# Network Resource Control for Xen-based Virtualized Software Routers

Rodrigo S. Couto*, Miguel Elias M. Campista, Luís Henrique M. K. Costa

*Grupo de Teleinformática e Automação*
*PEE/COPPE - DEL/POLI*
*Universidade Federal do Rio de Janeiro*
*P.O. Box 68504 - 21945-970, Rio de Janeiro, RJ, Brazil*
*Tel.: +55 21 2562-8635 Fax: +55 21 2562-8628*

## Abstract

The pluralist architecture is considered an alternative for the future Internet to support multiple services with contrasting requirements. In this approach, machine virtualization techniques play a fundamental role. Nevertheless, when applied to networking, they impose critical bottlenecks since they do not provide suitable mechanisms to orchestrate the utilization of the underlying resources. In this work, we propose XTC (Xen Throughput Control) to fill this gap and control the utilization of network resources in Xen-based virtual routers. The main idea is to provide aggregate control, regardless of the traffic on specific network interfaces. To achieve this goal, XTC indirectly adjusts the maximum throughput of a virtual router by controlling the amount of CPU given to it. Our experimental results show that XTC provides differentiation and fairness between virtual routers and also adapts

*Corresponding author.
*Email addresses:* `souza@gta.ufrj.br` (Rodrigo S. Couto), `miguel@gta.ufrj.br` (Miguel Elias M. Campista), `luish@gta.ufrj.br` (Luís Henrique M. K. Costa)

to system disturbances.

*Keywords:* Future Internet, Network virtualization, Xen, Virtual resource management.

## 1. Introduction

An emerging proposal for the future Internet, called the pluralist approach, advocates for the utilization of different protocol stacks in parallel. Each protocol stack is designed to meet the individual requirements of a given application type [1]. For example, an application requiring security guarantees would use a secure protocol stack, while a mobile application would use a protocol stack specialized for node mobility. The simultaneous protocol stacks are supported through virtualized network elements [2]. In order to share the physical resources, virtualization platforms assign a network slice to each virtual network, which is composed of a given amount of link bandwidth but also CPU and memory resources from the physical routers. In this work, we use the Xen hypervisor as router virtualization platform. Xen [3] allows sharing a physical machine among different virtual machines (VMs), running different operating systems. To accomplish that, Xen allocates physical resources, such as CPU, memory, and disk, to the different VMs. Using Xen, we can build a virtual network infrastructure by assigning specific router functions to each VM. Hence, a virtual network is defined as a set of interconnected virtual machines which act as virtual routers (VRs). A VR can also be implemented using other virtualization platforms, such as KVM [4], VMware [5], or OpenVZ [6]. Another way of providing slice assignment is by using programming primitives to configure the network elements, a concept known as Software Defined Networks (SDN)[7]. An example of SDN is OpenFlow [8] along withFlowVisor [9]. This article assumes the slice assignment using virtual routers.

A key requirement of network virtualization is isolating virtual networks. For example, the traffic transported by a virtual network must not interfere with the traffic of another one. In order to provide isolation, virtualization platforms implement resource sharing at different levels. The Xen platform implements resource sharing at the hypervisor level, a software layer located between the physical and virtual machines. Using the hypervisor, Xen defines the physical memory and the disk space used by each virtual router. The Xen hypervisor implements a scheduler that controls the CPU time assigned to each virtual router. For example, we can configure the scheduler to allow a virtual router to use twice as much CPU as another router. Xen guarantees the isolation of memory, disk, and CPU resources. Nevertheless, *network* resource isolation in complete in Xen. In default configuration, Xen uses a single privileged virtual machine, called Domain 0 (Dom0), to multiplex the physical network interfaces between all the virtual machines. When a packet is received at a physical interface, Dom0 forwards it to the appropriate virtual router based on the destination IP or MAC address, using Xen's router or bridge mode, respectively. When a virtual router forwards a packet, it first sends it to a virtual interface, and then the packet is forwarded from this interface to the appropriate physical device. The task of sending and receiving packets from virtual routers is CPU-intensive in Xen. Consequently, Dom0 is a bottleneck for packet forwarding. Fernandes *et al.* [10] have shown that a Xen virtual router can perform seven times worse than a software router implemented on native Linux due to the Dom0 bottleneck. An approach that mitigates the bottleneck in Dom0 is plane separation [11]. With plane separation the virtual router only implements the control plane, it does not forward traffic. Traffic is instead forwarded by Dom0, which implements the data plane by mirroring the routing tables of each virtual router. The CPU cost of forwarding packets using only Dom0 is much smaller than having virtual routers forwarding packets [10]. Although plane separation is advantageous from the performance viewpoint, it is less flexible because the data plane must be the same for all parallel protocol stacks. Hence, in this work we focus on the case of virtual routers forwarding traffic, and design a solution for coping with Dom0 network bottleneck.

