

XTC: A Throughput Control Mechanism for Xen-based Virtualized Software Routers

Rodrigo S. Couto, Miguel Elias M. Campista, and Luís Henrique M. K. Costa
Universidade Federal do Rio de Janeiro - PEE/COPPE/GTA - DEL/POLI
Email: {souza,miguel,luish}@gta.ufrj.br

Abstract—Xen is a tool for hardware virtualization often used to build virtual routers. Xen, however, does not assure the fundamental requirement of network isolation among these routers. This work proposes XTC (Xen Throughput Control) to fill this gap, and therefore, to guarantee multiple network coexistence without interference. XTC sets the amount of CPU allocated to each virtual router according to the maximum throughput allowed. Xen behavior is modeled by using experimental data, and based on these data, XTC is designed using feedback control. Results obtained in a testbed demonstrate the XTC ability to isolate virtual network capacities and to adapt to system changes.

I. INTRODUCTION

Today, more and more researchers are engaged in a discussion about future Internet directions. Some advocate the clean-slate architecture, which is based upon the idea that the Internet must be redesigned to accommodate requirements not initially considered, such as security, reliability, etc [1]. Embracing all current Internet requirements in a unique architecture may not be feasible, though. Therefore, another solution is to allow multiple independent networks running in parallel using a virtualized substrate. In this case, each network would be a virtualized network configured to suit specific requirements [2].

A key requirement of a virtualized infrastructure is to keep isolation among the different virtual networks [2]. Nevertheless, providing adequate isolation is still an open issue. Xen [3], often used to build virtual routers, virtualizes machine hardware among different operating systems running in parallel. In Xen, the virtual machines (VMs) play the role of virtual routers, allowing different networks to run concurrently. Nevertheless, only one privileged VM, using hypervisor calls, is responsible for granting access to I/O resources to the other VMs. Consequently, this privileged VM may become a bottleneck for certain operations, breaking down the isolation.

Up to date, resource sharing between VMs is a subject well investigated in data centers [4] [5], but an open issue in virtual networks. Anwer *et al.* [6] use a hardware-based control to reduce the network bottleneck in Xen. They use the NetFPGA platform to limit incoming packets in each virtual interface before they reach the hypervisor. Consequently, if a virtual machine is receiving more packets than permitted, their control mechanism blocks them, making sure that no extra packets go to the hypervisor and waste resources. Fernandes and Duarte [3] propose a control mechanism that handles resource sharing among virtual routers from the secu-

rity viewpoint. They monitor resource utilization and punish misbehaving routers.

In this work, we propose XTC (Xen Throughput Control), a software-based control mechanism to orchestrate the amount of machine resources provided to each virtual router. Our goal is to guarantee virtual network isolation, which is not possible with standard Xen. Unlike previous work, our mechanism controls the capacity of a *virtual router* to forward packets instead of using per interface control. This characteristic gives more flexibility and scalability to XTC. We tackle the problem using Xen because of its high programmability power and flexible resource management, such as memory and CPU. To control the virtual router packet forwarding capacity, XTC controls the amount of CPU time assigned to each virtual router to achieve a desired throughput. Our proposal thus provides isolation by limiting the throughput that a virtual router can forward, preventing this router from interfering on other virtual routers on the same machine. In addition, XTC can be used to perform virtual router differentiation during contention periods by limiting the throughput of each virtual router. XTC controls the amount of CPU given to each virtual router on the fly, allowing the system to adapt to dynamic network conditions. To analyze the performance of XTC, we first conduct experiments to tune its parameters. Then, we show that XTC can be used to differentiate virtual routers and, finally, that XTC can adapt to system changes. This paper has been accepted for publication in the Proceedings of the IEEE GLOBECOM 2011 [7].

This paper is organized as follows. Section II reviews CPU scheduling in Xen. Section III gives an overview of our proposed system. Our testbed is described in Section IV and Section V shows the mathematical model used to analyze the behavior of Xen. Section VI shows the design of XTC, while Section VII evaluates it experimentally. Finally, Section VIII concludes this work and points out future directions.

