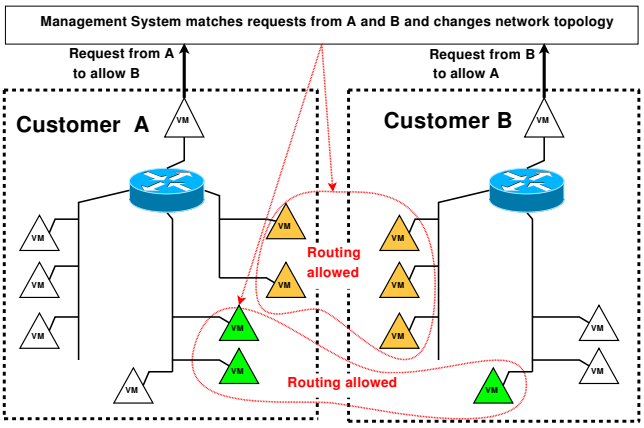


**Figure 1: Negotiating network topology changes in a virtualized data-centre.**

Challenges include for example:

- allowing multiple customers to have distinct, separate virtual networks that they can configure and tune individually to suit their particular (multi-subnet) application requirements,
- allowing many of these virtual networks to co-exist within a shared underlying fabric, whilst retaining separation and isolation between different (potentially competing) customers - by default no traffic from one customer's virtual network should be allowed into another customer's virtual network,
- allowing controlled and efficient inter-communication within virtual networks and between distinct customer virtual networks if required and permitted,
- scaling this network virtualization to thousands of customers and thousands of endpoints, each hosting tens of virtual machines.


*WREN'09,* August 21, 2009, Barcelona, Spain.

### *Limitations of traditional approaches*

Traditional approaches to virtual networking [6, 12] struggle to address the challenges mentioned above. Typically, they rely on network models that provide datalink layer or layer-2 [19] abstractions such as virtual segments or virtual subnets. Unfortunately, these approaches require routing elements to convey packets between virtual networks. These routing elements could be hardware routers or even routing logic within a multi-interface customer Virtual Machine (VM). Either way, these routing elements must be dynamically configured in response to changing virtual network topologies.

The use of hardware routers introduces significant portability, reliability, scalability and manageability issues. For instance:

- routers and physical nodes hosting VMs are often administered by different teams and this separation of concerns can make acceptance and implementation of an approach that reconfigures routers difficult. In the case of installations using shared networking infrastructure, administrators are understandably reluctant to allow direct access to networking equipment,
- there is still no widely-adopted direct interface and data-model for programming heterogeneous hardware switches and routers - this limits the portability of any solution to networks with supported switches and routers,
- conventional hardware routers also limit scalability: they only "understand" VLAN [9] encapsulation, and the use of VLANs limits the number of separate subnets to 4096 (since VLAN ids are 12-bits wide), not enough when millions of customers could require a private subnet. Note, this limitation does not apply to very modern routers which support so-called QinQ VLAN encapsulation [10], but QinQ also requires NIC support.

The multi-interface routing VM approach addresses some of these limitations but exhibits other problems. For instance:

- extra "routing" VMs are required in every customer virtual network that needs multiple subnets to, for example, deploy a traditional three-tier web application,
- all packets travelling inter-subnet or to other customer networks need to pass through the routing VMs, introducing a severe performance bottleneck (see Section 3), and also a single point of failure.

### *Our approach*

In order to overcome these difficulties the **main contribution** of our work is the design and implementation of a physical node, software-only network virtualization solution based on a fully-distributed virtualized routing system that facilitates single network "hop" communication between any endpoint, thus providing very good performance without compromising manageability.

Unlike traditional software solutions, our solution adopts a network layer or layer-3 [19] approach to virtual networking - nodes hosting VMs collaborate to implement a fully-distributed virtual router.

IP addresses of VMs are allocated in a way that reflects virtual network topology and such that no address clashes occur. This means no two customers ever need to use NAT techniques to communicate.

Our solution only requires software changes in physical nodes hosting VMs. It assumes a flat underlying layer-2 network and thus avoids having to configure switches or routers. Unlike AmazonAWS, we correctly support scoped broadcast delivery using physical network multicast, which most switches implement efficiently (by IGMP snooping). Whilst we acknowledge that there are scalability and performance concerns with large layer-2 networks due to packet broadcasting and packet flooding, we note that layer-2 manageability benefits mean that these issues are being addressed in contemporary research [13, 16]. Also, our solution includes a number of measures to reduce or even eliminate the use of broadcast and flooding.

