Improving Network Troubleshooting using Virtualization

Andreas Wundsam Amir Mehmood Anja Feldmann Olaf Maennel
TU Berlin / Deutsche Telekom Laboratories, Germany

ABSTRACT

Diagnosing problems, deploying new services, testing protocol interactions, or validating network configurations are hard problems in today's Internet. This paper proposes to leverage the concept of Network Virtualization to overcome such problems: (1) *Monitoring VNets* can be created on demand along side any production network to enable network-wide monitoring in a robust and cost-efficient manner; (2) *Shadow VNets* enable troubleshooting as well as safe upgrades to both the software components and their configurations. Both approaches build on the agility and isolation properties of the underlying virtualized infrastructure. Neither requires changes to the physical or logical structure of the production network. Thus, they have the potential to substantially ease network operation and improve resilience against mistakes.

1. INTRODUCTION

There is hardly any more difficult task than diagnosing problems in the Internet. This is in part due to the complexity of the problem itself. IP networks are distributed in nature, they span the globe and involve billions of components, while even individual network protocols, e.g., BGP, are very complex in their own right. There are interactions of various network layers, e.g., IP routing with SONET, and some applications rely on unstated assumptions, e.g., on RTTs not exceeding certain values. Moreover, problems are often only vaguely specified. For instance, a user may be complaining about the Web not working, when in fact DNS is failing.

While these complexities may be intrinsic to the nature of the Internet, coping with them is made all the harder by our dismal abilities to monitor large scale production networks. Monitoring capabilities are usually inversely proportional to the size of network – if the network is small we can monitor and analyze almost everything out-of-band. However, if the network is large, we can monitor and analyze data at only a small number of locations and often have to resort to in-band monitoring, adding substantial overhead.

As a result, error diagnosis is often delegated to "artificial" environments (simulator, testbed) that offer good monitoring capabilities, or attempted in production environment with only limited monitoring capabilities. Such approaches have significant drawbacks: Simulations have limited accuracy, testbeds are limited in scale, in-band monitoring is limited. Thus, these methods often do not suffice to catch the real-life network troubleshooting problems that stem from complex interactions of many parties, especially if they only occur sporadically. Even if it is possible to replicate a complete setup as done by some vendors, e.g., via Cisco's NSite [1], these setups only carry test traffic. As such, many problems are not observable.

To solve such problems, network engineers usually rely on their intuition to guide them when adjusting the network configuration or adding extra instrumentation, either external or in-band. However, this can and does introduce new bugs [14, 17] and the instrumentation overhead may even alter the very behavior that is meant to be observed. The latter is often referred to as *probe effect*.

In this paper, we propose to utilize the emerging concept of network virtualization to tackle the network diagnosis problem in a novel fashion. Network virtualization expands the existing concepts of host and link virtualization to the entire network. Indeed, a virtual network may span multiple physical network domains. Network virtualization frameworks enable *virtual networks* or *VNets* to be *dynamically provisioned* and *configured* and to operate in parallel on a *shared physical infrastructure*.

A controlling instance, often called the *virtual node monitor isolates* virtual networks from each other. As a result, virtual networks can be quickly instantiated on demand, and each virtual network is independent of the other virtual networks.

The contributions of this paper are two new novel approaches for adding to our network diagnosis capabilities while avoiding the undesirable effects outlined above: (a) *Monitoring VNets* enable decentralized, non-intrusive network-wide monitoring in production networks and (b) *Shadow VNets* for replicating not just networks but also their input, safely. This offers network operators a new range of capabilities at with low overhead: He can safely evaluate and test new configurations or test new software components, at the scale of his production network, under real user traffic. As the new setup exist in parallel with the old one he can swap them in a near-atomic fashion. This enables him to switch only once he has convinced himself that the new setup is operational and offers the expected benefits. We note that such capabilities are crucial for performance troubleshooting, for network diagnosis, for debugging new network protocols, or to evaluate new network configurations. The overhead is very limited given the assumption that there will be a reasonable number of VNets but only a small number of Monitoring or Shadow VNets.