II. XEN CREDIT SCHEDULER

The Xen hypervisor manages the amount of resources that each virtual machine (VM) can use. Concerning CPU allocation, the hypervisor uses by default the Xen Credit Scheduler [8], which controls the slice of CPU time given to each VM. The amount of CPU time is adjusted based on two parameters: *weight* and *cap*. The former defines weights for each VM and the scheduling decision gives priority to VMs with higher weights, in case of contention for CPU. On

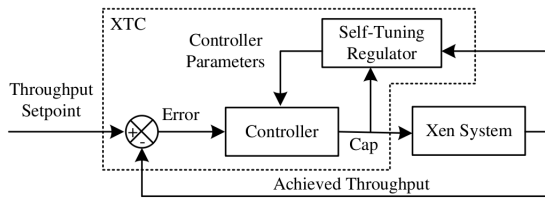


Fig. 1. XTC Feedback Control Loop.

the other hand, cap imposes a hard limit on CPU utilization by indicating the maximum percentage of CPU time given to each VM. The two parameters are configured on the fly from a privileged VM, called Domain 0 (Dom0). Dom0 is also responsible for managing shared I/O operations.

Limiting the slice of CPU is useful to control the runtime of the tasks running on each VM. Tasks such as processing, disk writing, packet forwarding etc. can have their runtime controlled. In this work, the VM concept is used to create routers which have as their main task packet forwarding. Hence, when controlling the CPU slice given to each virtual router, we can limit the maximum throughput each one can achieve forwarding packets. We limit CPU using cap because this parameter gives more control to our mechanism: cap gives a hard limit as opposed to weight which acts only when the CPU is saturated.

III. XEN THROUGHPUT CONTROL

This work proposes Xen Throughput Control (XTC), which controls the throughput of Xen-based virtual routers. XTC adjusts the cap of each virtual router according to the maximum throughput desired. In this work, we define throughput as the total bit rate forwarded by a virtual router. In other words, the throughput is the aggregated bit rate forwarded through all virtual router interfaces. Using throughput limitation we can provide isolation among the different hosted virtual routers. XTC is as flexible proposal for throughput control in virtual networks because it does not individually control the throughput of each virtual router network interface. Instead, XTC controls the aggregated traffic, acting on the virtual router capacity to forward packets using CPU slice assignment. Our mechanism then orchestrates the aggregated throughput of a virtual router, limiting only its influence on Dom0, leaving to the administrator of a virtual router the liberty to manage the traffic on each interface. Furthermore, XTC is scalable because the number of controllers does not increase with the number of network interfaces on each router.

XTC uses a feedback control loop that acts on the cap entitled to each virtual router in order to achieve a throughput setpoint, depicted in Figure 1. To accomplish that, XTC periodically measures the achieved throughput in the virtual router and computes the error between this measure and the throughput setpoint. This setpoint represents the desired throughput in a virtual router. The error is then used by the Controller block where we implement a Proportional Integral (PI) controller to compute and adjust the virtual router cap

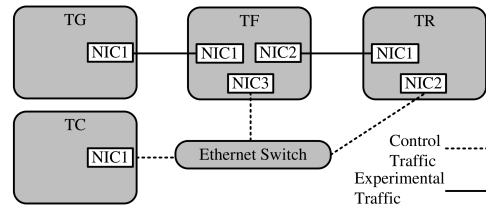


Fig. 2. Experimental Testbed.

according to this error. The PI controller was chosen because it has zero steady-state error, combined with short settling time. A shorter settling time could be achieved by using a PID (Proportional Integral Derivative) controller. However, the derivative factor of PID can cause oscillation in the real implementation of systems with high output variability, such as computer networks. XTC samples the achieved throughput every 1 second. The Xen System block represents the behavior of the virtual router throughput according to the cap entitled to it. We model this block using experimental data. This block is important to the Controller design, because it represents the basis on which the PI controller parameters are chosen. We first manually evaluate the parameters and later we use the Self-Tuning Regulator to autonomously evaluate the Xen System model and choose new PI controller parameters.

