XNetMon: A Network Monitor for Securing Virtual Networks

Natalia Castro Fernandes and Otto Carlos Muniz Bandeira Duarte
Universidade Federal do Rio de Janeiro - GTA/COPPE/UFRJ
Rio de Janeiro, Brazil

***Abstract*—Isolation and performance are critical issues for virtual networking. In this paper, we consider the use of Xen virtualization platform for building software-based virtual routers. We propose a network monitor for Xen to increase the isolation and the performance on packet forwarding. The network monitor controls the use of shared resources and punishes misbehaving virtual routers, guaranteeing an isolated operation of the virtual networks. In order to secure the shared data plane, we propose a secure communication protocol that provides mutual authentication, protection against replay attacks, and privacy between the virtual routers and the administrative domain. The results obtained with the developed prototype show that our proposal guarantees availability of the virtual-network control and packet forwarding services and also provides a fair resource sharing.**

I. Introduction

In the network virtualization paradigm, different virtual routers share a physical router in order to provide different network services simultaneously. Key aspects of this paradigm are isolation and performance on packet forwarding. Isolation ensures independent virtual network operation, preventing malicious or fault virtual routers interference in the operation of other virtual networks.

Xen [1] is a virtualization platform that can be used to create virtual routers, each with its own operating system and protocol stack, in commodity computers. The Xen platform, however, does not provide a complete isolation and also presents a low performance on handling network input/output (I/O) operations [2]. This paper proposes XNetMon, a network monitor for Xen that ensures isolation among virtual networks and provides a high packet forwarding performance. XNetMon is based on the paradigm of plane separation [3], in which data forwarding and control mechanisms are decoupled. Thus, control, such as routing, is accomplished inside the virtual machine, ensuring flexibility in the design of control mechanisms, while packet forwarding is performed in a privileged domain, called Domain 0 (Dom0), providing near-native performance. XNetMon guarantees a secure communication between virtual routers and Dom0, controls plane separation, and monitors the Dom0 resource consumption caused by each individual virtual router, ensuring that no router interferes with the others. Our controller affords a differentiated Dom0 resource allocation, allowing the administrator to assign different priorities to each virtual network.

We developed a prototype to evaluate the resource sharing and the security level provided by XNetMon in the presence of malicious virtual routers. The results show that XNetMon avoids that attack traffic disrupts the shared data plane. The proposed controller also reduces delays in transmissions between external machines through the virtual router during Dom0 overload in more than eight times. In addition, XNetMon effectively controls resource sharing, such as bandwidth, reducing errors in the resource division in more than three times when compared with other proposals.

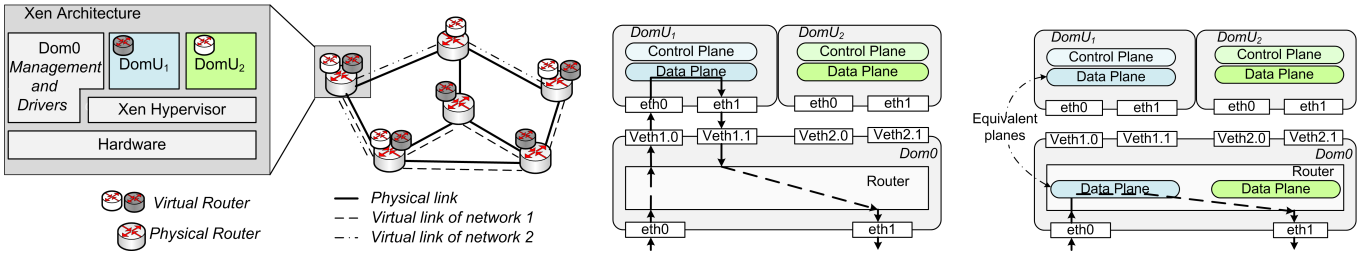
The rest of the paper is organized as follows. Section II and Section III describe the network and the attacker models, respectively. Section IV presents the proposal and Section V shows our prototype and the results. Finally, Section VI describes related work and Section VII concludes the paper.

