Network Troubleshooting with Mirror VNets

Andreas Wundsam, Amir Mehmood, Anja Feldmann
TU Berlin / Deutsche Telekom Laboratories, Germany
{andi, amir, anja}@net.t-labs.tu-berlin.de

Olaf Maennel
Loughborough University, UK
o.m.maennel@lboro.ac.uk

***Abstract*—Today diagnosing problems, deploying new services, testing protocol interactions, or validating network configurations are still largely unsolved problems for both enterprise and Internet Service Provider (ISP) networks. Due to the intrinsically distributed nature of network state, frequent timing dependencies, and sources of non-determinism involved, any change may introduce undesired effects—even the impact of a simple configuration change can be hard to predict.**

In this paper we show how to leverage network virtualization to improve our debugging ability: By replicating and cloning production networks and then applying the changes to the cloned network in a safe fashion. *Mirror VNets* thus enable troubleshooting as well as safe upgrades to both software and configuration.

I. Introduction

Despite all efforts, problem diagnosis, troubleshooting and safe evolution of networks remains a challenging problem due to their distributed nature. As such, diagnosis is often attempted in "artificial" environments (analytical models, simulators, testbeds) that offer good monitoring capabilities, or in a trial-and-error fashion in the actual production environment. However, these approaches have severe limitations: Models and simulations have limited prediction capabilities due to abstract modeling assumptions, testbeds are limited in scale due to their costliness, and trying to fix things on the production network often leads to other errors [16], [24]. Therefore, all these approaches often do not suffice to diagnose and then resolve real-life network problems since these often stem from complex interactions of many parties. Problems that only occur sporadically or are triggered by user interactions are known to be especially problematic, even if it is possible to replicate a complete setup in a lab environment.

We propose to utilize *Virtual Networks* (VNets) to help tackle these network diagnosis and troubleshooting problems. VNets expand the existing concepts of virtualized hosts and links to the entire network. Therefore, a VNet may span multiple physical network domains. Note that virtualized and shared resources are already common in today's networks. Examples include Ethernet VLAN's and MPLS, VPN's and VRF tables, IP service over DSL lines, shared undersea cables, shared mobile phone stations, etc. VNet management frameworks unify the management of these resources and enable VNets to be dynamically provisioned and configured and to operate in parallel on a shared physical infrastructure. A controlling instance *isolates* the individual VNets from each other. As a result, it is possible to experiment with one VNet, while maintaining stability in the rest of the system.

In this paper we show how properly implemented VNets enable us with good diagnosis and debugging capabilities. We propose *Mirror VNets*, which replicate networks and traffic in a safe fashion. Thus, the new Mirror VNet and the production VNet can operate in parallel and the user traffic is duplicated either completely or in part to both networks.

This allows the network owner to investigate a problem and locate its root cause in the Mirror VNet without interfering with the production traffic. Then, a fix can be developed, tested, and deployed even with real user traffic again without affecting the production setup. Once the network operator is convinced that the new setup is stable and fixes the problem he can switch the production VNet with the Mirror VNet in a near-atomic fashion. Therefore, Mirror VNets offer network operators a new range of capabilities from (a) safely evaluating and testing new configurations, via (b) testing new software components at same scale of the production network and under real user traffic, to (c) troubleshooting live networks.

This approach builds on the agility and isolation properties of the underlying virtualized infrastructure and does not require changes to the physical or logical structure of the production network. Instead, a network owner can dynamically provision a Mirror VNet as required. It does not require changes to the production network within the VNet— its protocols, its applications, or its configuration.

This paper makes the following contributions: (i) we introduce *Mirror VNets*, (ii) discuss their intrinsic benefits and limitations (iii) present an implementation based on XEN and OpenFlow and (iv) demonstrate its usefulness in a case study. We conclude that *Mirror VNets* have the potential to substantially ease network operation and improve resilience against mistakes.