The rest of this paper is organized as follows: In Section 2 we describe details of our virtual networking solution. We present results from a first prototype implementation in Section 3, demonstrating the positive effect on performance achieved by our technology compared to alternative solutions. Section 4 discusses related work and in Section 5 we summarize our overall approach and its practical implementation, and additionally sketch some first ideas on future work. Note that for the sake of brevity we do not describe here a distributed policy engine that we use to configure endpoints.

## 2. VIRTUAL NETWORKING

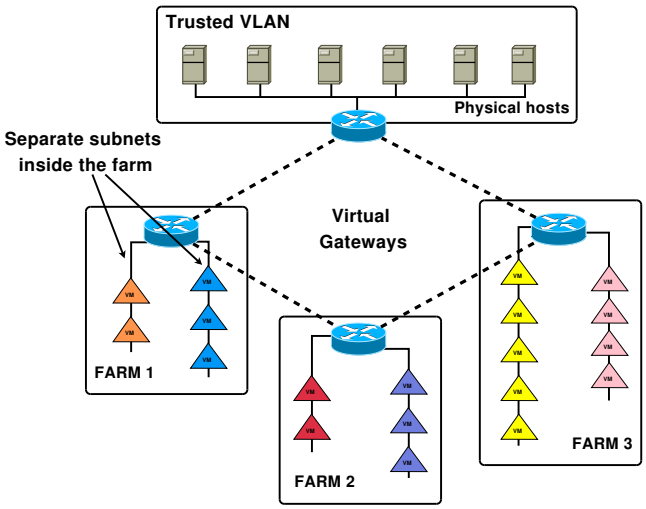


**Figure 2: Distributed Virtual Router: A logical view.**

We virtualize based on layer-3 (network-level) information and provide support for multiple, isolated, independently managed customer virtual networks or "farms" each consisting of several subnets. This is a common configuration in non-virtual settings. Figure 2 shows the logical view of a small system.

By default, our system mimics the endpoint visibility provided in typical physical networks - VMs inside a subnet can communicate with each other without any restrictions, but

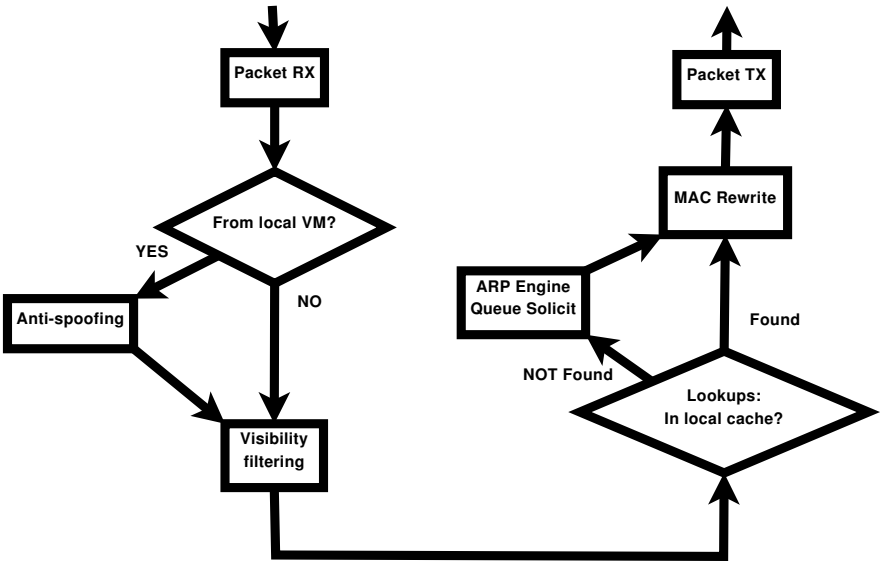


**Figure 3: Packet Processing Steps.**

communication between VMs of different subnets has to be explicitly allowed by the farm owner. Communication across two different farms is only permitted if both sides have mutually agreed on such a communication.

VM IP addresses are allocated in a regular manner using a simple one-to-one mapping between network IP addresses and $< f : s : h >$ triplets, where $f$ is the farm identifier, $s$ is the subnet identifier relative to that farm, and $h$ is the host identifier within the subnet. We currently use $10.f.s.h$ addresses, allowing up to 16 million VMs system-wide, but this can be changed by the infrastructure owner at set-up time.