II. Network virtualization using Xen

The virtual network model using Xen considers that virtual machines behave as routers. A virtual network is defined as a set of virtual routers and links, created over the physical infrastructure, as illustrated by Fig. 1(a). The Xen architecture is composed of the hypervisor, the virtual machines, called unprivileged domains (DomU), and a privileged virtual machine called Domain 0 (Dom0). The Xen hypervisor controls the physical resource accesses and handles the I/O operations performed by the domains. Dom0 is a privileged domain that directly accesses the hardware. Since Dom0 is a driver domain, it stores all physical device drivers and creates an interface between the virtual drivers placed in the unprivileged domains and the physical devices. In addition, Dom0 is also the management interface between the administrator and the hypervisor to create virtual machines, modify Xen parameters, and manage Xen operation.

Sending and receiving packets are I/O operations, which require the use of the device drivers located at Dom0. Thus, all network operations of the DomUs generate an overhead in both memory and CPU of Dom0. The Xen hypervisor, however, does not efficiently isolate Dom0 resource usage, which is a major vulnerability of Xen. Table I shows that a DomU can easily increase Dom0 CPU consumption by performing network operations[1]. Since data transfer between two DomUs and data transfer between DomU and Dom0 are Dom0 CPU-demanding operations, a malicious or fault action in a DomU can easily exhaust the Dom0 resources and thus compromise the performance of all the other domains. One of the goals of

[1]Tests performed with the Top tool in a machine with Intel Core 2 Quad processor with 4GB of RAM and Xen 3.4-amd64. Each DomU is configured with one virtual CPU and 128 MB of memory, and Dom0 is configured with one virtual CPU and no memory constraints. Each virtual CPU is associated with an exclusive physical CPU. TCP traffic was generated with Iperf. The basic CPU consumption indicates the CPU usage in Dom0 when there are no operations in the DomUs. We assume a confidence interval of 95%.



(a) Xen architecture with two virtual networks: DomUs behave as virtual routers.

(b) Conventional packet forwarding.

(c) Packet forwarding with plane separation: data packets pass only through Dom0.

Fig. 1. Xen architecture and packet forwarding modes assuming two virtual networks.

the proposal is to prevent that any operation performed on a virtual network breaks the isolation between networks.

TABLE I
CPU CONSUMPTION ON DOM0.

CPU (%)	Description
$0,71 \pm 0,60$	Basic CPU consumption on $Dom0$
$66,43 \pm 8,93$	TCP traffic from $DomU$ to $Dom0$
$85,49 \pm 5,91$	TCP traffic from $DomU_1$ to $DomU_2$
$1,79 \pm 1,01$	TCP traffic from an external machine to $DomU$

The Xen conventional architecture is not efficient for network operations because DomU packet forwarding takes a long and slow path. As depicted in Fig. 1(b), the packet arrives at Dom0, follows to DomU, and returns back to Dom0 to be forwarded to the next router. The plane separation paradigm is an alternative to improve the forwarding performance because packets are directly forwarded by a shared data plane in Dom0, as shown in Fig. 1(c). The plane separation is accomplished by maintaining a copy of the current forwarding table of DomU in Dom0, which has direct access to the hardware. It is important to note that data packets are directly forwarded by Dom0, but control packets are forwarded to DomU to update the control plane. Also, plane separation does not impede flexibility in packet forwarding. If a virtual router needs to do specialized operations not supported by the shared data plane, such as monitoring or modifying a specific header field, it can ignore plane separation by inserting a default route to the virtual machine in the forwarding table in Dom0, as done in the conventional packet forwarding.

III. ATTACKER MODEL

DomUs can affect each other intentionally or not, due to malicious actions and system faults. We classify a malicious behavior when a DomU intentionally performs activities to degrade the performance of other virtual networks. For fault behavior, we denote every attempt of a DomU to exceed its resource reservation when all the Dom0 resources are in use, which non-intentionally can prevent the operation of other domains. Both behaviors are harmful and, for simplicity, we define all the domains that perform such behaviors as opponents. The other domains are called common domains.