For a complete resource allocation system we have one XTC per virtual router and a policing mechanism which controls all XTCs. This mechanism can configure the throughput setpoint, activate or deactivate each XTC and tune other XTC parameters. The actions taken by the policing mechanism are based on its knowledge about the system environment obtained via resource utilization measurements and policies specified by the system administrator. These policies can be based on SLAs (Service Level Agreements). As an example, this mechanism can detect a bottleneck situation and then activate XTC to limit the throughput of a virtual router in order to meet other routers' requirements. In this paper we focus on XTC design and thus the policing mechanism is manually executed.

IV. EXPERIMENTAL TESTBED

Figure 2 illustrates our testbed, composed of four PCs, which is used to model the Xen System block and to perform experimental analysis of XTC. The Traffic Generator machine (TG) produces all data traffic destined to the Traffic Receiver (TR) machine. The Traffic Forwarder (TF) machine hosts the virtual routers used in our experiments. In our Xen configuration Dom0 has two exclusive CPU cores while virtual routers share another core. TF runs the Xen hypervisor version 3.4.2 and has instantiated virtual routers which forward packets from TG to TR. The Traffic Controller (TC) machine runs XTC. Note that the Traffic Generator (TG) and Traffic Receiver (TR) are directly connected to the Traffic Forwarder (TF) whereas TC is connected to TF and TR through different links to avoid mixing control and data traffic.

TG, TR, and TC are general-purpose PCs equipped with an Intel Core I7 860 2.80 GHz processor in a Intel DP55KG

motherboard. These machines run Debian Linux kernel version 2.6.32. The TF machine is an HP Proliant DL380 G5 server equipped with two Intel Xeon E5440 2.83 GHz processors. This machine runs Debian Linux paravirtualized kernel version 2.6.26. On the one hand, TG and TR are connected to TF via their on-board Intel PRO/1000 PCI-Express network interfaces. TF, on the other hand, is connected to TG and to TR via the two interfaces of a PCI-Express x4 Intel Gigabit ET Dual Port Server Adapter.

V. XEN SYSTEM MODELING

The Xen System block models the behavior of virtual router throughput according to the cap entitled to it. We use a black-box approach to model the system [9] based on experiments using the following framework.

A. Training Data Acquisition

Using the testbed described in Section IV, with the TC machine turned off, we model the Xen System by using an experiment to capture the relationship between cap and throughput. We send packets from TG to TR through one virtual router inside TF using fixed packet rate and fixed packet length. This flow consists of UDP segments generated by Iperf during 30 seconds. The experiment is repeated for different cap values assigned to the virtual router. Figure 3 shows the throughput achieved using 64-Byte packets for different packet rates. The X axis shows the cap assigned to the virtual router. Note that the relationship between cap and throughput depends on the packet rate forwarded by the virtual router. The higher the rate, the more CPU is needed.

Figure 3 also shows that from a certain cap entitled on, the throughput stops increasing. In this case, the throughput obtained matches the bit rate produced, because the virtual router receives enough CPU resources. Nevertheless, below these cap values, the throughput changes according to the cap in a log-scale fashion. Thus, this region is considered in our system modeling. We perform the same experiment with 1470-Byte packets and we also observe the same behavior seen in the 64-Byte packet experiment unless by the fact that the throughput achieved is higher for each cap value, as expected. Consequently, in the remainder of this paper we use flows of 64-Byte packet to allow higher packet rates on a Gigabit link. Nevertheless, our results could extend to larger packets on 10 Gigabit links. It is important to note that, as our experiment uses cap, which is a percentage of CPU, the values of throughput obtained will depend on the hardware used. However, the behavior shown in the experiment will remain the same. As the design of XTC depends on the model obtained with this experiment, our mechanism requires an initial training for a certain hardware specification. Yet, considering the Self-Tuning regulator, which adjusts XTC according to system changes, this requirement is not an obstacle.