Our work bears some similarities to the *Shadow Config* approach by Alimi et al. [6], designed to improve the safety and smoothness of configuration updates, but by virtue of the underlying virtualization extends the scope beyond configuration changes. Another sibling to our approach has been proposed by Lin et al. [23]. The authors introduce a framework that enables full reproducibility by recording and replaying all non-deterministic events. Note that the necessary coordination and transaction modules inside the Hypervisor add significant complexity to the *production datapath* and thus cannot be fully transparent to the production network. Numerous other approaches for improving troubleshooting support in networks have been proposed. Some authors [8], [18] propose adding pervasive tracing support throughout the network, which is

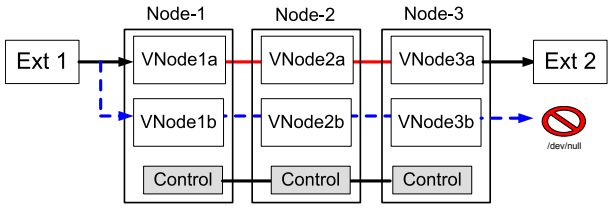

Fig. 1. Substrate running production VNet A, consisting of VNodes $1a, 2a, 3a$, and Mirror VNet B, consisting of VNodes $1b, 2b, 3b$

difficult because it requires changes to all involved hosts, routers, and software stacks. Automatic inference [5], [10], [27] can help debug many problems. The scale of testbeds can be increased with the help of time-dilating frameworks [19], [20], but sporadic problems remain challenging to reproduce. A complementary approach [21] is to use virtualization to improve bug-tolerance in software routers.

II. Mirror VNets

In this section, we show how network virtualization enables Mirror VNets. Then we discuss Mirror VNet use-cases and discuss their intrinsic benefits and limitations.

A. Assumptions

Our approach is based on the assumption that the network is virtualized. This includes node as well as link virtualization with proper isolation and resource management and accountability. As mentioned, many virtualized resources are present even in today's network infrastructures, e.g., VLANs, MPLS, VPNs, and comprehensive management frameworks are currently emerging. Virtualization is also considered a key enabler the Future Internet [17]. In addition, we assume that the infrastructure is shared among a reasonable number of VNets and thus none of the VNets uses more than a fraction, say 10%, of the overall substrate resources.

Another assumption is that we can duplicate input traffic to multiple VNets. Such capability is also commonly deployed in today's network infrastructure, e.g., spanning ports on Ethernet switches, multicast in routers, optical splitters, lawful intercept, and the functionality offered by OpenFlow.

B. Approach

To upgrade or troubleshoot, e.g., a production VNet (VNet A) in Fig. 1, the operator clones it and pairs it with a parallel Mirror VNet (VNet B). Thus VNet B starts with the identical configuration and state as VNet A, e.g., by relying on the same techniques as used for network node migration. The operator ensures that the input traffic for VNet A is *mirrored* either completely or in part to VNet B. This has to happen at all attachment points between the VNet and external entities, e.g., end-systems or network entry points. Therefore, any user traffic traverses both VNets. However, only traffic from the production VNet A is send back to external nodes. Traffic from the Mirror VNet B is discarded, silently.

Both VNets A and B now operate in parallel. Moreover, they are fully isolated from each other, on a node- and link-level. Thus, from now on they operate independently but under the same (or partial) user traffic. This means that it is now possible to use VNet B to troubleshoot network problems, test a new software or hardware release or reconfigure the network.

Once the problem has been diagnosed and a solution has been found, deployed to the Mirror VNet B and validated, the Mirror VNet B can be upgraded to become the new production VNet. To this end, the VNet attachment points of both VNets to external hosts are swapped: External hosts now communicate with VNet B, and VNet A acts as a Mirror VNet, and its output is discarded. This is a relatively simple operation and can be completed quickly. Finally, the old VNet A can be dismantled and its substrate resources released.

Note that not all traffic has to be mirrored and not all mirrored traffic needs to be transmitted over the wire. Depending on the scenario, the operator can use *mirror strategies* to adapt the volume of the traffic to be mirrored and transmitted. Possible Mirror strategies include: *(a) Full Mirror:* Every packet is mirrored verbatim and transmitted independently in both VNets. *(b) Packet Headers:* Only packet headers are transmitted in the Mirror VNet, the payload is reconstructed with dummy data at the nodes. *(c) Traffic Stats:* Instead of actual packets, only aggregated traffic statistics are transmitted and similar traffic is reconstructed at the nodes under investigation. *(d) Sampling:* Only a selected fraction of packets or flows is mirrored to the Mirror VNet.

These strategies can dynamically and selectively be applied to parts of the traffic. Note, for instance, that Control plane traffic has been shown to account for $< 1\%$ of traffic volume, but cause $> 95\%$ of bugs [7]. Accordingly, many problems (e.g., routing problems) can be investigated by only mirroring control-plane messages verbatim. For the data-plane traffic, headers or even reconstructed summaries may well suffice. Stress testing of a new software release can be started at only a fraction, e.g., 5%, of the overall traffic.