Notionally, at the core of each farm network is a virtual gateway connecting all the subnets within the farm. However, these virtual gateways do not actually exist; their functionality is distributed amongst all the physical nodes hosting VMs, which collaborate to form a fully-distributed virtual router for all farms. This fully-distributed virtual router determines where to send packets by examining virtual IP addresses alone, ignoring the virtual MAC addresses used.

Our network virtualization technology is implemented by a software module (which we call VNET) residing within the host OS on each physical node. It intercepts packets as they emit from VMs or the host OS and as they arrive from the wire, processes them, discards them if they violate packet filtering rules and ultimately transmits them to another physical node or passes them into a local VM or the host OS.

### *Basic packet processing*

Whenever a packet arrives from a physical NIC or one of the virtual NICs on the system, it is passed to our VNET module. A number of processing steps then occur, as visualized in Figure 3:

1. Any locally-sourced packet first encounters an anti-spoofing check that ensures the sending VM is using valid source addresses - this prevents a VM impersonating another VM.
2. All packets then traverse a filter that contains all network visibility rules that enforce separation between virtual machines. If the packet does not pass the rule checks, it is discarded.
3. The destination IP address in the packet is then examined to determine where to send it. The VNET module manages a mapping table within each host OS that maps IP addresses to local virtual NICs or to physical MAC addresses of remote nodes.
4. If the packet is destined for a local VM, the VNET module checks to see whether the source MAC address needs rewriting - this might be to preserve the illusion of traversing a gateway or to recreate the source virtual MAC address that the packet originally possessed.
5. If the packet is destined for a remote node the destination MAC address is rewritten to be the physical MAC address of the remote node.
6. If no mapping exists in the table then the packet is queued until a mapping can be determined by an associated ARP engine which builds up and manages the mapping table.

In our approach we rewrite MAC addresses such that no virtual MAC addresses ever appear "on the wire". Our network virtualization technology translates virtual MAC addresses into the MAC address of the physical node that currently runs the VM with the target IP and rewrites Ethernet packet headers accordingly before sending them onto the wire.

We exploit this MAC rewriting technique for providing the illusion of routing through our notional virtual gateways. Within each virtual subnet in each virtual farm a specific MAC and IP address is reserved for the farm's virtual gateway (conventionally address $< f : s : 1 >$). Whenever

packets are sent from one subnet to another, on the receiving node the source MAC address is rewritten to be the MAC address of the gateway - this creates the illusion that the packet has traversed the gateway and has been routed. MAC rewriting is also used to limit broadcast and multicast distribution as described later.

### *Advantages of our approach*

The main benefit of our network virtualization approach is that all communication within our virtualized infrastructure always involves just a single network "hop". As the underlying physical infrastructure is a flat, layer-2 network, our fully-distributed router can quickly discover the location of (virtual) IP addresses and pass packets directly to their required destination. This has a huge, positive performance impact, especially for inter-farm and inter-subnet communication as we will see in Section 3. As we operate directly on layer-3 network-level information, and transmit packets directly from source to destination, we can avoid deploying and configuring dedicated routing entities (like routing VMs) which is an important advantage in terms of performance and manageability.

The use of system-allocated IP addresses avoids clashes - this means that two customer farms, if required, can always communicate directly, without the use of NAT techniques. We are also able to exploit knowledge of IP address structure to limit broadcast and multicast distribution (as described later).

MAC rewriting has three benefits - the first is that we reduce the occupancy of physical switch forwarding tables - given that our addressing scheme can potentially accommodate very many VMs simultaneously, the sheer number of MAC addresses could cause a significant problem for conventional physical switches which typically can only hold thousands of entries. The second benefit is that each physical node no longer needs to listen in promiscuous mode (i.e., receiving all packets which arrive at its NICs) which means that physical nodes need not be interrupted and need not examine every flooded packet on the network. A third benefit is we avoid encapsulating the packet thus allowing VMs to use the full MTU of the physical network. A further benefit is that by using physical MACs only, we avoid "leaking" virtual address information onto the wire.

A further important differentiator of our networking approach is that as well as supporting "infrastructural" policies that define which VMs can communicate with which other VMs, our system allows customers to define their own per-VM policies - this allows customers to specify their own IP filters, stateful firewalls, etc. However, crucially, because these policies are implemented outside the customer VM within our virtualization layer, there is no way in which they can be bypassed or subverted even if the VM is compromised in some way. This, of course, does not prevent a customer using filters within their VM if they wish to do so.