In addition to the bottleneck problem, Xen does not have a mechanism to orchestrate the amount of Dom0 resources used by each virtual router. As a consequence, we cannot differentiate virtual routers in terms of throughput when the utilization of Dom0 is high. Therefore, a mechanism is needed which limits the maximum amount of Dom0 resources that a virtual router can use, to guarantee isolation between them.

In this work, we propose a controller called XTC (Xen Throughput Control)[1]. This mechanism orchestrates the amount of Dom0 resources assigned to each virtual router by controlling the aggregate throughput each virtual router is allowed to forward. It is important to control only the aggregate throughput because the primary concern of XTC is to orchestrate the amount of Dom0 CPU resources that each virtual router uses. In this way, the

[1]A preliminary version of this work appeared in [12].

bandwidth utilization of individual virtual router network interfaces is left to another mechanism that is concerned only with the utilization of each physical link. The main idea of XTC is to indirectly control the aggregate throughput, by controlling the amount of CPU given to each virtual router. The CPU resources of each virtual router can be adjusted using the *cap* parameter of Xen. The operation of XTC is transparent to the virtual router administrator because the cap parameter is adjusted outside the virtual machine. Also, XTC does not require modifications to Xen's source code. The mapping between throughput and cap is performed by a feedback controller, which periodically measures the virtual router throughput and acts on the corresponding VM cap, according to the desired throughput. The dynamic characteristic of a feedback controller allows XTC to reject disturbances introduced in the system, for example when the virtual router is also using its CPU for other tasks in addition to packet forwarding. We experimentally analyze the performance of XTC with UDP and TCP flows. The results show that XTC effectively controls the throughput for a large operation range and provides service differentiation between the virtual routers. Also, we show that XTC can be used to maintain an equal distribution of resources among virtual routers where Xen alone fails.

Although this work focuses on the problem of Dom0 network bottleneck in Xen, XTC can be used to solve problems of network bottleneck in other platforms. The only requirement is that the platform provides an interface to control, on the fly, the CPU limit of the virtual routers. For example, OpenVZ has a parameter called *cpulimit* that plays the same role as *cap* in Xen, and can also be configured dynamically [6].

This article is organized as follows. Section 2 provides background for the Xen Credit Scheduler, which is used by XTC to provide throughput control. Section 3 gives an overview of our proposed system. Our experimental testbed is described in Section 4. Section 5 models the Xen system behavior used to design the proposed controller. Section 6 presents the design of XTC, while Section 7 shows the evaluation results. Section 8 describes the XTC implementation and provides scalability results. Finally, Section 9 discusses related work and Section 10 concludes the article and presents future work directions.

## 2. 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 [13], which controls the amount of CPU time given to each VM. The amount of CPU time can be 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 CPU contention. On 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 can be configured on the fly inside Dom0. These parameters can also be remotely configured by performing remote procedure calls to Dom0, using the Libvirt [14] API.

Limiting the slice of CPU is useful to control the runtime of each VM and, consequently, of all of the tasks running in the VM. Hence, tasks such as processing, disk writing, and packet forwarding, have their runtime controlled according to the amount of time given to the VM. In this work, each VM corresponds to a router, whose main task is packet forwarding. Thus, when controlling the CPU slice given to each virtual router, we can limit the maximum throughput that the router can achieve. We use cap to limit the CPU because *cap* gives a hard limit as opposed to *weight* which is effective only when the CPU is saturated.