B. Model Evaluation

Using the results from Section V-A, we model the Xen System as a linear first-order system given by Equation 1.

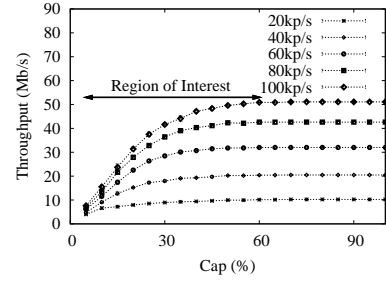


Fig. 3. Cap variation with 64-Byte packets.

This model will be useful in the Controller design.

$$y(k+1) = ay(k) + bu(k). \quad (1)$$

As we are modeling a non-linear system using a linear model, we consider the system as linear over an operating point. Consequently, the signals $y(k) = \tilde{y}(k) - \bar{y}$ and $u(k) = \tilde{u}(k) - \bar{u}$ are offset values from their operating points, where $\tilde{y}(k)$ and $\tilde{u}(k)$ are the actual values of the Xen System signals and \bar{y} and \bar{u} the operating points. The operating points are the mean values of $\tilde{y}(k)$ and $\tilde{u}(k)$ over the region of interest. In Equation 1, $y(k)$ and $u(k)$ indicate, respectively, the throughput achieved by the router and the $\log(\text{cap})$ in the system input at the k^{th} sample. We use $\log(\text{cap})$ instead of absolute cap because the relationship between cap and throughput has approximately a logarithmic behavior over the region of interest. Consequently, the first-order model of Equation 1 suits our purposes and simplifies the control system design.

The next step to model the Xen System is to obtain the variables a and b that characterize this system on Equation 1. The Xen System behavior and thus the variables a and b depend on the packet rate and on the packet length of the controlled flow. The parameters a and b can also model one aggregated flow with average packet rate and packet length.

We model the Xen System forwarding a 64-Byte flow at constant packet rate of 100 kp/s to show that a first-order system suits well our purposes. To estimate a and b , we employ the least squares regression method [9] using as an input the data obtained in Section V-A for the 64-Byte packet flow at 100 kp/s. We evaluate a and b over the operating point $\bar{y} = 40$ Mb/s and $\bar{u} = 1.39$. This region was chosen because cap still acts and the throughput does not saturate. In our example, this region corresponds to $\text{cap} \leq 60$ as indicated in Figure 3. We obtain $a = 0.0915$ and $b = 32259$ using MATLAB. To evaluate our model accuracy regarding the data collected, we compute the R^2 . This metric quantifies the variability of the output explained by the model and varies from 0 (worst model) to 1 (best model). In our case, we obtain $R^2 = 0.9899$ which suggests a very good fit.

VI. XTC DESIGN

The design of the main parts of XTC, the Controller and the Self-Tuning Regulator, are described below.

A. Controller

The Controller must decide which value of cap will be given to the virtual machine based on the difference between the throughput setpoint and the achieved throughput. We implement this block as a Proportional Integral (PI) controller, which evaluates periodically the cap using Equation 2. In this equation, $u(k)$ is the controller decision in the k^{th} sample, denoted as $\log(\text{cap})$, and $e(k)$ is the error computed by the difference between the setpoint and the achieved throughput.

$$u(k) = u(k-1) + (K_p + K_i)e(k) - K_p e(k-1). \quad (2)$$

The design issue of a PI controller is to choose K_p and K_i parameters to meet the system requirements, such as stability and small settling time. The first indicates that the system converges to a steady-state value, whereas the second indicates the time when the system would meet this value. Using the pole placement method we evaluate manually the controller parameters using the 64-Byte packet at 100kp/s. This method considers variables a and b of the model evaluated in Section V-B. The controller parameters that lead the system to a small settling time and a stable behavior are $K_p = -3.422 \times 10^{-6}$ and $K_i = 22.158 \times 10^{-6}$. The complete system seen in Figure 1, disregarding the effect of the Self-Tuning Regulator, was simulated with Simulink from MATLAB. Results, not shown here, show that the theoretical system is stable and has a settling time of 10 seconds.