### *The VNET ARP engine*

The VNET module uses multicast ARP to discover mappings between IP and MAC addresses, and to discover the physical node on which a VM is running (inferred from the source MAC in ARP response packets). It also manages the VNET mapping table, refreshing stale entries, and timing out unused entries. There is a single, shared mapping table per-node. This allows VMs to reuse mappings previously discovered by the actions of other local VMs. Normally ARP is used to discover the MAC address corresponding to destination IP addresses. However, in our system ARP may also be used to discover the MAC addresses of new **source** IP addresses. This is necessary after a packet has traversed the physical network - in this case the virtual source and destination MAC addresses will have been rewritten to be the physical MAC addresses of the sending and receiving nodes and so a "backward" ARP may be required to discover and recreate the original virtual source MAC address.

The mapping table is limited to a fixed size. A linked list is maintained within it, allowing the least-recently-used entry to be discarded and reused when the mapping table is full. However, as we operate on an underlying flat layer-2 network (i.e. with no default route) the mapping table has to be sized significantly larger than a typical ARP table in a typical physical node. This is not a significant scaling problem as, normally, the number of endpoints in active use by the VMs on any given node is a small subset of the total number of VMs in the system. Also the size of a mapping table entry is relatively small, a single megabyte of physical memory accommodating thousands of mapping entries.

### *Virtual IP broadcast and IP multicast traffic handling*

Each virtual farm within our system may consist of multiple virtual subnets. In a physical network, broadcasts are normally restricted to their originating subnet. Similarly IP multicasts are restricted to their originating subnet unless their TTL is $> 1$. To provide similar behaviour within our virtual networks each virtual farm is allocated a fixed Ethernet multicast address, and each subnet within each virtual farm is allocated a fixed Ethernet multicast address. Broadcast and multicast traffic emitting from VMs is then mapped to the appropriate Ethernet multicast address depending on whether the packet is broadcast or multicast.

The VNET module on each physical node listens on the appropriate farm and subnet multicast addresses corresponding to each of the VMs hosted. Thus only nodes that possess a VM in farm X are interrupted with virtual broadcasts or virtual multicasts from farm X. The VNET ARP engine also uses this technique to limit the distribution of ARP request packets - if it is attempting to determine the MAC address of a VM in farm X subnet Y it sends the ARP packet to the Ethernet multicast address corresponding to X and Y.

### *Virtual ARP handling*

The VNET ARP engine also handles ARPs issued from VMs. There are three possibilities that need to be addressed: firstly, the IP address being resolved may belong to the same subnet as the VM doing the resolving. In this case VNET consults its mapping table to find a virtual MAC address corresponding to the virtual IP address being sought. If a mapping exists VNET responds immediately with the looked-up MAC address. If no mapping exists, VNET sends an Ethernet multicast ARP as described above. On receipt of an ARP response for the IP, VNET updates its mapping table with the new binding, and passes the ARP response into any requesting VM. Secondly, the IP address being resolved may belong to a different subnet. In this case VNET discards the ARP request, but may ARP respond with the MAC address of the notional virtual gateway if the farm is configured to use ProxyARP. Thirdly, the IP address being resolved may be that of the notional virtual gateway itself. In this case