Our naming of Shadow VNets is inspired by the results of Alimi et al. [6], who implement *shadow configurations* to improve the safety and smoothness of configuration updates. While shadow configurations provide a convincing solution to today's configuration management, they are intrinsically unable to handle, e.g., software updates or updates to the end-systems. Moreover, network virtualization intrinsicly offers features that have to be added to support shadow configurations. As such, Shadow VNets offer, on the one hand, a clean implementation of shadow configuration by relying on the abstraction and isolation properties of network virtualization. Additionally, they substantially extend their scope.

The remainder of this paper is organized as follows: In Section 2 we present our approaches and review their intrinsic benefits and limitations. We then discuss, in Section 3, enabling technologies and currently available implementation options for our approaches. With the help of a prototype implementation, we evaluate the feasibility of our approach in Section 4. Finally, we conclude with an outlook in Section 5.

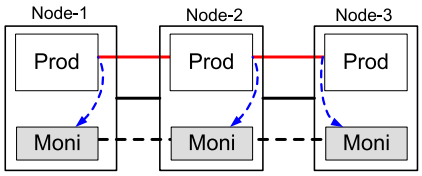


Figure 1: Substrate running production VNet (Prod) and Monitoring/Analysis VNet (Moni)

2. APPROACH

In the following we discuss the benefits and limitations of our ideas on how virtualization can help network troubleshooting.

2.1 Monitoring VNets

Network operation heavily relies on network traffic monitoring at different granularities. For instance, link statistics and CPU utilization on routers are commonly used to assess the load on network links and routers. If, e.g., a router's CPU usage is getting too close to 100% most ISPs require network operations to react to prevent the router from crashing. Other beneficiaries of network measurements include traffic engineering and legal intercepts. Among the most powerful tools for network diagnosis is fine grained packet monitoring. This is done, e.g., with `tcpdump` or `wireshark` on end-systems, via *monitoring ports* on switches, or via *passive optical taps* on the wire. Unfortunately, deployment of such monitoring often requires changes to the logical configuration or even additional hardware and thus changes to the physical configuration of the network. Accordingly, such solutions are often costly and difficult to deploy on demand.

Therefore, we propose to exploit the agility and isolation offered by network virtualization to enable a lightweight alternative that can be deployed without changes to the production network itself. With the help of a virtualization framework we couple any production network with a monitoring virtual network, as shown in Fig. 2.1 for VNet *Prod*. The monitoring network has its own resources and consists of VNodes (called *Moni*) on each of the substrate nodes of VNet *Prod* which are interconnected by separate virtual links. On each node, traffic going in and/or out of the production VNodes can be selectively mirrored to interfaces on the monitoring VNode depending on the requirements of the mon-

itoring use case. The selected traffic can then be processed by the software running within the monitoring VNode. For example, *Moni* can record all selected packets, or compute aggregate statistics to reduce the data volume, or even analyze them in real-time using a specialized application. Finally, the results can be made available to the network operator. All the computation as well as the data transfer is accounted to Moni rather than the production VNet. As such, the production network can remain unaware of the monitoring VNet.

The benefits of network-wide traffic monitoring are multifold and include:

Cost effectiveness: This approach does not require any physical or logical changes to the configuration of the production network – it only adds a virtual network. Thus it has a light deployment "footprint". As a result it might be possible to deploy it more widely than traditional monitoring approaches even with limited resources, thus enabling network-wide monitoring of a geographically distributed network.

Resilience against operator mistakes: The absence of changes to the production network also results in increased operational safety – configuration mistakes in the Monitoring VNet do not result in production network outages. Note, our approach assumes a virtualization framework that enables automatic instantiation of virtual networks without the need for manual configuration.

Performance isolation: Virtualization offers resource isolation. Thus it limits the performance degradation caused by the monitoring VNet on the production VNet. Even if a monitoring VNode is overloaded, e.g., due to an unexpected traffic spike or a bug in the deployed IDS, the production networks is isolated and can continue to operate at regular speed (given appropriate priorities).