## 3. XTC - Xen Throughput Control

This work proposes Xen Throughput Control (XTC), which controls the throughput of Xen-based virtual routers. This control is performed by adjusting the cap attributed to the virtual router according to the maximum throughput allowed to it. In this work, we define the throughput as the packet rate forwarded by a virtual router. In other words, the throughput is the aggregate packet rate forwarded by all the network interfaces of the virtual router. In our previous work [12], we used the bit rate to define the throughput, instead of packet rate. The packet rate differs from the bit rate by a multiplicative factor, the packet length. Nevertheless, controlling the packet rate instead of the bit rate is more intuitive for our purpose, since we are concerned with the CPU bottleneck of Dom0. The demanded CPU processing power increases with the number of packets forwarded, not with the bit rate. For example, a flow with bit rate of 100Mb/s with 10000-bit packets has a packet rate of 10kp/s, while a flow with the same bit rate but 100-bit packets has a packet rate of 1000kp/s. Although the two flows use the same link capacity, the last one consumes more processing power in Dom0.

XTC uses a feedback control loop as shown in Figure 1. XTC periodically

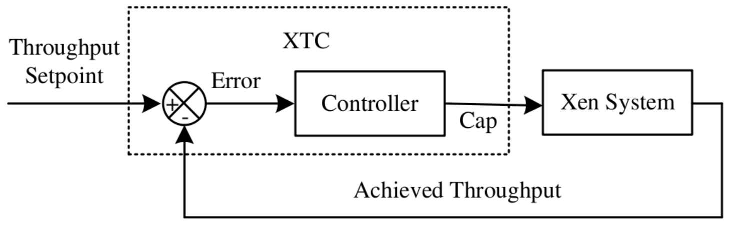


Figure 1: XTC Feedback Control Loop. The system periodically measures the Achieved Throughput and then computes the difference between this measurement and the Throughput Setpoint. The Error found is used as the input of the Controller block, which evaluates a new cap value to reach the desired Throughput Setpoint.

adjusts the cap of the virtual router cap to achieve the Throughput Setpoint specified by the administrator of the physical network. To accomplish that, XTC periodically measures the Achieved Throughput, which represents the packet rate that a virtual router forwards, and then computes the difference between this measurement and the Throughput Setpoint. This difference, which we refer to as Error in Figure 1, is the input of the Controller block. The Controller block is responsible for computing a new cap value based on a control law. In this work, we use a sample and control period of 1 second. Hence, the Achieved Throughput is measured every second and the cap is reconfigured at the same frequency. The Controller block is implemented by a Proportional Integral (PI) controller, which has a control law with zero steady-state error and short settling time. An even shorter settling time could be achieved by a PID ( Proportional Integral Derivative) controller. The derivative factor of a PID controller, however, can cause oscillation in systems with high output variability, such as computer networks.

The Xen System block models the behavior of the throughput according to cap adjustments. We have done this modeling based on experimental data. The Xen System model is important in the design of XTC because the parameters of the PI controller are chosen based on it. In this work, we manually evaluate the Controller parameters. In our previous work [12], however, we have used a Self-Tuning Regulator block that periodically estimates the Xen System model and proposes new controller parameters. This block was used to provide adaptation to changes in the system behavior that might not be captured by the initial model. For example, our model is a linearization of a non-linear behavior as shown in Section 5. In order to linearize the system, we choose an operating point, which means that the system will present a linear behavior around this point. As the Throughput Setpoint deviates from the specified operating point, the model parameters become less precise and, consequently, the controller parameters. We have shown in [12] that the Self-Tuning Regulator can adapt the Controller parameters when the Throughput Setpoint is far from the specified operating point. Nevertheless, further experiments revealed that the output of this block does not converge to a steady state value even if the system behavior remains unchanged, which may disturb the stability of the entire system. In this work, rather than trying to specify an adaptive control scheme, we evaluate in Section 7.2 two simple techniques to expand the system operating range and maintain its stability.