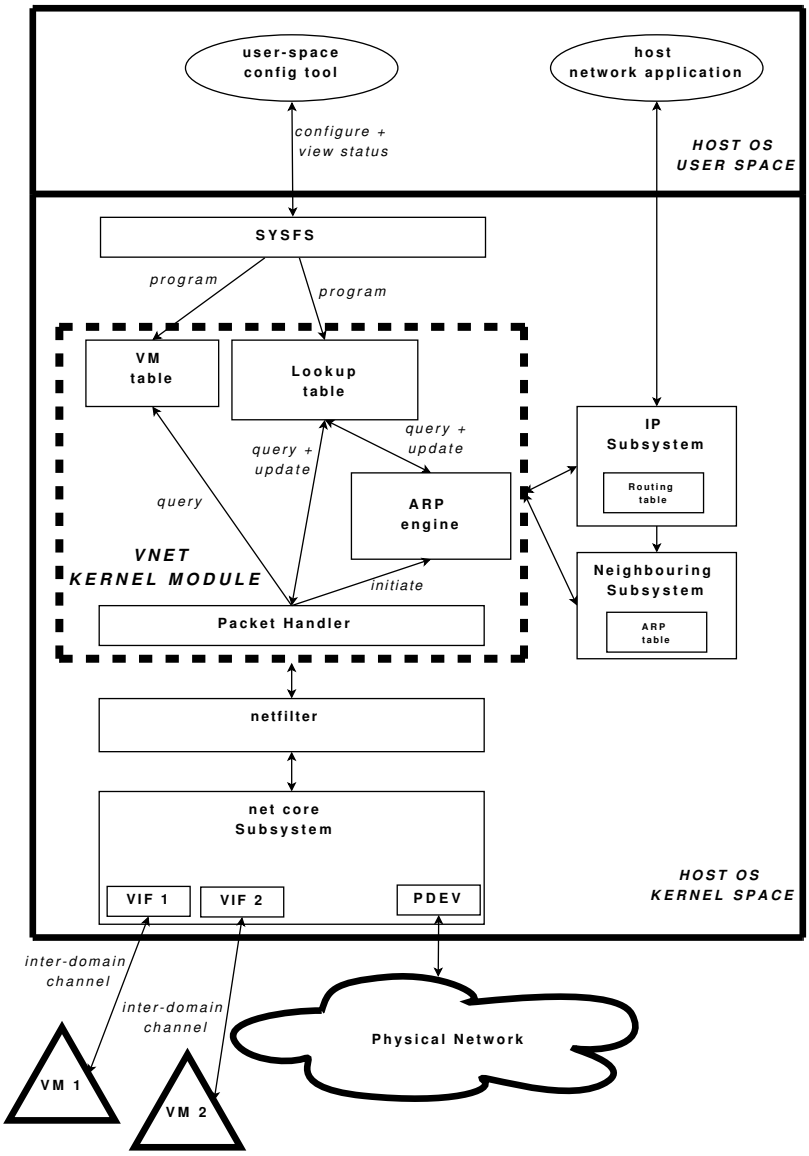


**Figure 4: VNET kernel module integration within Linux.**

VNET ARP responds immediately with the MAC address of the notional virtual gateway.

### *Virtual DHCP and other configuration*

The VNET module responds to DHCP broadcasts locally, synthesizing DHCP responses to program the VM with its IP address - this also allows the VNET module to configure the gateway which the VM should use (in our case the farm's notional virtual gateway), and also to configure other addresses, such as the DNS server the VM should use. Most operating systems (within the VMs) honour this information and dynamically configure their environment to use this gateway/DNS address information.

### *Broadcast and flood avoidance*

As described above, Diverter operates on a flat layer-2 network. As such, there are potentially a number of issues which might limit scalability of our system. The primary issues are the use of broadcast and flooding. In a flat layer-2 network broadcasts from any node are delivered to **all** nodes - also packets sent to unknown MAC addresses are flooded to **all** nodes. Floods are slightly less bothersome as NICs typically discard flooded packets (assuming the NICs are not in promiscuous mode) - however they do traverse all switches in the layer-2 network and thus consume switch bandwidth, switch processing resources and link capacity. Broadcasts are more problematic. As well as consuming resources in all switches and on all active links, broadcasts are also received by **all** nodes. Thus, all nodes potentially receive an interrupt for broadcast packets and have to devote processing time to examining each broadcast and typically, discarding it.

These overheads limit the scale of layer-2 networks and normally a layer-2 network does not exceed 1000 nodes. In

our system we intend to use a large flat layer-2 network in order to avoid the need to configure switches and routers and thus to benefit from improved manageability of the network. Thus we have taken a number of steps to reduce or eliminate the use of broadcast and flooding.

The primary users of broadcast in a typical system are ARP and DHCP. The system DHCP server we use is configured to issue IP addresses with long lease-times so typically physical nodes in our system only DHCP broadcast a few times a day. We employ a separate out-of-band mechanism to distribute information about physical nodes, and this information can be used to setup static ARP mappings, eliminating physical node ARP broadcasts altogether.

The virtual machines in our system still emit broadcast DHCP requests but these are intercepted by the VNET module and a local response synthesized as described above. Broadcast ARP requests emitted by VMs are also intercepted by the VNET module; these are either resolved from the local mapping table or transformed into ARPs to a more limited Ethernet multicast address corresponding to the farm and subnet of the VM whose address is being resolved. Taking things one step further, address mappings within the system mapping table are periodically refreshed but VNET always attempts unicast to the previous resolved location before resorting to multicast.

Floods are less problematic as indicated earlier. In our system virtual MAC addresses never appear on-the-wire so all we have to concern ourself with is flooding to unknown physical MAC addresses. Physical nodes can arrange to emit at a low rate (but less than the 'ageing' timeout) a small unicast to a guaranteed non-existent destination MAC address. This would be enough to keep physical MAC addresses "alive" in all switches, limiting flooding, whilst only consuming a tiny, fixed fraction of available bandwidth.