IV. PROPOSAL DESCRIPTION

The main objective of XNetMon is to provide the isolation of virtual networks and a secure data plane update. A secure data plane update is achieved when the control service that updates the shared data plane is always available and the communication between DomU and Dom0 is done through a secure channel. XNetMon achieves isolation by controlling the use of Dom0 resources. XNetMon also allocates and monitors physical resources used by all DomUs according to the parameters set by the administrator of the physical machine. Then, the administrator specifies the minimum resources of Dom0 that each virtual network can use and assigns priorities in the use of the idle resources. The architecture of XNetMon, illustrated in Fig. 2, is client-server based, assuming that the server is placed at Dom0 and the client is placed at each virtual router (DomU). The XNetMon server is composed of a controller module that monitors bandwidth, CPU, and memory usage in Dom0 and punishes opponent domains. Additional features of XNetMon include: the virtual network definition, the data plane manager and the secure communication modules. The virtual network definition module describes the traffic characteristics of each virtual network, ensuring that different virtual networks do not overlap. The data plane manager module updates routing tables and packet filters in Dom0 according to the control plane on each DomU. Finally, the secure communication module ensures a reliable message exchange between virtual machines and Dom0 for synchronizing the data plane.

A. Controller Module

The controller allocates Dom0 resources and also monitors their total usage, $U(t)$, and their usage by each virtual router i, $U_i(t)$, in every T seconds. The allocation of Dom0 resources takes place in two ways: by fixed reservation and on demand. In the allocation based on fixed reservation, the administrator reserves a fixed amount of Dom0 resources for each DomU, ensuring a minimum quality for each virtual network. The on-demand allocation guarantees high efficiency in resource usage, because XNetMon redistributes the idle resources among the DomUs that have a demand greater than their fixed reservation. We classify as idle resources all the non-reserved resources as well as the reserved resources that are not in use by the virtual networks. Thus, a premise of the controller is to provide the fixed resources of a virtual router i, represented as a percentage α_i of the total resources of the Dom0, $R(t)$, whenever there is a demand. Another premise is to allocate all the idle resources on demand to the virtual routers according to the priority preset by the administrator. This priority is a parameter called weight that belongs to $\{W_i \in \mathbb{Z} \mid 1 \leq W_i \leq 1,000\}$. The higher the

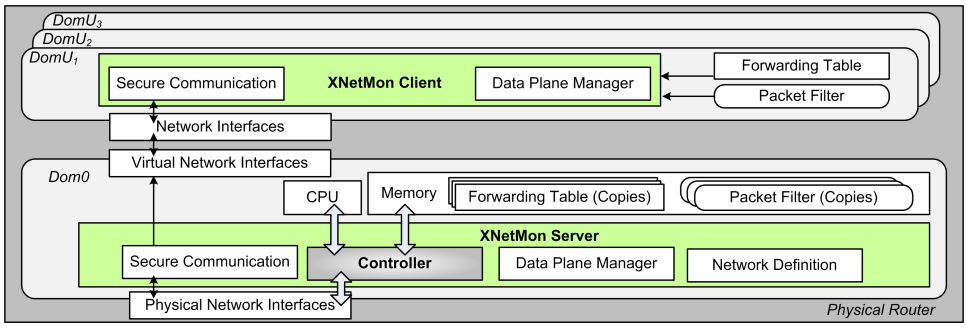


Fig. 2. XNetMon architecture with the main data flows and the monitored Dom0 resources, assuming three virtual routers.

weight of a virtual router, the more idle resources on Dom0 it has access to. Thus, the on-demand allocation provides an additional differentiated quality for each virtual network.

The controller monitors bandwidth by observing the volume of bits being transmitted by each output physical link. If a router exceeds the allocated bandwidth in an output link, it is punished by having its packets (destined to that link) dropped.

The CPU usage in Dom0 is monitored based on the volume of packets passing through Dom0. The monitored data is then weighted on the cost of each network operation. The packet processing cost is assigned according to the source and the destination of the packet because, as shown in Table I, the packet impact on the Dom0 CPU depends on whether the packet comes from/goes to a DomU or an external machine. If a router exceeds the allocated CPU, it is punished by having its packets dropped in the input interface. To avoid attacks that generate unfair CPU punishments, it is important to define the responsible domain for each measured operation. In transfers between DomUs, the DomU that sends the packet is responsible for all the costs of CPU usage, because we want to prevent an opponent domain from starting unsolicited traffic to exhaust CPU resources of a common domain. Besides, in transfers between DomU and Dom0, the CPU usage cost is always accounted for the DomU.