Our proposed Controller also uses the concept of dead zone, where it decides to act only when the error exceeds a threshold. As the Controller acts using Dom0 calls, limiting the Controller actions reduces these calls. In systems where an external machine performs these calls, this concern becomes more important because it reduces the control overhead between the control machine and the machine running Xen. The threshold chosen in our implementation is 10% of the setpoint.

B. Self-Tuning Regulator

The manual evaluation of Controller parameters K_p and K_i is not suitable in systems with fast dynamics, such as routers, as it requires a prior evaluation of several parameters that are appropriated to each system behavior, and the Controller should detect when it will use each of these values. Furthermore, undetected changes in system dynamics or even unknown may cause undesirable behavior. To avoid these problems, XTC uses adaptive control techniques to self-tune according to system changes. The Self-Tuning Regulator block is responsible for adapting the Controller to changes on Xen System characteristics. This block periodically estimates constants a and b of Equation 1 based on the observation of the Controller decision $u(k)$ and the output $y(k)$ of the Xen System. This estimation uses the Gradient Projection algorithm given by Equation 3, where $\alpha = 0.001$ and $c = 0.0001$. Using the constants that characterize the Xen System, the Self-Tuning Regulator automatically evaluate new values of K_p and K_i using the Pole Placement method, as in Section VI-A. Therefore, the Self-Tuning Regulator aims at keeping the

system properties, thus meeting the desired requirements even when the system changes.

$$\theta(k) = \theta(k-1) + \alpha \epsilon(k) \phi(k), \quad (3)$$

where $\theta(k) = [b, a]^T$, $\epsilon(k) = \frac{y(k) - \theta^T(k-1)\phi(k)}{c + \phi^T(k)\phi(k)}$ and $\phi(k) = [u(k-1), y(k-1)]^T$.

VII. EXPERIMENTAL RESULTS

In this section, unless stated otherwise, the experiments use XTC with the Self-Tuning Regulator block disabled.

A. Practical Implementation

We implement our proposed mechanism in the testbed of Figure 2. Packets are sent from TG to TR at a fixed rate using Iperf. A virtual router hosted in TF forwards these packets. The Traffic Controller (TC) measures the throughput achieved by the virtual router and plays the role of the Controller as seen in Figure 1. To measure the achieved throughput, TC periodically collects the output of the Iperf Server reported by TR. In practice, the throughput measurement and also the XTC execution have to be performed on the machine with Xen (*e.g.* TF) to provide more scalability. We choose to separate these functions from TF to guarantee that our results are independent of this machine, which may be overloaded by high packet rates. TC executes XTC, computing the cap of the virtual router based on Equation 2 and remotely acting on the virtual router cap. Note that Equation 2 computes the $\log(\text{cap})$, rather than the absolute cap, and thus the actuator must compute the inverse of $\log(\text{cap})$. The complexity of this computation is negligible in our testbed. XTC relies on simple operations, allowing the control of large number of virtual routers.

The experiment consists of sending 64-byte packets from TG to TR at 100 kp/s during 100 seconds, which corresponds to a flow of 51.2 Mb/s. The TC machine must adjust the cap of the virtual router to track the throughput setpoint of 20 Mb/s. This value of setpoint is chosen to show the behavior of the system when it is quite far from the operating point but not so far as to cause undesired system behavior. Our first evaluation measures the average throughput achieved and the root mean square error (RMSE) with respect to this average, as seen in Figure 4. These measurements are computed using the values obtained during the interval from 20 to 100 seconds of each Iperf run. We use this interval to discard the system transient behavior before 20 seconds. The average throughput indicates whether the system achieved the throughput of 20 Mb/s as required. The RMSE, on the other hand, quantifies the oscillatory behavior of the system showing how the system response deviates from the average throughput.