### *Prototype Implementation*

In our current prototype implementation the VNET software is implemented as a Linux dynamically-loaded kernel module. Figure 4 visualizes some of the main components of the VNET kernel module and shows how they integrate with the rest of the Linux kernel subsystems. The VNET module uses as much of the available standard kernel features as possible: it hooks into the Linux IP routing and neighbouring subsystems, mainly to learn IP-to-MAC address bindings that the host OS has already discovered and synchronize those with the shared global lookup table. It listens on the virtual network interfaces (*VIFs*) that send and receive traffic to and from local virtual machines, and additionally intercepts packets coming from the physical network (through *PDEV*) and also from the host OS IP stack (through hooking into the IP routing table). Figure 4 also demonstrates that all packets get filtered by the traditional Linux kernel netfilter infrastructure before they enter the VNET kernel module (if they pass all filtering rules).

## 3. RESULTS

In this section we evaluate the performance of our VNET implementation on a Xen-based platform [3] and compare it to alternative approaches. Similar results were also obtained for a VMware Server based [17] prototype.

We have compared our new solution, i.e., **Diverter**, with the following alternative approaches:

**Xen BRG** Xen configured in standard bridging mode.

**EtherIP** An implementation of EtherIP encapsulation [8] developed within the HP Labs SoftUDC project [12].

**VLAN tagging** A VLAN tagging implementation developed within HP Labs (OpenTC project) [6, 14].

These represent the most commonly used network virtualization technologies on the market today. We have two main objectives for the performance evaluation. We would like to demonstrate that the packet processing performed in our solution, such as address rewriting and table lookups, does not cause performance degradation compared to other approaches and, our *one-hop* L3 network virtualization introduces a significant performance advantage over other approaches whenever network traffic has to travel across one or more virtual network segments.

We use two test scenarios for our comparisons: **intra-subnet** tests show how the solutions perform when communicating VMs reside on the same virtual network segment while **inter-subnet** tests demonstrate what performance can be achieved when communication has to be routed over multiple segments.

We obtained throughput measurements using the *netperf* benchmark tool and latency results using the *ping* tool. Our test systems run Xen 3.0.4 (release version) with a 2.6.16.33 Linux kernel in domain 0. We used HP ProLiant BL25p G2 blade servers each fitted with two AMD Opteron processors running at 2 GHz, 8GB system memory and a Gigabit Ethernet card. We have configured *netperf* with a confidence level of 99% and a confidence interval of 5% to ensure that our results only include consistent measurements. Throughput results have been recorded for a *netperf TCP_STREAM* with a message size of 8192 bytes and a socket size of 65536 bytes.

### 3.1 Intra-subnet Communication

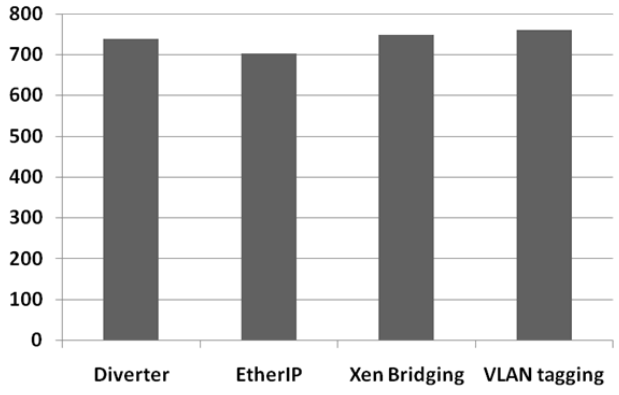


**Figure 5: Intra-subnet throughput results (Mbps).**

Figure 5 shows raw network throughput results in Megabits per second (Mbps) for communication between VMs that are hosted on the same subnet. The results show that in this scenario there is no significant performance difference between the technologies. This is mainly due to similarities in packet handling implementations.

**Table 1: Ping Intra-subnet Round-Trip (msec).**

| | Min | Avg | Max | Mean Dev |
|---|---|---|---|---|
| Xen BRG | 0.1355 | 0.18 | 0.294 | 0.0235 |
| Diverter | 0.1582 | 0.2062 | 0.3182 | 0.0276 |
| VLAN | 0.1395 | 0.21225 | 0.35675 | 0.0295 |
| EtherIP | 0.151 | 0.246 | 0.378 | 0.0335 |

Table 1 lists latency measurements recorded using the Linux *ping* tool. Each test run sent 1000 ICMP ping packets and was repeated several times for each approach. In this scenario, latency is very similar for all solutions. A more detailed intra-subnet performance evaluation can be found in [5].