C. Use-cases

Mirror VNet can be useful in many scenarios, including routing optimizations and updates of network services.

Routing Configuration Optimization: When a network operator decides that he wants to re-optimize routing, e.g., by changing the link weights of his interior routing protocol or by introducing routing policies, he often does not want to experiment on the production network. Among the drawbacks are unintentional path choices, link overloads, packet reordering and losses during convergence, erroneous configurations, etc. However, by using a Mirror VNet he can perform all changes while traffic in the production VNet continues to flow unaffected. Indeed, the operator can monitor the Mirror VNET for the expected benefits or unexpected side effects. Only when the operator is "happy" with the new configuration state, he switches roles, and exposes the new routing to his users. The capacity overhead is very small as it is sufficient to mirror routing protocol messages exchanged between external entities and the Mirror VNet.

Update of a faulty network service: Consider a wide area network service based on an overlay network, e.g., a popular

Internet Telephony system. Suppose a number of overlay supernodes crash in irregular intervals. In this case, the operator can instantiate a Mirror VNet, in the exact same state as his production VNet. The bugs responsible for the problem are tracked down by manipulating selective traffic passed to the Mirror VNet. When a possible fix has been developed, the new version is deployed to the Mirror VNet, and an increasing amount of traffic is mirrored to the Mirror VNet.

This can cause some overhead especially with regards to bandwidth capacity. However, the benefit is that the new software components are stress-tested under real user traffic and are effectively probed for possible unknown regressions. Circumstances that might be very hard to discover in analytical models or test-labs. Upon satisfactory results the Mirror VNet can be swapped with the production VNet.

D. Discussion

The benefits of Mirror VNets are many-fold and include:

Resilience against operational mistakes: Mistakes during the configuration and/or update process are limited to the Mirror VNet. Thus they do not affect the production network. Moreover, the entire change set is tested under realistic conditions before affecting production.

Real user traffic: Mirror VNets expose the new system and its configuration to real user traffic at full scale and thus offer the opportunities to detect more bugs earlier.

Complex setups: Some problems can only be reproduced in complex setups, which can not be reproduced with models, simulations, or in test-labs. The only way to test under real-world conditions is to deploy in a real network – however, mirror VNets enable us to do this in a safe manner, without impacting the reliability of the production network.

Rollback/Undo for Networks: If after being set into production, an upgraded Mirror VNet exposes unexpected problematic side effects (e.g., due to a closed loop effect, see below), the operator can easily revert back to the old production network, and continue to investigate the problem without affecting the production users any longer.

An inherent limitation of this approach is that closed-loop effects caused by hosts outside the VNets cannot be predicted by monitoring the Mirror VNet. For example, an upgrade that provides faster delivery of requests to an external server may result in faster response traffic which may cause an overload on the network. Such effects only occur once the Mirror VNet is made productive and thus cannot be predicted by monitoring the Mirror VNet while it is still in mirror mode. Note that closed-loops *within* the VNets are not affected by this problem, so one possible solution is to integrate all communicating parties, including the end hosts, *into* the VNet itself, and have the VNet operate *end-to-end*.

Another concern is that the substrate has to carry the additional load imposed by the Mirror and the Control VNets. Note that depending on the problem under investigation, the mirroring strategies discussed in Section II-B can reduce the amount of traffic that actually has to be mirrored significantly, and stress-testing can be done gradually (e.g., at 10% of the traffic load). Finally, assuming many co-existent VNets on a well dimensioned (e.g., 2/3 loaded) substrate, debugging a limited number even at Full-Mirror mode is feasible. If the impact of a proposed change cannot be determined by other means, then the capacity-overhead may reflect a fair price. In the event of an unexpected traffic or load spike, the production VNets are always granted priority over Mirror VNets. Therefore, debugging or testing is affected but the user service in the production VNet is not deteriorated.

III. Implementation Options

This work assumes the presence of a virtualization solution that enables duplication of a virtual machine. In addition, we require the network substrate to be able to dynamically replicate traffic as required.

Node virtualization: Today, virtualization concepts are already ubiquitous in data centers [9] and router vendors, including Cisco and Juniper, offer support for virtualized routers. XEN [11], an open source virtual node monitor or *hypervisor*, is among the most commonly used open source virtualization solutions and allows to run multiple operating systems side-by-side. XEN has been shown to be a viable solution for building high performance routers on commodity hardware [14] if functional blocks are selected and placed carefully. Performance isolation and fairness issues can be challenging especially with regards to network I/O [12], [13]. Many other node virtualization solutions are available offering individual trade-offs between performance and flexibility, e.g., VMWare [4], KVM [22], OpenVZ [2]. XEN, and many others, feature *live migration* support, which can be easily extended to enable instant creation of mirror nodes, which essentially corresponds to a live-migration to `localhost`, without disbanding the original guest.