XNetMon controls memory usage by observing the size of the forwarding table of each virtual router. If the Dom0 memory reaches critical limits, the virtual routers whose tables/filters occupy more memory than the fixed reservation are punished, through the disposal of a percentage of routes. To avoid packet losses, a default route to the virtual router is added. Therefore, reducing the size of the routing table does not imply dropping packets, but only in a reduced forwarding performance because the packet is then forwarded by DomU instead of by Dom0.

1) Punishment Computing: Opponent domains are punished by having their packets or routes dropped. XNetMon controller searches and converges for a dropping probability that balances the use of Dom0 resources among virtual routers according to the fixed reservation and weight values. To avoid drops when there are idle resources on the physical machine, a virtual router is punished only if its usage overcomes its fixed reservation value and if the total resource usage reaches a critical level, given by a percentage β of total resources $R(t)$ in Dom0. With no idle resources, all nodes that use more than the fixed reservation are punished to avoid that other virtual routers cannot use their fixed reservation. Given that the total non-reserved resources is given by $D(t) = R(t) - \sum_{\forall i} \alpha_i R(t)$, then the dropping probability in $t+T$, given by $\Phi_i(t+T)$, is updated according to Algorithm 1.

Algorithm 1: Heuristics for punishment computing.

input : $\Phi_i(t)$, W_i, α_i, $R(t)$, $U(t)$, $U_i(t)$, $D(t)$, β
output: $\Phi_i(t+T)$
if *($\alpha_i \cdot R(t) < U_i(t)$) or ($\Phi_i(t) > 0$)* **then**
  if *($\alpha_i \cdot R(t) < U_i(t)$)* **then**
    % Calculate an idle resource usage indicator
    $\Upsilon_i(t) = (U_i(t) - \alpha_i \cdot R(t))/D(t)$
    if *($\beta \cdot R(t) \leq U(t)$)* **then**
      % Since there are no idle resources, some network can be damaged. Thus, we increase punishment.
      if *($\Phi_i(t) > 0$)* **then**
        $\Phi_i(t+T) = min(\Phi_i(t) + (1+\Upsilon_i(t)) \cdot (1+\frac{1}{W_i}) \cdot \frac{\Phi_i(t)}{(3-\frac{1}{W_i})}, 1)$
      else
        $\Phi_i(t+T) = \Phi_{initial}$ % Set initial punishment
      end
    else
      % Reduce punishment, because there are idle resources
      $\Phi_i(t+T) = max(\Phi_i(t) - (1+(1-\Upsilon_i(t))) \cdot (1-\frac{1}{W_i}) \cdot \frac{\Phi_i(t)}{(3+\frac{1}{W_i})}, 0)$
    end
  else
    % Fastly reduce punishment, because the router used only its fixed resources.
    $\Phi_i(t+T) = max(\Phi_i(t) - 3 \cdot (1-\frac{1}{W_i}) \cdot \frac{\Phi_i(t)}{3}, 0)$
  end
else
  $\Phi_i(t+T) = 0$
end

It is important to note that even if a DomU consumes fewer resources than its fixed reservation value, the punishment is not immediately reset to avoid instabilities. Also, to prevent that traffic generated by virtual routers interrupt other Dom0 services due to CPU overload, a residual punishment is constantly applied in the output interfaces of virtual machines. Such punishment should be small enough to not impact the low-volume transmissions, but should prevent that a DomU consumes all the resources of Dom0.

B. Data plane manager module

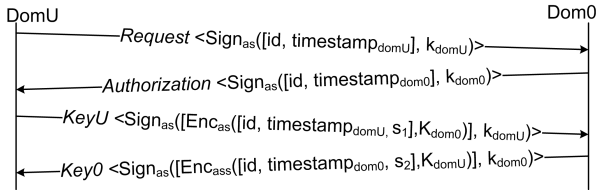
As mentioned earlier, for a higher performance, it is essential to forward packets through Dom0. However, by separating the packet forwarding from the routing control, the virtual routers are unable to update their forwarding tables and their

packet filters, because they have no access to Dom0 memory. XNetMon monitors tables and filters in DomUs and make a replica of them on Dom0 through the data plane manager module. Therefore, the XNetMon client in each DomU monitors changes in the forwarding table and packet filters, and the XNetMon server in Dom0 maps both the forwarding table and the packet filter built in each DomU to Dom0.