## 3.2 Inter-subnet Communication

In this section we outline how our new layer 3 approach can have a significant positive impact on overall network performance compared to solutions that deploy dedicated routing VMs. The tests demonstrated here do not include approaches deploying **VLAN tagging** or **Xen Brg**. We expect that those would use high-performance physical layer 3 switches that route packets between virtual network segments and in that case **Diverter** does not have any performance advantage. However, we have already emphasized the drawbacks of implementing highly-flexible virtual network routing in hardware network devices in Section 1.

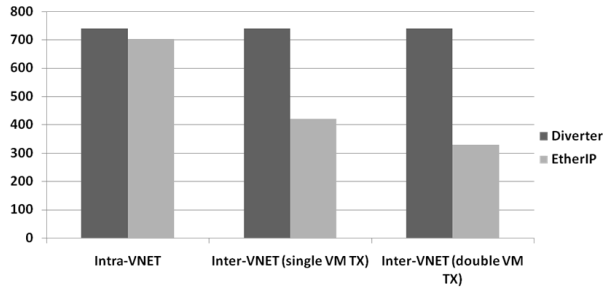


**Figure 6: Inter-subnet throughput results (Mbps).**

This test compares our **Diverter** approach with one that uses a remote VM to route **EtherIP** packets between virtual network segments. Figure 6 compares raw throughput results for the following cases: (1) intra-subnet VM communication (as described in the previous section) (2) inter-subnet communication between two VMs and (3) inter-subnet performance when two pairs of VMs are transmitting at the same time (using the same routing VM in the **EtherIP** case). Note how **Diverter** ensures that throughput performance is consistent - this is because even when traffic has to pass multiple virtual subnets, **Diverter** implements a *one-hop* communication path. In contrast, using a routing VM with **EtherIP** affects throughput significantly and we see a **40%** reduction in case (2) and a **60%** reduction in case (3) (a throughput improvement for **Diverter** of 66 % and 150 % respectively). The significant drop in network performance for **EtherIP** is mainly due to the fact that packets have to traverse the routing VM which introduces virtualization overhead (i.e. network I/O processing and context switches). Note that throughput continues to decrease significantly as further VMs attempt to communicate via the routing VM which very quickly becomes the bottleneck of inter-subnet networking.

**Table 2: Ping Inter-subnet Round-Trip (msec).**

| | Min | Avg | Max | Mean Dev |
|---|---|---|---|---|
| Diverter | 0.1582 | 0.2062 | 0.3182 | 0.0276 |
| EtherIP | 0.287 | 0.37 | 0.4975 | 0.387 |

Similarly, Table 2 shows a much higher latency when a routing VM is required, even with almost idle CPU loads, in fact doubling the average round-trip time.

# 4. RELATED WORK

Overlay networks allow the creation of a virtual topology on top of an existing infrastructure. Overlay networks have been an extremely useful tool for the development of the Internet since its foundation. For example, experimental features like Internet multicast [7] used an overlay to ensure network stability. There are many different implementations of overlays targeting strong isolation [8, 9], improved reliability [2], custom manageability [4] or just better network performance [15]. Implementations also differ on whether the overlay context is explicitly added to network packets. Explicit context typically requires some form of packet encapsulation, for example, VLAN tagging [9] uses a small frame header to provide a virtual subnet identifier, IP-in-IP schemes [7] add one or more extra IP headers per IP packet, Ethernet-in-IP [8] adds an IP header to an Ethernet frame - to allow tunnelling non-IP protocols over IP networks-, and others like RON [2] add a custom packet header. In contrast, implicit overlay context uses some conventions understood by all the endpoints: for example, PlanetLab [4] negotiates bindings from socket port numbers to overlays to reuse IP addresses. Other implementations use a combination of explicit and implicit context: for example, Violin [11] uses UDP tunnelling on top of a PlanetLab overlay to provide better IP address space isolation. We favor implicit context in our approach and by mapping IP addresses into $< f : s : h >$ triplets we achieve strong isolation without encapsulation.

A critical difference between our approach and existing overlay network technologies is the way we implement routing between virtual network segments (or subnets). The primary focus of most existing overlay technologies is to efficiently implement a layer 2 abstraction and use external elements like routing VMs [6, 11] or VLAN-aware physical routers [18] to create a (managed) layer 3 abstraction. Instead, we just focus on creating a single, fully-distributed virtual router that gives us directly layer 3 capabilities. We have argued the benefits of our approach when either routing performance or administrative difficulties causes problems for conventional approaches.