Reduced result transfer volume: We offer local analysis on each node. Thus it is possible to substantially reduce the data transfer volume to the central data warehouse of the operator.

In essence, our approach offers the benefits of a monitor port on every switch/router in the network coupled with a close-by analysis node. It can be provisioned dynamically on demand and without the need for additional or dedicated hardware. The main limitations of the proposed approach are that

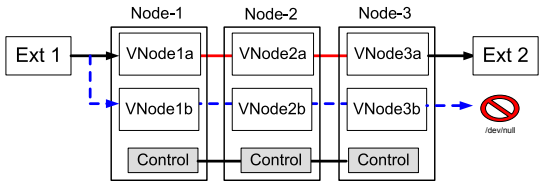


Figure 2: Substrate running production VNet(a), Shadow VNet(b) and Control/Monitoring/Analysis VNet(Control)

current virtualization frameworks do not yet offer the monitoring service and the question of how high the overhead is. For a study of the latter see Section 4.1.

2.2 Shadow VNets

Next, we show how network virtualization can be used to enable smooth and fail-safe network upgrades, including network software, operating system images and configurations. Inspired by Shadow Configs [6] we call the resulting VNets: *Shadow VNets*.

To upgrade, e.g., the production VNet (VNet A) in Fig. 2, we clone it by creating a parallel Shadow VNet (VNet B). It uses the identical configuration and is in the same state, e.g., by relying on same techniques as used for network node migration. In addition, we create a third virtual network, Control VNet, that serves as a monitoring and control facility. Both VNets A and B continue to operate in parallel and isolation from each other. However, VNets rarely operate in complete isolation from the outside world. Rather they interact with external entities, e.g., end-systems or non virtualized legacy parts of the Internet. We thus duplicate traffic from external entities on entering the production VNet and mirror it to the Shadow VNet. Therefore, any user traffic traverses both VNets. However, only traffic from the production VNet A is propagated to external nodes. Traffic from the Shadow VNet B is discarded, silently. To upgrade the production network from VNet A to B the only necessary change is to discard the output of VNet A and use the output from VNet B, making VNet A the new Shadow VNet of VNet B. As such, it is always possible to quickly roll back to the "old" network. Finally, at some later point in time, the old VNet, VNet A, can be dismantled to free resources.

Therefore, any operator can use Shadow VNets to safely reconfigure and then upgrade their network. For example, he can optimize the routing protocols

by changing link weights or introducing QoS parameters, or he can upgrade faulty software components to newer versions. To judge if his changes improve the network operations he can use the nodes of the control network to verify the impact of his changes. Only if a new configuration is stable, e.g., if the routing protocols in VNet B have converged, may he decide to use VNet B as his new production network. In contrast to Shadow Config, Shadow VNet does properly separate real traffic from shadow traffic (shadow config abuses the IP protocol version field), and Shadow VNets support software and configurations, even operating system running in parallel.

The benefits of Shadow VNets are also multifold and include:

Resilience against operator mistakes: Mistakes during the configuration and/or update process are limited to the Shadow VNet. Thus they do not effect the production network. Moreover, the entire change set can be tested under realistic conditions before moving them into production.

Real user traffic: Shadow 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.

Near Atomicity: Changing a network setup usually takes time and it may require some time for stabilization. This is hidden from the users of the production VNet.

An inherent limitation of this approach is that closed-loop effects cannot be captured by monitoring the Shadow VNet. For example, a configuration 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 Shadow VNet is made productive and thus cannot be predicted by monitoring the shadow VNet while it is still in shadow mode. One possible solution is to integrate all relevant communicating parties *into* the VNet itself. Another limitation is that the substrate has to carry the additional load imposed by the Shadow and the Control VNets. However, if we assume that there is a reasonable number of active VNets supported by the substrate at any given time it should be possible to add Shadow VNets for a fraction of the VNets. In the event of an unexpected traffic or load spike, the productive VNets can be given priority over the Shadow VNets, thus impairing the measurements and testing but not deteriorating the quality experienced by the user.