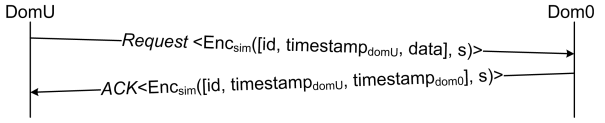
Every change in the forwarding table or packet filter in DomU must be immediately updated in its replica in Dom0. For this reason, after every control message arrival, the data plane manager client checks for changes in the forwarding table and packet filters in DomU. If any difference is found, XNetMon client transmits the changes to Dom0 via the secure communication module. When the XNetMon data plane manager server receives a message notifying a forwarding table change, it searches for the settings of that DomU to find out in which Dom0 table to insert the change. In packet filter updates, the server modifies the received rule, inserting rule parameters that specify the characteristics of the virtual network. This avoids that one virtual router creates rules in the packet filter that influence other virtual router traffic.

C. Secure communication module

The secure communication module creates a secure communication channel between the Dom0 and the DomUs, providing mutual authentication and privacy in data transfer. Since it is often used to update the data plane in Dom0, this must be a light module. Mutual authentication is required to ensure that no opponent domain can forge the identity of a common domain or of Dom0 to generate spoiled information in the data plane that corresponds to the attacked domain.



(a) Establishment of a session key, $s = f(s_1, s_2)$.



(b) Message exchange to update data plane.

Fig. 3. Creating the secure channel between Dom0 and DomU.

The secure communication module is composed of two protocols: one based on asymmetric cryptography for exchanging session keys, as described in Fig. 3(a), and other based on symmetric cryptography for securely transmitting data between a DomU and the Dom0, described in Fig. 3(b), where k is the private key, K is the public key, $Enc_{sim}([M], key)$ and $Enc_{as}([M], key)$ are the encryption of the message M with the key for symmetric and asymmetric cryptography, respectively, $Sign_{as}([M], k)$ represents message M and its signature with k, and id is the source node identity.

These protocols avoid replay attacks, in which the opponent domain repeats old control messages to spoil information in the data plane of the attacked domain. Hence, Dom0 and each DomU exchange *timestamps* during the establishment of the session key and then both know the difference between their clocks, T_{d0} and T_{du}, according to equation $\delta_{est} = |T_{d0} - T_{du}|$. Thus, whenever an update message is sent, the source inserts into the message its current timestamp to the destination check whether the message is a replay. If so, the timestamp does not satisfy

$$|T_{d0_n} - T_{du_n}| - e_{max} < \delta_{est} < |T_{d0_n} - T_{du_n}| + e_{max}, \quad (1)$$

where e_{max} is the maximum transmission delay and T_{d0_n} and T_{du_n} are the current timestamps of Dom0 and DomU[2].

Therefore, we can affirm that the communication between Dom0 and DomU is secure because XNetMon checks the authenticity, the privacy, and the non-reproducibility of data.