# 5. CONCLUSIONS

We have identified performance and manageability issues that traditional layer 2 approaches to virtual networking exhibit in supporting a new generation of large-scale virtual-

ized data-centres. Instead, we have implemented a fully-distributed virtual router abstraction that solves some of these issues.

Our current focus is on implementing support for IPv6, enhancing virtual network performance by leveraging new hardware virtualization features of CPUs and chipsets, switches and network cards, and adding Quality-of-Service guarantees to virtual networking for better network resource control and resilience to Denial-of-Service attacks.

## Acknowledgements

We would like to thank HP intern Thom Haddow for implementing a custom IPTables matching module. HP colleagues Eric Deliot, Alistair Coles, Mike Wray, Peter Toft and Patrick Goldsack provided many insights during relevant discussions. Nigel Edwards, Paul Congdon and Jeff Mogul (also from HP) and Jennifer Rexford (Princeton University) gave us useful advice on how to improve this paper.

## 6. REFERENCES

[1] Amazon.com. Amazon.com - Amazon Web Services. `http://aws.amazon.com`.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *SOSP*, pages 131–145, 2001.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. SOSP 2003, November 2003.

[4] A. C. Bavier, M. Bowman, B. N. Chun, D. E. Culler, S. Karlin, S. Muir, L. L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services. In *NSDI*, pages 253–266. USENIX, 2004.

[5] S. Cabuk, C. I. Dalton, A. Edwards, and A. Fischer. A Comparative Study on Secure Network Virtualization. Technical Report HPL-2008-57, HP Labs, 2008.

[6] S. Cabuk, C. I. Dalton, H. V. Ramasamy, and M. Schunter. Towards automated provisioning of secure virtualized networks. ACM CCS, 2007.

[7] H. Eriksson. Mbone: The Multicast Backbone. *Communications of the ACM*, 37(8):54–60, 1994.

[8] R. Housley and S. Hollenbeck. EtherIP: Tunneling Ethernet Frames in IP Datagrams, September 2002. RFC 3378.

[9] IEEE. Virtual Bridged Local Area Networks. Technical Report ISBN 0-7381-3662-X, IEEE, 2003.

[10] IEEE. Provider bridges, IEEE Standard 802.1ad. IEEE Standards, 2006. `http://www.ieee802.org/1/pages/802.1ad.html`.

[11] X. Jiang and D. Xu. VIOLIN: Virtual Internetworking on Overlay Infrastructure. In J. Cao, L. T. Yang, M. Guo, and F. C.-M. Lau, editors, *ISPA*, volume 3358 of *Lecture Notes in Computer Science*, pages 937–946. Springer, 2004.

[12] M. Kallahalla, M. Uysal, R. Swaminathan, D. E. Lowell, M. Wray, T. Christian, N. Edwards, C. I. Dalton, and F. Gittler. SoftUDC: A Software-Based Data Center for Utility Computing. *Computer*, 37(11):38–46, 2004.

[13] C. Kim, M. Caesar, and J. Rexford. Floodless in Seattle: a scalable ethernet architecture for large enterprises. In *Proceedings of the ACM SIGCOMM 2008*, pages 3–14. ACM, 2008.

[14] D. Kuhlmann, R. Landfermann, H. V. Ramasamy, M. Schunter, G. Ramunno, and D. Vernizzi. An Open Trusted Computing Architecture – Secure Virtual Machines Enabling User-Defined Policy Enforcement. Technical Report RZ 3655 (#99675), IBM Research, 2006.

[15] S. Miura, T. Okamoto, T. Boku, M. Sato, and D. Takahashi. Low-cost high-bandwidth tree network for PC clusters based on tagged-VLAN technology. *Parallel Architectures,Algorithms and Networks, 2005. ISPAN 2005. Proceedings. 8th International Symposium on*, 7-9 Dec. 2005.

[16] R. Perlman, D. Eastlake 3rd, D. Dutt, S. Gai, and A. Ghanwani. Rbridges: Base Protocol Specification, July 2008. draft-ietf-trill-rbridge-protocol-08.txt.

[17] VMware Inc. VMware Infrastructure 3 architecture. `http://www.vmware.com/pdf/vi_architecture_wp.pdf`, June 2006.

[18] VMware Inc. VMware Virtual Networking Concepts. `http://www.vmware.com/files/pdf/virtual_networking_concepts.pdf`, July 2007.

[19] H. Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communication*, 28:425–432, April 1980.