In a virtualized infrastructure, an XTC instance controls the packet rate of a single virtual router, defining its slice in terms of Dom0 network resources, i.e. the amount of Dom0 CPU time used to forward packets. A policing mechanism is responsible of specifying the Throughput Setpoint of each XTC instance and also of deciding whether it activates XTC or not. For example, when there is no contention for Dom0 network resources, the policing mechanism can deactivate all XTC instances. The network resource allocation system can also have controllers for bandwidth utilization on each link attached to the physical router. As an example, a bandwidth controller can use known tools such as Linux TC (Traffic Control) [15]. Similarly to the way the policing mechanism manages XTC by defining its Throughput Setpoint, it can manage the bandwidth controllers by setting the maximum capacity of a virtual network interface. XTC is transparent to the bandwidth controllers as it is exclusively concerned with Dom0 utilization.

## 4. Experimental Setup

Figure 2 illustrates our testbed used to model the Xen System block and to perform experimental analysis of XTC. The Traffic Generator machine (TG) produces traffic destined to the Traffic Receiver (TR). The Traffic Forwarder (TF) hosts the virtual routers. In our Xen configuration, two CPU cores are exclusively assigned to Dom0 and one CPU core is assigned to each virtual router. Therefore virtual routers do not contend for CPU cores. TF runs Xen hypervisor version[2] 3.4.2 and has instantiated virtual routers to forward packets from TG to TR. The Traffic Controller (TC) machine runs XTC. Note that the Traffic Generator (TG) and the Traffic Receiver (TR) are directly connected to the Traffic Forwarder (TF), whereas TC is con-

[2]We have used Xen 3.4.2 to produce results comparable with our previous work. Newer versions of Xen also provide cap configuration and can be used by XTC. Moreover, the bottleneck problem of Dom0 persists for newer Xen versions.

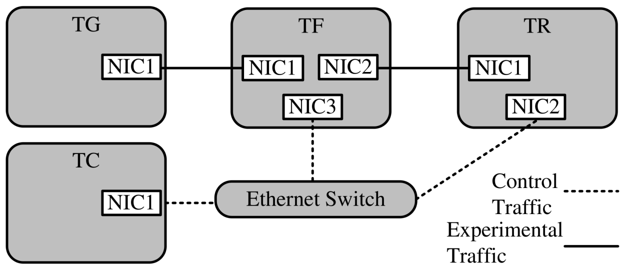


Figure 2: Experimental Testbed.

nected to TF and TR through different links, separating control traffic from data traffic.

TG, TR, and TC are general-purpose PCs equipped with an Intel Core I7 860 2.80 GHz processor and an 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. TG and TR are connected to TF via on-board Intel PRO/1000 PCI-Express network interfaces. TF is connected to TG and to TR via the two interfaces of a PCI-Express x4 Intel Gigabit ET Dual Port Server Adapter.

## 5. System Modeling

The Xen System block models the packet rate behavior of a virtual router according to its configured cap. To build this model, we use the black box approach [16], which is useful when the system internal variables are unknown. We use experimental data to model the relationship between the system input and output. The steps followed to apply this approach to Xen System modeling are described below.

## *5.1. Training data acquisition*

To acquire input data to model the Xen System, we perform an experiment using the testbed described in Section 4 with the TC machine turned off. In this experiment, we send a flow from TG to TR through one virtual router running on TF. The traffic used is an UDP flow with different constant bit rates generated by Iperf during 30 seconds. The experiment is repeated for different cap values manually configured in the virtual router. The packet length is 64 bytes. The small packet size was chosen to force high packet rates, before achieving the maximum network interface capacity. Also, we use UDP instead of TCP to model the extreme situation where the virtual router is trying to forward at a packet rate greater than the one limited by the cap with no congestion control mechanism. Later, we show that the proposed model is also valid for TCP flows.