3. IMPLEMENTATION OPTIONS

This work assumes that network virtualization architectures exist that can automatically provision VNets and offer proper VNet isolation. In addition, we need the ability to mirror packets.

VNet Architectures: Several past and present projects are proposing network virtualization architectures that offer some of the necessary features, e.g., X-Bone [21], Genesis [10], DaVinci [15], VINI [8], Cabernet [22] and 4WARD [5]. While the earlier proposals, X-Bone [21], Genesis [10], DaVinci [15], offer automatic configuration, they were never designed to offer Internet wide VNets with performance isolation which potentially involves many ISPs.

VINI [8] is available today as it builds upon planetlab and special servers. It offers simple container based host virtualization coupled with network virtualization capabilities for building test setups with virtual topologies. Other proposals aim at providing perspectives for virtualizing the Internet at large, and incorporate economical aspects and detailed roles. Examples include Cabernet [22] and 4WARD [5]. We loosely base our terminology on the latter.

Manual Resource Virtualization: Given that no virtualization framework is already widely deployed we test the feasibility of our proposed network diagnosis approaches by manually configuring the monitoring networks using available node and link virtualization techniques. 802.11Q VLANs are commonly used in local are networks to build virtual links while MPLS tunnels are prevalent in the WAN.

Recently, node virtualization has become very popular for data centers and server farms as it offers better utilization of the computing capabilities, increased availability, and ease of management. XEN[7], 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. Hereby, one OS assumes a privileged role. It serves as the interface to the Hypervisor and controls the hardware (*Dom0*). The others OSs are unprivileged guests (*DomU*). XEN has been shown to be a viable solution [12] for building high performance routers on commodity hardware. Still it is not perfect – functional blocks

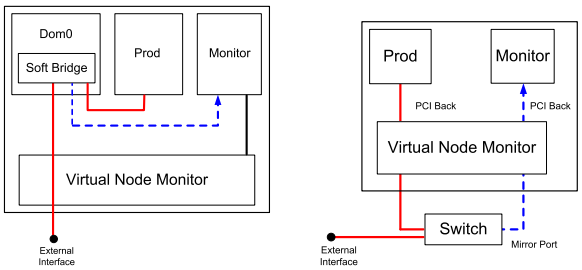
have to be selected and placed very carefully and performance isolation and fairness remain problematic especially with regards to network I/O [9, 11]. Many other node virtualization solutions are available offering individual trade-offs between performance and flexibility, including VMWare [4], KVM [16], OpenVZ [2], and Trellis building on VServer and NetNS [9].

Packet mirroring: For both of our approaches, we need packets to be duplicated and delivered to two different virtual networks in parallel. We thus study how links are *attached* to the virtual nodes, either in software or with hardware support. Software options include using the standard Linux bridge device or the `tc mirred` command. However, naively bridging interfaces can impose severe performance problems [9].

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 [18] 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 offers to be a flexible building block for virtualized solutions.

Prototype implementation: For our feasibility study, we build a prototype system utilizing XEN for host virtualization and VLANs for link virtualization. Unfortunately, current virtualization techniques do not yet offer perfect isolation especially regarding network I/O [11, 9]. To study its impact we compare multiple options of link attachment and packet duplication. Option (a) uses the standard Linux software bridge, see Fig. 3(a). Option (b) maps the NIC directly into the DomUs and thus hides it from the Dom0(`pciback` see Fig.3(b)).

In Option (a) we duplicate the packets inside the host, using `tc`. In Option (b) this is not possible. Packets have to be inspected and duplicated externally. We can use a statically configured switch port or a dynamically configured open flow switch for this purpose. In this work, we simulate the use of Multi-Queue NIC cards by using several dedicated cards, and the use of OpenFlow-enabled traffic duplication by manually configuring monitoring ports on a switch.