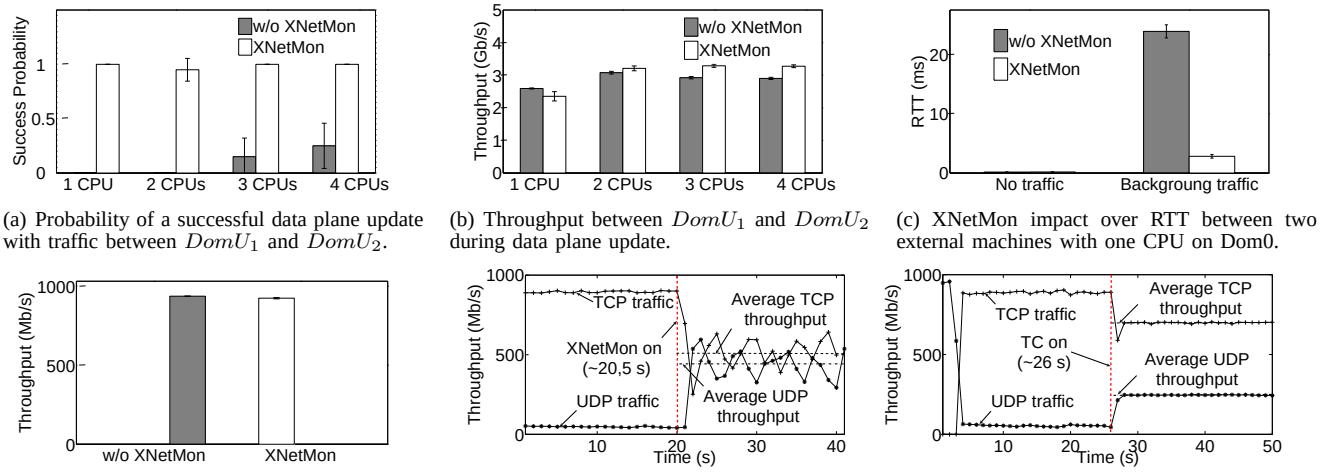
V. Prototype description and analysis

We developed a prototype to analyze the effectiveness of XNetMon in the presence of opponent domains and verify the efficiency of the controller to share resources. The prototype was implemented in C and Python and provides the data plane manager module, the secure communication module, and a controller able to monitor both bandwidth and CPU of Dom0. Monitoring and punishment were implemented with Iptables. We use Blowfish, which is considered a lightweight protocol for symmetric encryption, and RSA to implement the message exchange described in Fig. 3.

We performed the tests on a machine, hereafter called router, equipped with an Intel core2 quad processor with 4GB of RAM, using Xen 3.4-amd64 in router mode. The router has five physical Ethernet interfaces of 1 Gb/s each. We instantiated four virtual machines running Debian operating system with Linux kernel 2.6-26-2, each with one virtual CPU, 128 MB of memory, and five network interfaces. The number of virtual CPUs in Dom0 varies according to the test and there are no memory constraints for this domain. The physical CPUs are shared by all virtual CPUs and the hypervisor dynamically maps virtual CPUs to the real CPUs. The tests use two external machines that generate or receive packets, each with a network interface of 1 Gb/s. All traffic is generated with Iperf and the results present a confidence interval of 95%.

The first test evaluates the availability of the secure data plane update. The test is considered successful if the DomU updates its data plane using the secure communication protocol and no operation of other domains that passes through Dom0 prevents the data plane update. We compared the availability of the secure data plane update of XNetMon with the conventional plane separation, described in Section II. We used the secure communication module of XNetMon to secure the conventional plane separation. The test consists of a maximum of three attempts from $DomU_1$ to update the data plane, while $DomU_2$ sends TCP traffic to $DomU_1$. The scenario simulates an opponent virtual router, $DomU_2$, trying to prevent a common router, $DomU_1$, from normally operating. In XNetMon, $DomU_1$, which is trying to update

[2]Known solutions, such as the use of NTP, avoid that the drift in the clocks of the DomU interferes with XNetMon. Other possibility is to re-run the protocol for exchanging session keys whenever (1) is false.



(a) Probability of a successful data plane update with traffic between $DomU_1$ and $DomU_2$.

(b) Throughput between $DomU_1$ and $DomU_2$ during data plane update.

(c) XNetMon impact over RTT between two external machines with one CPU on Dom0.

(d) Background traffic throughput between two external machines with one CPU on Dom0.

(e) Resource sharing in XNetMon with UDP traffic from DomU and TCP traffic from external machine

(f) Resource sharing in TC with UDP traffic from DomU and TCP traffic from external machine

Fig. 4. Availability of data plane update and resource usage control in XNetMon.

the data plane, has $\alpha_1 = 0.5$ and the opponent, $DomU_2$, has $\alpha_2 = 0.3$. All DomUs have weight $W = 500$.

Fig. 4(a) shows the success probability of data plane update and Fig. 4(b) shows the volume of data transmitted between the virtual machines. The $DomU_2$ attack is effective when using the conventional plane separation, even if there is a great number of CPUs in Dom0. XNetMon, however, increases by up to 100% the probability of a successful data plane update. Due to the controller, XNetMon limits attack traffic from the $DomU_2$ avoiding the overload of Dom0 resources. Also, the XNetMon controller reserves the CPU resources required by $DomU_1$ to send the update messages as well as to perform cryptographic operations. Fig. 4(b) shows that XNetMon punishes traffic from $DomU_2$ to $DomU_1$ to ensure that the Dom0 CPU resources are not exhausted. Therefore, the throughput achieved when using XNetMon is smaller than when using the secure data plane update with only one CPU in Dom0. When we increase the number of CPUs in Dom0, the CPU restriction is relaxed and the throughput using XNetMon increases. Indeed, XNetMon throughput is even greater than the throughput of the secure data plane separation. The XNetMon controller ensures the fixed resources of $DomU_1$, and then $DomU_1$ can handle the data plane update as well as the ACK messages of the TCP connection started by $DomU_2$. Losing ACK messages is worse to throughput than the limitation imposed by XNetMon controller to traffic due to the CPU consumption. Thus, XNetMon ensures a secure data plane update with high availability and also ensures a high performance connection between virtual machines because of the proposed architecture with the controller module.