Figure 3 plots the results with a confidence interval of 95% for the above experiment. The $x$-axis shows the cap assigned to the virtual router, whereas the $y$-axis shows the achieved throughput. Results are shown for different UDP packet rates. Note that the achieved throughput increases logarithmically with the configured cap, for different packet rates. Also, from a certain cap value on, the Achieved Throughput is almost constant. In the case of UDP, the plateau happens when the Throughput Setpoint is greater than the packet rate, since XTC is not effective and thus the Achieved Throughput becomes equal to the packet rate. For a TCP flow, it happens when the Throughput Setpoint is greater than the maximum throughput achieved by a TCP flow. We define as cap threshold the cap value from which the Achieved Throughput does not change significantly. Obviously, the cap threshold varies according to the packet rate. The higher the rate, the more CPU is needed to forward packets. Therefore, higher packet rates produce higher cap thresholds. The arrow in the figure shows an example of a cap threshold equal to 60 for a 100kp/s flow. We choose this value because, after it, the Achieved Throughput varies less than 1kp/s. The cap threshold will be used in the next section to select the data to model the system.

As XTC uses the cap parameter, which is a CPU percentage, the relationship between cap and throughput is hardware dependent. Thus the design of XTC depends on the Xen System model which, in turn, depends on the data acquired. Because these data are a function of the hardware specification, XTC needs an initial training whenever new hardware is used. It is important to mention, however, that the most important characteristic of the XTC design is the behavior of the throughput according to the amount of cap assigned to each virtual router, as seen in Figure 3. This behavior remains the same, and does not affect XTC design.

## *5.2. Model Evaluation*

In this work, we use the results of Section 5.1 to model the Xen System block. We consider only the data obtained before the cap threshold, when the variation of cap leads to throughput variation. We model the Xen System behavior in this region as a linear first-order system governed by Equation 1:

$$y(k+1) = a \times y(k) + b \times u(k), \tag{1}$$

where $y(k)$ represents the throughput achieved by the virtual router and $u(k)$ its $log(cap)$ at the $k^{th}$ sample. We use $log(cap)$ instead of cap because of the logarithmic relation between cap and throughput (Figure 3). This behavior

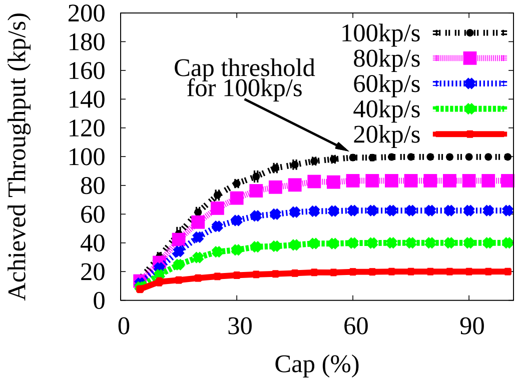


Figure 3: Measurement of the Achieved Throughput as a function of the cap values assigned to the virtual router. This experiment is done for different packet rates of an UDP flow.

leads to an approximately linear relationship between $y(k)$ and $log(cap)$. Consequently, the first-order model of Equation 1 suits our purposes while simplifying the control system design.

As we are modeling a non-linear system with a linear function, Equation 1 is a linearized model. In order to linearize our system, we choose an operating point and consider that the system is linear around this point. Therefore, 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 $\bar{u}$ are the operating points. Appendix Appendix A gives further details about the linearization method applied.

The next step to model the Xen System is to obtain the parameters $a$ and $b$ of Equation 1 that characterize the system. These parameters are chosen based on our experimental data. As the curves are different depending on the packet rate, as seen in Figure 3, we also have different $a$ and $b$ values.

Moreover, these parameters change according to the choice of the operating point. In this work, we fix the operating points $\tilde{y}$ and $\tilde{u}$ to their mean values over the region of interest, i.e. the region before the cap threshold. The proposed model is also appropriate for aggregate flows by considering them a single flow sending packets at the average rate.