(a) internal bridging (b) external bridging

Figure 3: Options for link attachment

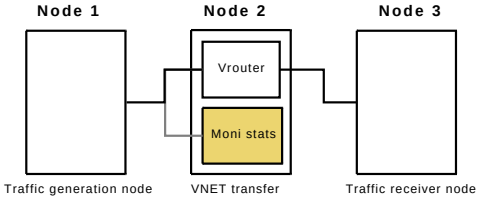


Figure 4: Monitoring experiment setup

4. FEASIBILITY STUDY

4.1 Monitoring VNets

To assess the feasibility of network-wide Monitoring VNets, we need to study if current virtualization approaches offer sufficient isolation. As such, we focus on the monitoring capabilities and not on the ability to setup the monitoring virtual network. The task we choose for the virtual network is packet forwarding – the base task of any virtual node.

For our evaluation we rely on a three node setup. Each node has two Quad Core Intel Xeon L5420 processors running at 2.5GHz, 16GB of RAM, and 4-8 1GBit/s Intel Ethernet ports. The schematic setup is shown in Fig. 4. We deploy Debian Linux 4.0 with XEN 3.0.3, which is part of the distribution. We use this out-of-the-box configuration as it resembles a possible production deployment better than custom, hand optimized kernel and hypervisor versions. The virtual network consists of 3 nodes. Node 1 is the traffic source while Node 3 is the sink. Node 2 forwards the traffic from Node 1 to Node 3.

On UNIX systems forwarding performance is usually dominated by the per-packet overhead. As such, our baseline experiment uses minimum sized packets and explores the performance impact of the various options on how to attach the link to the virtual node and how to do packet duplication, see Sec. 3. Fig. 5 shows the boxplots of the forwarding per-

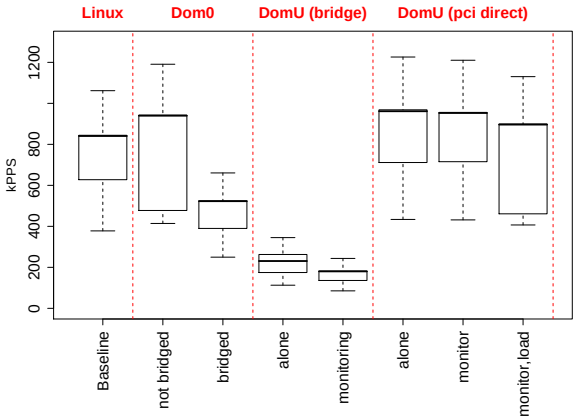


Figure 5: Forwarding performance for 64-Byte packets, with and without Monitoring

formance within 1 second time bins for an experiment duration of 500 seconds. Results are grouped according to the phases of the experiment by the dashed vertical (red) lines.

In the first two phases, we baseline our setup by measuring the forwarding rate in native Linux (group 1) and Dom0 (group 2). Native Linux (Debian 2.6.18-6) supports a median forwarding rate of 840 kpps. Interestingly, the XEN kernel performs better when doing native, unbridged forwarding in Dom0 (939.9 kpps). Next, we introduce the soft-bridge that is required for internal attachment of the DomUs, but still keep the forwarding in Dom0. We notice that the performance drops to 522 kpps. We then switch to option (a) by delegating the forwarding to DomU via a soft-bridge (third phase). Notice that the forwarding performance drops to 215 kpps even without monitoring, After enabling mirroring forwarding performance is further reduced to 180 kpps which indicates that network performance isolation is a problem in this setup.

Next we study option (b) (directly attaching the DomU interface to the NIC via `pcidirect`, group 4 in the figure). There is hardly any performance degradation. Indeed, we see a performance improvement: without monitoring, a directly attached DomU forwards at 961.6 kpps. We then enable packet duplication via option (b) (simulated OpenFlow packet duplication using a manually configured switch monitoring port). As expected, the performance impact is minimal – forwarding is at 953 kpps. Even when the monitoring domain is overloaded, with 100% CPU load and 100% hard drive I/O load, forwarding performance degrades by only 5% to 896 kpps.