We evaluate separately three different configurations. The first one, called FC (Fixed Cap), consists of turning off XTC and adjusting a fixed cap of 14% to the virtual router. This value is chosen because we expect an average throughput close to 20 Mb/s. In practice, this implementation is not recommended because one must know in advance the fixed cap value that leads to the specific throughput. Furthermore, the system behavior may vary because of traffic dynamics, which

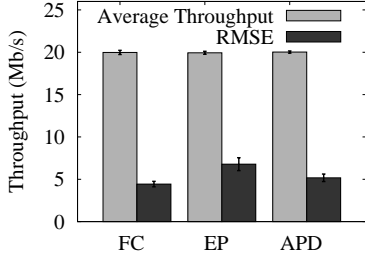


Fig. 4. Average throughput and RMSE measurement.

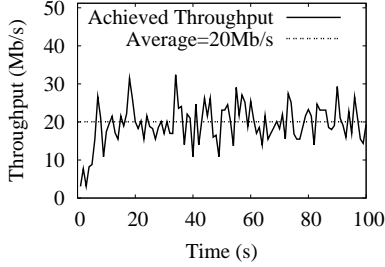


Fig. 5. Achieved throughput - APD XTC experiment.

justifies the use of a feedback controller to periodically adjust the cap. We use, however, this result as a reference to analyze XTC performance. Figure 4 shows that FC implementation obtains a high RMSE value, which indicates that the system oscillates when the throughput is limited using Xen’s cap parameter. Therefore, the Controller will have to cope with this particular behavior of cap adjustment. The EP (Evaluated Parameters) configuration uses the XTC controller parameters $K_p = -3.422 \times 10^{-6}$ and $K_i = 22.158 \times 10^{-6}$ as already evaluated. Results show that the average throughput obtained is close to the throughput setpoint showing the effectiveness of our proposal. The EP implementation, however, inserts more oscillation as compared with FC. Finally, we implement the APD (Adjusted Parameters with Dead Zone) configuration which has the controller parameters manually adjusted, with $K_p = -3.422 \times 10^{-6}$ and $K_i = 10.158 \times 10^{-6}$, in order to reduce the oscillation. This configuration also uses the concept of dead zone to reduce message exchange between TC and TF. In this experiment, we can suppress $29 \pm 2.4\%$ of the control messages using the dead zone concept. As seen in Figure 4, the APD configuration reduces the RMSE and achieves the desired throughput. Figure 5 exemplifies the system output obtained in a single run of the APD configuration.

In this section, XTC achieves a throughput extremely close to the desired throughput. Nevertheless, the system response oscillates around this value because of cap adjustment, which represents a tradeoff of the Xen platform (independent of XTC). In spite of this, we show that APD configuration adds negligible oscillation compared with the FC configuration.