Packet replication: Our approach requires packets to be duplicated and delivered to two different virtual networks in parallel. It is known that naively bridging interfaces can impose severe performance penalties [12]. Fortunately, several hardware accelerated link virtualization technologies are about to become commercially available, including Multi-Queue NIC cards, and OpenFlow. Multi-Queue NIC cards (e.g., [3]) lower the I/O overhead by allowing packets to be delivered directly to the target VNode. OpenFlow [25] enables an external entity, the controller, to control Ethernet switches. This controller can dynamically set the forwarding rules using wildcard patterns across the packet headers while the frame forwarding is done by the switch hardware. Thus, OpenFlow is a flexible and powerful solution for the link substrate capabilities required by our approach.

IV. Case study

We present a case study highlighting the potential of Mirror VNets for troubleshooting a problem in a production network. We build a prototype system utilizing XEN 3.4 for host virtualization and OpenFlow 0.8.9 for link virtualization. A custom OF controller based on NOX serves the purpose of

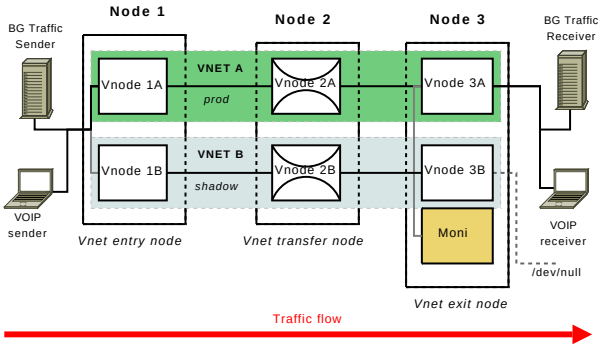

Fig. 2. Mirror VNet experiment setup

mirroring the incoming traffic to both VNets and filtering outgoing traffic according to the status.

Consider the following scenario: A VNet operator that offers both VoIP and Internet access across a best effort VNet, considers moving to a setup with service differentiation to offer better quality of service (QoS) to its VoIP traffic.

For our experiment, a VoIP call and background traffic of varying intensity is routed through the virtualized substrate (Fig. 2). The substrate network consists of three nodes. We now instantiate two parallel VNets, VNet A and B, each with a maximum bandwidth of 20 Mbs throughout the experiment, enforced by traffic shaping on node 2. Moreover, we setup an additional virtual network for monitoring. On entry to the VNet, traffic is duplicated to both VNets A and B and forwarded within each via node 2 to node 3 using separate virtual links (VLANs). On exit, when leaving node 3, only output from one VNet is sent to the receivers. In addition, the monitoring VNet "Moni" receives a copy of only the VoIP traffic from both VNets.

Metrics: For our evaluation, we measure at two points in the experiment: Moni records data for both VNets on exit of the VNet, while the receiver records the quality as experienced by the user. We record the percentage of dropped packages on the VoIP call as a rough quality indicator, and calculate the *Mean Opinion Score* (MoS) as defined by the ITU-T E-model [15][1].

Setup: For the VoIP traffic we use the $pjsip$ [26] client, an open source VoIP client based on SIP. It generates traffic at a constant rate of 80 kb/s using the $G.711$ codec with a net bitrate of 64 kb/s. Each VoIP RTP packet contains 20ms voice and has a payload of 160 Bytes. A pool of servers is used to generate the background traffic, using Harpoon [28], with properties that are consistent with those observed in the Internet – heavy-tailed file distributions and self-similar traffic, emulating the Internet access traffic by the users of the VNet. To account for different intensities of the background traffic during different times of the day we use two different load levels: L/H that correspond to 20-25%, 60-86% average link utilization. All traffic sources are located on the left in Fig. 2.

[1]The E-model states that the various impairments contributing to the overall perception of voice quality (e.g., drops, delay, jitter) are additive when converted to the appropriate psycho-acoustic scale (*R factor*). The *R-factor* is then translated via a non-linear mapping to the *Mean opinion Score*(MoS), a quality metric for voice. MoS values range from 1.0 (*not recommended*) to 5.0 (*very satisfied*).