The second test evaluates the transmission delay of XNetMon when compared to conventional plane separation. This test measures the delay caused by XNetMon overhead according to Dom0 workload. Because we are not evaluating fairness in resource sharing, we created a virtual network with a fixed reservation of 100%. The test consists of two experiments that measure the Round Trip Time (RTT) between two external machines using Ping. In the first experiment, there is no background traffic, whereas in the second experiment background TCP traffic was generated between the two external machines. The results of both experiments are in Fig. 4(c). Without background traffic, data transmission presents a low RTT for both configurations. However, when there is background traffic, the Dom0 CPU is overloaded, increasing the response time of the system, and, consequently, increasing the RTT. The results show that the CPU and bandwidth control provided by XNetMon prevents that the Dom0 CPU is overloaded and, thus, XNetMon presented an RTT up to eight times lower than the conventional plane separation configuration. It is important to note that even though XNetMon control implies in dropping packets, these drops do not cause a major impact on traffic, as shown in Fig. 4(d).

The third experiment concerns sharing output links. In this experiment, a DomU and an external machine on different virtual networks initiate a communication with another external machine. Thus, both networks share the output link to the destination machine. Both networks have equal access to physical resources, with $\alpha = 0.5$ and $W = 500$ in both virtual routers. To assess XNetMon, we also test the bandwidth control using Traffic Control (TC), a widely used tool for traffic control on Linux machines. In TC experiments, we use the Hierarchy Token Bucket (HTB) to create two output queues, each one with a minimum bandwidth of 512Mb/s and a maximum bandwidth of up to 1Gb/s to simulate the same resource usage policy than XNetMon. Figs. 4(e) and 4(f) present the results when the DomU sends UDP traffic with a maximum rate of 1.5 Gb/s and packets of 1500 B while the external machine sends TCP traffic. In the beginning, there is no control in the network and from the specified moment XNetMon or TC controls traffic.

The results show that external machine traffic has priority over DomU traffic when there is no control on the resource sharing. This is an isolation failure when using plane separation, because external traffic influences the maximum volume of traffic generated by a DomU. Thus, an external machine that belongs to an adversary network could generate attack traffic to

damage the performance of a virtual router of another virtual network. In this test, we equally share the resources between the two virtual networks and then both network should have an equal slice of the link, which means 512 Mb/s for each virtual network. Although XNetMon presents a larger variation in traffic than TC, XNetMon average throughput has a lower error with respect to the ideal rate of 512 Mb/s for each virtual network than the TC average throughput. In fact, if there is no control of the external machine inflow, UDP traffic from the virtual machine is underprivileged and is unable to achieve high rates. Therefore, XNetMon control presented a maximum throughput error with respect to the ideal rate of 512 Mb/s of -14.2 % for UDP traffic and of -0.62 % for TCP traffic and TC presented a maximum throughput error with respect to the ideal rate of 512 Mb/s of -52.18 % for UDP traffic and of $+35.68$ % for TCP traffic. Thus, the XNetMon presented a higher fairness in the link resource sharing because it is adapted to the Xen architecture particularities.

VI. Related Work

Isolation in Xen is a known issue. Jin et al. proposed a mechanism to ensure fairness in the use of L2 and L3 cache on Xen, which is not contemplated by the isolation mechanisms of the Xen hypervisor [4]. Another proposal evaluates the performance of virtual routers created with Xen [5], investigating the routing bottlenecks caused by CPU and memory when forwarding packets at Dom0. The authors use Click, which is a platform for building modular routers, to create different data planes with efficiency and flexibility. However, authors do not provide mechanisms differentiate the use of Dom0 resources between the virtual routers, as well as they do not present any scheme to do a secure data plane update guaranteeing authentication, privacy and availability. Thus, these proposals and XNetMon are complementary approaches.