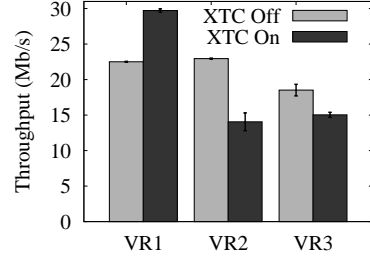


Fig. 6. Traffic differentiation using XTC

B. Traffic Differentiation

In this section we show the ability of XTC to provide traffic differentiation between virtual routers. XTC can dynamically guarantee higher throughput to a virtual router by limiting the amount of resources used by the other routers. This feature is also used to guarantee isolation among virtual routers, which is not possible with native Xen. In the default network implementation of Xen, all packets sent and received by the virtual routers are forwarded by Dom0. According to [10], Dom0 consumes a lot of CPU resources when doing this task and, even when reserving more CPU cores to Dom0, the performance of network-related tasks do not increase because they are single-threaded. Hence, Dom0 becomes a bottleneck and the packet rate of each virtual network influence each other. We conduct an experiment using TF hosting three virtual routers (VR1, VR2, and VR3) forwarding packets. In this experiment, TG sends to TR three 64-byte packet flows at 51.2 Mb/s during 100 seconds. Each virtual router forwards one of these flows. Although the virtual routers share the same CPU core, there is no contention for this core. Dom0, in turn, has two reserved CPU cores. First, we neither use XTC nor simple cap adjustment to measure the average throughput obtained in the last 80 seconds of each run. This configuration is named XTC Off in Figure 6.

Results show that the virtual routers cannot forward packets at the full rate, 51.2 Mb/s, because of the high contention for resources at Dom0. Consequently, the maximum throughput obtained in a virtual router is 23 Mb/s. To allow VR1 to forward more packets, we can limit the amount of packets that the other virtual routers can send to Dom0. As a consequence, VR1 has more opportunity to send packets to Dom0, increasing its throughput. We use XTC on each virtual router and repeat the latter experiment. For VR1, XTC uses the same K_p and K_i used in APD configuration of Section VII-A. The only difference is that we now limit the throughput to 30 Mb/s. For VR2 and VR3, XTC is configured to limit the throughput to 15 Mb/s. Because this rate is far from the model’s operating point used in Section VII-A, we also evaluate, for VR2 and VR3, a system model for the operating point of 27 Mb/s, resulting in $a = 0.00339$ and $b = 34816$. We then evaluate the controller parameters, as explained in Section VI-A, and we find $K_p = -4.825 \times 10^{-6}$ and $K_i = 18.530 \times 10^{-6}$. Results are labeled as XTC On in Figure 6. They demonstrate

that it is possible to assign priority to a virtual router using XTC. In our experiments, XTC was used in Xen's default configuration, where Dom0 is the bottleneck. Nevertheless, XTC can also be used when Dom0 is not the bottleneck, but when there is contention for resources on the CPU core shared by the virtual routers. This situation may occur when using Direct I/O techniques [11] where the network tasks are not intermediated by Dom0. In this case, XTC can also reduce the maximum throughput allowed to a virtual router, freeing some CPU resources from the shared core to improve the performance of the other routers. In the case of network virtualization with plane separation [12], where the data plane is implemented within Dom0 whereas control planes reside in DomUs, XTC must be redesigned, since it assumes that packets are forwarded by the virtual router.

C. Adaptability to System Changes

In this section we discuss the ability of XTC to adapt to system changes, using the Self-Tuning Regulator of Section VI-B. The experiment of Section VII-A is repeated using the same K_p and K_i parameters and, in the case of the Self-Tuning Regulator, these are the initial values of the Controller parameters. First, a flow of 51.2 Mb/s is generated and the setpoint of XTC is 20 Mb/s. In this scenario, the XTC performance is analyzed with and without the Self-Tuning Regulator block. Results shown in Figure 7 are represented, respectively, with the labels A_20 and N_20. As in the case of Section VII-A, results show that XTC without the Self-Tuning Regulator block can achieve the desired throughput. This is true because the distance of this throughput value compared with the operating point does not cause undesirable system behavior. Using the Self-Tuning Regulator, the throughput of 20 Mb/s is also achieved but with an oscillation greater than with static parameters, as seen by the RMSE value. The same experiment is performed again but using a throughput setpoint of 15 Mb/s, which has a greater distance from the operating point. This result is shown in Figure 7 labeled N_15, where the throughput achieved is ten times less than the one desired. Using the Self-Tuning Regulator, thus, the Controller parameters are evaluated automatically in order to adapt XTC to the new system requirements. The obtained results are labeled A_15 in Figure 7, showing the performance of the adaptive control. The system achieves 15 Mb/s even when the initial controller parameters are evaluated to an operating point far from the desired. These experiments demonstrate the ability of XTC to adapt itself to system changes without needing to evaluate parameters in advance for each operating point or system state. The Self-Tuning Regulator can also be used to evaluate new controller parameters when traffic dynamic changes. However, the adaptive control introduces more oscillation in XTC when the controller parameters do not need to be adjusted, as in the case of A_20 experiment. This behavior is acceptable because, in this case, we have a generic system that is not specific for a certain operating point.