We conclude that probe-effect free Monitoring VNets

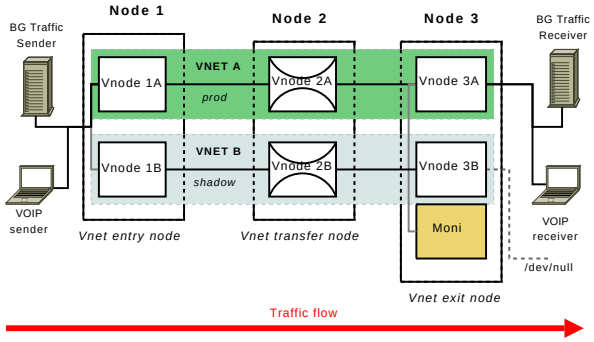


Figure 6: Shadow VNet experiment setup

are possible. However, their performance strongly depends on the isolation properties offered by the virtualization platform for network I/O.

4.2 Shadow VNets

To assess the feasibility of a network-wide Shadow VNet we choose 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. This move is motivated by customer complains about their VoIP quality during certain times of the day. Given the open-loop nature of VoIP traffic we expect that VoIP performance can be estimated with some accuracy by the Shadow VNet approach.

For our experiment, a VoIP call and background traffic of varying intensity is routed through a virtualized substrate (Fig. 6). The substrate network again consists of three nodes. We now instantiate two parallel VNets, VNet A and B, each with a maximum bandwidth of 20 Mbit/s 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 *MoS* value as defined by the the E-model,

Table 1: Experiment outline across phases

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

an ITU-T standard for measuring the transmission quality of voice calls [13][1].

Setup: For the VoIP traffic we use the *pjsip* [19] client, an open source VoIP client based on SIP. It generates traffic at a constant rate of $80kbps$ using the $G.711$ codex with a net bitrate of $64kbps$. Each *voip* RTP packet contains 20 ms voice and has a payload of 160 Bytes. A pool of servers is used to generate the background traffic, using Harpoon [20], 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 during the day we use two different load levels: L/H correspond to 20-25%, 60-86% average link utilization, respectively. All traffic sources are located on the left in Fig. 6.

The experiment is conducted in six phases; each of five minutes. Fig. 7 shows the rate of the background traffic (shaded area) averaged over 10s (scale on right axis) across time. In addition, Fig. 7 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 1 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 8. The perceived quality drops from a MoS score of 4.34 which corresponds

[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*). Values above 4.0 indicate *satisfied* users, values below 4.0/3.6/3.1 indicate that *some/many/nearly all users* are *dissatisfied*.

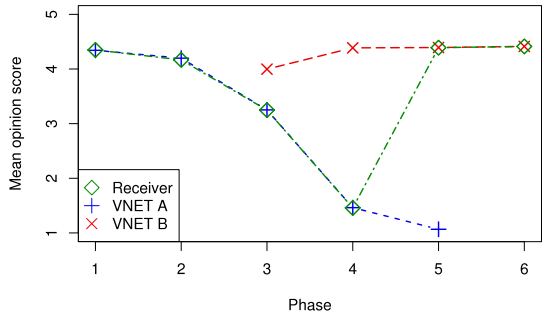


Figure 8: MoS results per phase

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 Shadow 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. this reduces the loss rates within VNet B significantly and the MoS value increases again to 4.38. Indeed, due to some bad congestion the VoIP MoS score within VNet A drops to 1.45 which corresponds to "not recommended".

At the start of phase 5 the operator switches his production VNet from VNet A to VNet B. Therefore, the user is now getting the good performance provided by VNet B. With phase 6 the operator deactivates VNet A.

Already this very simple scenario shows how an operator can benefit from Shadow VNets, e.g., to smoothly upgrade this network configuration to amend a network performance problem.