We choose to model the system when the virtual router receives a 100kp/s flow, which has the largest region of interest. We use this model in the entire work. As we show later in Section 7.1, the Controller block designed using the model for a 100kp/s rate is also suitable for smaller packet rates. Thus, to accommodate a larger region of interest, we could build our model based on higher flow rates, depending on the limits the hardware used. Our experiments show that this model is also suitable for TCP flows. Using the black box approach to model the Xen System, we first obtain experimental data by varying the system input and observing its output, as done in Section 5.1. Using these results, the parameters $a$ and $b$ are evaluated using the least squares regression method [16]. We evaluate $a$ and $b$ over the region of interest (i.e. the region before the cap threshold of 60), which has the operating points $\bar{y} = 78.14$ and $\bar{u} = 1.39$. We obtain $a = 0.0915$ and $b = 63.0051$ using the `mldivide` function of MATLAB, which implements the least squares regression method for a given data set. To evaluate the accuracy of our model regarding the experimental data, we use the $R^2$ metric [17], which quantifies the variability of the output captured by the model and varies from 0 (worst case) to 1 (best case). The worst case happens when the model is not better than using the mean value of the data set outputs as estimation. In our model, we find $R^2 = 0.9899$, suggesting a very good fit. Figure 4 shows a

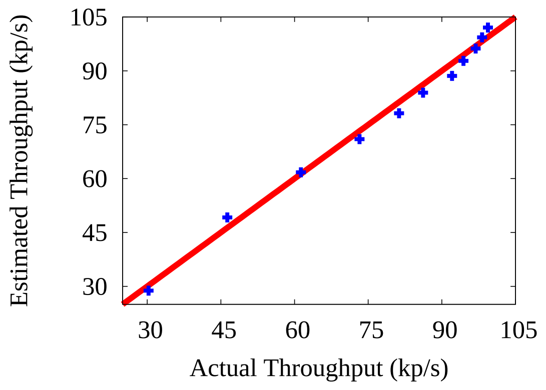


Figure 4: Residual Plot. This Figure shows the estimated throughput versus the actual values obtained in the experiments. In a perfect model the points lie exactly on the unit slope.

graphical representation called Residual Plot to evaluate the accuracy of our model. In this figure we plot the estimated value versus the actual values of experimental data. For reference, we also plot a unit slope to indicate a perfect model, where the estimated values are equal to the actual values. Figure 4 shows that the experimental points are very close to the unit slope, again suggesting a very good fit.

## 6. XTC Design

The main component of XTC is the Controller block. Therefore, the design of our system consists of choosing the Controller parameters according to the Xen System model. The Controller decides which value of cap will be assigned to a virtual router in order to achieve the Throughput Setpoint. We implement this block as a PI (Proportional Integral) controller, which is governed by the control law of Equation 2. In this equation, $u(k)$ is the controller output at the $k^{th}$ sample, that represents the $log(cap)$, and $e(k)$ is the error evaluated by the difference between the Throughput Setpoint and the Achieved Throughput.

$$u(k) = u(k-1) + (K_p + K_i)e(k) - K_p e(k-1). \tag{2}$$

The design of a PI controller consists of choosing the $K_p$ and $K_i$ parameters of Equation 2 that meet the system requirements, such as stability and small settling time. The former indicates that the system output, the Achieved Throughput in our case, converges to a stable value. The latter indicates if the system rapidly converges to the final output value.

We use the pole placement method to choose the $K_p$ and $K_i$ parameters, based on the values of $a$ and $b$ obtained in Section 5. Basically, we can represent a system with a transfer function that characterizes the relationship between its input and output. In XTC, the input is the Throughput Setpoint ($r(k)$) and the output is the Achieved Throughput ($y(k)$). The transfer function of the entire XTC loop is given by Equation 3. This function is written in the form of $Z$-transform, used to represent discrete systems. As an example, the output $y(k)$ becomes $Y(z)$ in Equation 3.

$$F(z) = \frac{Y(z)}{R(z)} = \frac{z(K_p + K_i)b - K_p b}{z^2 + z[(K_p + K_i)b - (a+1)] + (a - K_p b)}. \tag{3}$$

The values of $z$ that produce a zero in the numerator of Equation 3 are called zeros, whereas the $z$ values that produce a zero in the denominator are called poles. Poles and zeros impact system properties such as stability and settling time. The pole placement method considers only the influence of the poles in the system properties, ignoring the effect of zeros. This method gives the appropriate pole values in order to achieve the desired values of system properties. The values of $K_p$ and $K_i$ are evaluated thus by simple algebraic manipulations [17].