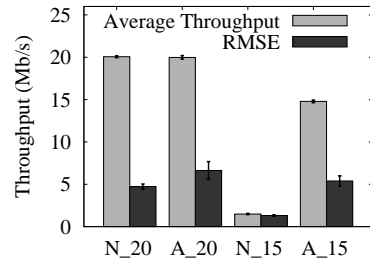


Fig. 7. Adaptability to system changes.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper we have addressed the issue of traffic isolation, a key challenge for network virtualization using Xen. Our preliminary results show that the traffic forwarded by Xen virtual routers interfere with each other using Xen's default implementation based on Dom0. To minimize this problem, we have proposed a traffic control mechanism (XTC) which adjusts the amount of CPU entitled to each virtual router according to the desired throughput. The experimental results show that, with the obtained isolation, XTC provides differentiation between virtual routers and can adapt itself to system changes. Furthermore, XTC is a flexible and scalable solution to control aggregated flows, avoiding the fine-grained control of individual virtual network interfaces. XTC can also be used as a building block for a larger virtual network resource allocation system, combined with a policing mechanism.

Our future work includes building a policing mechanism that adjusts the throughput of each virtual router based on service level agreements and knowledge about the network.

ACKNOWLEDGEMENT

This work was partially funded by CNPq, CAPES, FAPERJ, and FINEP.

REFERENCES

- [1] J. Rexford and C. Dovrolis, "Future Internet architecture: Clean-slate versus evolutionary research," *Communications of the ACM*, vol. 53, no. 9, pp. 36–40, 2010.
- [2] J. Carapinha and J. Jiménez, "Network virtualization: a view from the bottom," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. ACM, 2009, pp. 73–80.
- [3] N. C. Fernandes and O. C. M. B. Duarte, "XNetMon: A network monitor for securing virtual networks," in *Proceedings of the IEEE International Conference on Communications (ICC'11)*, Jun. 2011.
- [4] M. Kjaer, M. Kihl, and A. Robertsson, "Resource allocation and disturbance rejection in web servers using slas and virtualized servers," *Network and Service Management, IEEE Transactions on*, vol. 6, no. 4, pp. 226–239, 2010.
- [5] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 289–302, 2007.
- [6] M. Anwer, A. Nayak, N. Feamster, and L. Liu, "Network I/O fairness in virtual machines," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. ACM, 2010, pp. 73–80.
- [7] R. S. Couto, M. E. M. Campista, and L. H. M. K. Costa, "XTC: a throughput control mechanism for Xen-based virtualized software routers," in *IEEE Global Communications Conference (GLOBECOM'2011) - accepted for publication*, Houston, Texas, USA, Dec. 2011.

- [8] D. Ongaro, A. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in *ACM VEE*, 2008, pp. 1–10.
- [9] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-based control for dynamic sizing of resource partitions," *Ambient Networks*, vol. 3775, no. 1, pp. 133–144, 2005.
- [10] 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," *Annals of Telecommunications*, pp. 1–17, 2010.
- [11] J. Liu, W. Huang, B. Abali, and D. Panda, "High performance VMM-bypass I/O in virtual machines," in *USENIX*, 2006, pp. 29–42.
- [12] P. S. Pisa, N. C. Fernandes, H. E. T. Carvalho, M. D. D. Moreira, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "Open-flow and Xen-based virtual network migration," *The World Computer Congress 2010 - Network of the Future Conference*, pp. 170–181, 2010.