5. SUMMARY AND FUTURE WORK

In this paper, we present two novel approaches that leverage the capabilities of network virtualization to add to our network troubleshooting capabilities, especially for large production networks. *Monitoring VNets* can be used to provide cost-effective, side effect free network-wide monitoring capabilities. *Shadow VNets* enable operators to upgrade configurations and software in an operationally safe way and with transaction semantics while exposing

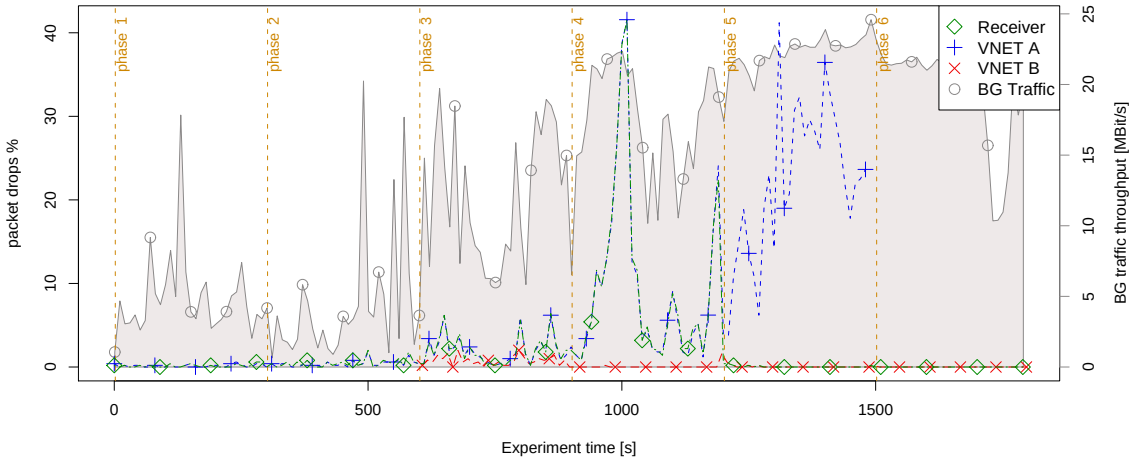


Figure 7: VoIP packet drops (left) and background traffic throughput(right) across time

the new system and configuration to real user behavior. As such the system can be tested before putting it into the wild.

The experiences with our prototype implementation underlines the feasibility of the approaches, especially if used on a virtualization platform that offers good isolation. It also hints at the power of these new troubleshooting tools.

In the future we plan further experiments using hardware virtualization enablers (e.g., Open Flow, Multi Queue NIC) as these promise proper isolation and explore the scalability limits. Moreover, we plan to integrate Monitoring VNets and Shadow VNets into one of the emerging VNet architecture platforms.

6. REFERENCES

[1] Networked solutions integrated test engineering: Delivering on the promise of innovation. `https://www.cisco.com/en/US/solutions/ns341/ns522/networking_solutions_products_genericcontent0900aecd80458f98.pdf`.

[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] 4WARD Project. `http://www.4ward-project.eu`.

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

[7] 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.

[8] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: Realistic and controlled network experimentation. In *ACM Sigcomm*, 2006.

[9] 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.

[10] P. Ch, A. Fisher, C. Kosak, T. E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable resource management for value-added network services. *IEEE Network*, 2001.

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

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

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

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

[15] J. He, R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford, and M. Chiang. Davinci: Dynamically adaptive virtual networks for a customized internet. In *Proc. ACM CONEXT*, 2008.

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

[17] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration. In *ACM Sigcomm*, 2002.

[18] 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.

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

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

[21] J. D. Touch. Dynamic Internet overlay deployment and management using the X-Bone. In *ICNP*, 2000.

[22] Y. Zhu, R. Zhang-Shen, S. Rangarajan, and J. Rexford. Cabernet: Connectivity architecture for better network services. In *ACM ReArch Workshop*, 2008.