TABLE I
EXPERIMENT OUTLINE ACROSS PHASES

Phase	1	2	3	4	5	6
Active VNets	A	A	A&B	A&B	A&B	B
Production VNet	A	A	A	A	B	B
QoS enabled	-	-	-	B	B	B
BG traffic load	L	L/H	H	H	H	H

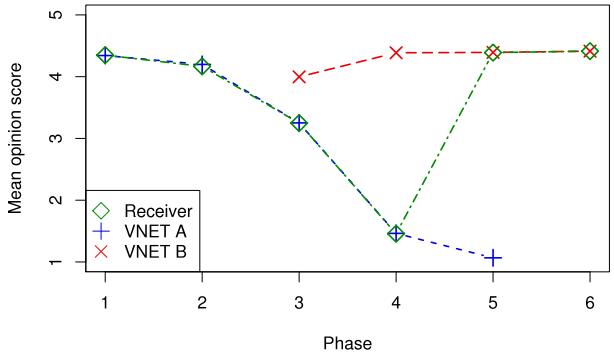

Fig. 4. Averaged MoS values per experiment phase, as measured within VNets A, B and at the end host (receiver).

The experiment is conducted in six phases with a length of five minutes each. Fig. 3 shows the rate of the background traffic (shaded area) averaged over 10s (scale on right axis) across time. In addition, Fig. 3 shows the number of dropped packets across time, again using 10s bins (scale on left axis). Drop rates for VNet A are depicted as blue plus signs, VNet B as red crosses, and the values measured at the receiver as green diamonds. Table I summarizes the configuration of each phase.

Results: In phase 1, background traffic is running at low intensity. In the middle of phase 2 the intensity of the Internet traffic is switched to high. This causes a problem in VoIP quality as measured by the MoS value, see Figure 4. The perceived quality drops from a MoS score of 4.34 which corresponds to a "very satisfied" service level drops to 4.16 which corresponds to a level of "satisfied".

As such, the VNet operator asks to instantiate a Mirror VNet at the beginning of phase 3. This means that all packets are now duplicated at node 1 and are routed in both VNets A and B. However, the end user for VoIP service is still getting service through VNet A. This allows the operator to assess the impact of the degradation and to do root cause analysis in VNet B. Indeed, the quality of the call decreases further in our experiment. In our case the operator decides to prioritize VoIP traffic to counter the bad performance. He enables QoS at the start of phase 4. Even though the background traffic increases further in this phase, the QoS reduces the loss rates within VNet B significantly and the MoS value increases again to 4.38. At the same time, VNet A is hit hard by the increased traffic, causing heavy congestion. In consequence, the VoIP MoS score drops to 1.45 ("not recommended").

At the start of phase 5 the operator switches his production VNet from VNet A to VNet B. Note that packet drops as experienced by the end user do not increase noticeably during the switch. After the switch is completed, the user can profit

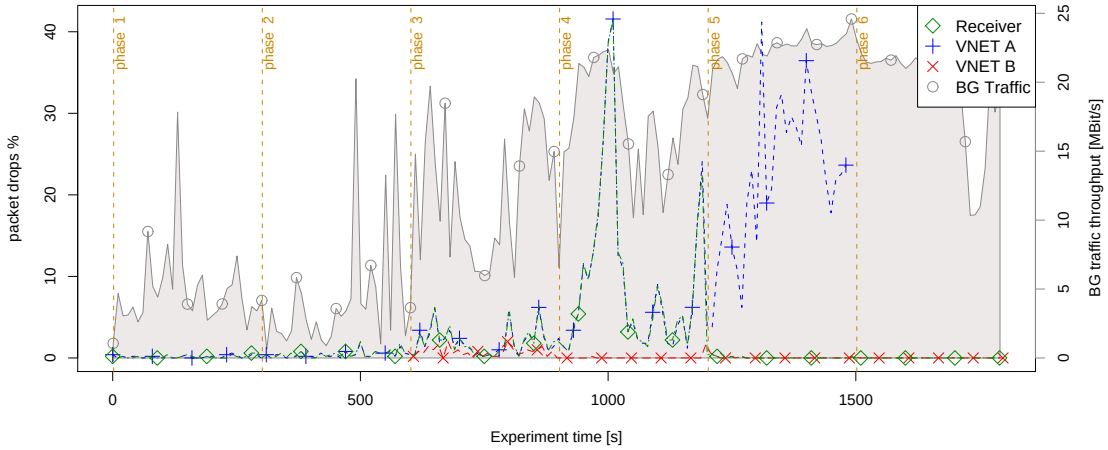

Fig. 3. VoIP packet drops (left) and background traffic throughput (right) across time

from the good performance provided by VNet B. With phase 6 the operator deactivates VNet A.