Isolation is also a concern for other virtualization platforms. Trellis [6] provides isolation between virtual networks on the VINI platform [7]. In this platform, the system is virtualized on the operating system level and then all virtual environments share the same kernel. The main problem of the approaches based on operating system virtualization [6], [3] is that all control planes execute on the same operating system restricting flexibility. Another virtualization platform is OpenFlow [8], which is based on a centralized control plane. FlowVisor [9] creates slices in OpenFlow networks by controlling CPU, memory and bandwidth usage by each virtual network.

An alternative to provide fairness is the use of a hardware-based isolation, but these approaches increase equipment costs. Anwer et al. propose the use of NetFPGAs to provide fairness [10]. Other proposal is the direct I/O model [11], in which virtual machines directly access the hardware. A different queue is provided to each virtual machine in the hardware to guarantee fairness. Although this approach provides high performance on packet forwarding, it violates the driver domain model, breaking fault isolation and device transparency [10].

XNetMon is a software-based approach for the Xen platform. The main difference from XNetMon to other proposals is that our proposal deals with the issue of creating a secure plane separation and prevents external machines or opponent DomUs from hindering other virtual networks.

VII. Conclusions

XNetMon is a network monitor for safely isolating virtual networks created with the Xen platform. Our research on XNetMon raises several questions about isolation, flexibility, performance, and security in the design of virtual routers. The proposal guarantees a secure data plane update due to the use of the secure communication and the controller. The results show that the proposal properly shares Dom0 resources and reduces delays in packet forwarding, because the controller prevents Dom0 from being overloaded, which increases the response time of this domain. In fact, the results show that an overload on Dom0 affects routing, management, and control performed in Dom0. XNetMon also provides high availability in the data plane update, which is essential for a secure plane separation. These results were mainly obtained due to the efficiency of the XNetMon controller, which is adapted to the Xen architecture, providing an appropriate division of resources. Therefore, XNetMon provides isolation between the virtual networks, preventing the opponent domain actions under different scenarios.

As future work, we intend to implement the memory control and to study the parameter setting on the controller to a faster stabilization of the punishment. In addition, we intend to integrate XNetMon with other proposals to provide a flexible data plane in Dom0.

References

[1] N. C. Fernandes, M. D. D. Moreira, I. M. Moraes, L. H. G. Ferraz, R. S. Couto, H. E. T. Carvalho, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "Virtual networks: Isolation, performance, and trends," *To be published in the Annals of Telecommunications*, 2010.

[2] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley, "Evaluating Xen for router virtualization," in *ICCCN'07: International Conference on Computer Communications and Networks*, Aug. 2007, pp. 1256–1261.

[3] Y. Wang, E. Keller, B. Biskeborn, J. V. der Merwe, and J. Rexford, "Virtual routers on the move: Live router migration as a network-management primitive," in *ACM SIGCOMM*, Aug. 2008, pp. 231–242.

[4] X. Jin, H. Chen, X. Wang, Z. Wang, X. Wen, Y. Luo, and X. Li, "A simple cache partitioning approach in a virtualized environment," in *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, 2009, pp. 519–524.

[5] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Fairness issues in software virtual routers," in *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, Aug. 2008, pp. 33–38.

[6] S. Bhatia, M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, "Hosting virtual networks on commodity hardware," Princeton University, Georgia Tech, and T-Labs/TU Berlim, Tech. Rep. GT-CS-07-10, Jan. 2008.

[7] *VINI - A Virtual Network Infrastructure*, http://www.vini-veritas.net/, Acessed in august 2010.

[8] 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 Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[9] R. Sherwood *et al.*, "Carving research slices out of your production networks with OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 129–130, 2010.

[10] M. B. Anwer, A. Nayak, N. Feamster, and L. Liu, "Network I/O fairness in virtual machines," in *VISA '10: Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. New York, NY, USA: ACM, 2010, pp. 73–80.

[11] "Advanced virtualization I/O queuing technologies - an Intel-Microsoft perspective," White paper, Intel, 2009. [Online]. Available: download.intel.com/network/connectivity/virtualization/321968.pdf