This very simple scenario shows how an operator can benefit from Mirror VNets, e.g., to smoothly upgrade his network configuration to amend a network performance problem. Early results indicate that the approach also works for the other use cases mentioned in Section II-C and that it can scale to larger topology sizes. Ongoing evaluation within larger OpenFlow-enabled infrastructures that are currently being deployed will provide additional insights into the scalability.

V. Summary and Future Work

In this paper, we present an approach that adds network troubleshooting capabilities to virtualized enterprise or ISP networks. *Mirror VNets* enable operators to upgrade configurations and software in an operationally safe manner with transaction semantics while exposing the new system and configuration to real user behavior. Less expensive, specific problems can be investigated and debugged, and interactions can be studied by sending only selected traffic to the Mirror VNet, e.g., control plane messages. The experiences with our prototype implementation underline the feasibility of the approaches, especially if used on a virtualization platform that offers good isolation. In the future, we plan to further evaluate the scalability of the system in the large scale OpenFlow-enabled testbeds currently planned, e.g., the Ofelia testbed [1]. The debugging capabilities of Mirror VNets are of particular interest in such environments where real users are using experimental software.

References

[1] EU Project Ofelia. http://http://www.fp7-ofelia.eu/.

[2] Openvz. http://wiki.openvz.org/.

[3] Solving the hypervisor network I/O bottleneck. http://www.solarflare.com/technology/documents/SF-101233-TM-5.pdf.

[4] Vmware infrastructure. http://www.vmware.com/products/vi/.

[5] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *ACM SOSP*, 2003.

[6] R. Alimi, Y. Wang, and Y. R. Yang. Shadow configuration as a network management primitive. In *Proc. ACM SIGCOMM*, 2008.

[7] G. Altekar and I. Stoica. Focus replay debugging effort on the control plane. Technical Report UCB/EECS-2010-88, UC Berkeley, 2010.

[8] A. Anand and A. Akella. Netreplay: a new network primitive. In *ACM HotMetrics Workshopt*, 2009.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Ra bkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, 2009.

[10] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proc. ACM SIGCOMM*, 2007.

[11] 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. In *ACM SOSP*, 2003.

[12] S. Bhatia, M. Motiwala, W. Mühlbauer, Y. Mundad, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In *ACM ROADS Workshop*, 2008.

[13] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Fairness issues in software virtual routers. In *ACM PRESTO Workshop*, 2008.

[14] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley. Evaluating xen for router virtualization. In *IEEE PMECT*, 2007.

[15] The E-model, a Computational Model for Use in Transmission Planning, ITU-T Rec. G.107, 2005.

[16] N. Feamster and H. Balakrishnan. Detecting bgp configuration faults with static analysis. In *USENIX NSDI*, 2005.

[17] A. Feldmann. Internet clean-slate design: what and why? *ACM Sigcomm CCR*, 37(3), 2007.

[18] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *USENIX NSDI*, 2007.

[19] D. Gupta, K. Vishwanath, and A. Vahdat. Diecast: Testing distributed systems with an accurate scale model. In *USENIX NSDI*, 2008.

[20] D. Gupta, K. Yocum, M. Mcnett, A. C. Snoeren, A. Vahdat, and G. M. Voelker. To infinity and beyond: Time warped network emulation. In *ACM SOSP*, 2005.

[21] E. Keller, M. Yu, M. Caesar, and J. Rexford. Virtually eliminating router bugs. In *Proc. ACM CONEXT*, 2009.

[22] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proc. of the Linux Symposium*, 2007.

[23] C.-C. Lin, M. Caesar, and J. van der Merwe. Towards interactive debugging for isp networks. In *ACM HotNets Workshop*, 2009.

[24] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration. In *Proc. ACM SIGCOMM*, 2002.

[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM Sigcomm CCR*, 38(2), 2008.

[26] Open Source SIP Stack and Media Stack for Presence, Instant Messaging, and Multimedia Communication, 2009. http://www.pjsip.org.

[27] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. Wap5: black-box performance debugging for wide-area systems. In *ACM WWW*, 2006.

[28] J. Sommers and P. Barford. Self-configuring network traffic generation. In *ACM IMC*, 2004.