The controller parameters that lead the system to a small settling time and a stable behavior, according to the pole placement method, are $K_p = -1.747 \times 10^{-3}$ and $K_i = 11.311 \times 10^{-3}$. The XTC loop was simulated in Simulink from MATLAB. The results show that the theoretical system is stable and has a settling time of 10 seconds.

XTC also uses the concept of dead zone in which the system only actuates on the virtual router cap when the absolute value of the error $|e(k)|$ exceeds a given threshold. Thus, when $|e(k)|$ is lower than the defined threshold, the last configured cap is maintained. As XTC works by performing calls to Dom0, limiting the controller actions reduces the number of these calls. Also, if the XTC engine is executed in a remote machine, rather than on the physical router, the dead-zone technique reduces the control traffic. The threshold used in our XTC implementation is 10% of the Throughput Setpoint. The choice of this threshold is a trade-off between the number of control messages (smaller threshold) and the precision of the achieved throughput (higher threshold).

## 7. Experimental Results

We implement our proposed mechanism using the testbed of Figure 2. Packets are sent from TG to TR at a fixed rate using Iperf. Virtual routers hosted in TF forward data packets. The Traffic Controller (TC) measures the throughput achieved by the virtual router and plays the role of the Controller of Figure 1. To measure the achieved throughput for UDP flows, TC periodically collects the output of the Iperf Server executing in TR. For TCP flows, TC collects the output of the Iperf Client because, for TCP, the Iperf server only reports the throughput of the whole experiment. The congestion control algorithm used in the experiments is TCP CUBIC, the default in Linux. Measuring the TCP throughput in the Iperf Client is not a problem, since the throughput measured in the Iperf Client is the same of the Iperf Server due to TCP congestion control. We validate this statement by comparing all measurements with the final Iperf Server output. The comparison is omitted for sake of brevity. In a practical implementation, the throughput measurement and the XTC execution have to be performed in the machine running Xen (TF in our scenario). In our prototype, we split these functions into the TF and TC machines to guarantee that our results are independent of the implementation of the throughput meter running on TF, which might be overloaded by high packet rates. Section 8 describes the practical implementation of XTC and further experiments based on it.

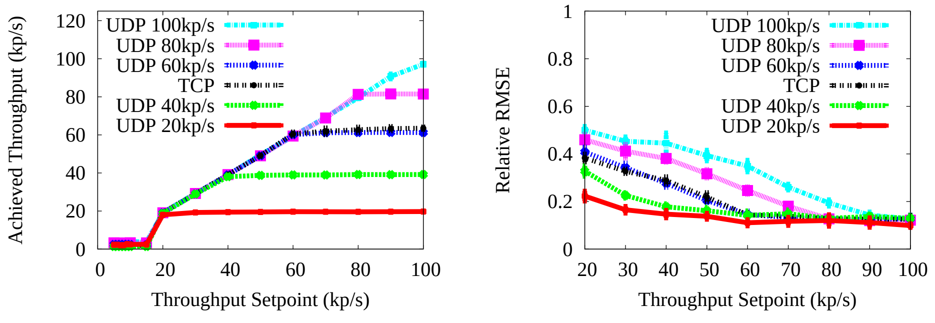
XTC running on the TC machine computes the cap of the virtual router based on Equation 2. In addition, it remotely adjusts the virtual router cap using Libvirt. Note that Equation 2 computes $log(cap)$, rather than $cap$, and thus the actuator must compute the inverse of $log(cap)$. The complexity of this operation was negligible in our experiments. Hence, XTC performs only simple operations, allowing the control of a large number of virtual routers. In the remainder of this section, we present results regarding the operation of XTC. All the following results are shown with a confidence interval of 95%.

## *7.1. System Validation*

Our first experiment consists of validating the XTC control system by applying different Throughput Setpoint values to the system input. This experiment uses either UDP or TCP flows. The latter allows the analysis of the interaction between XTC and another control mechanism, i.e. the congestion control of TCP.

The experiment consists of sending small packets (64 bytes for UDP and 524 bytes for TCP) from TG to TR during 120 seconds via a virtual router running on TF. XTC works by controlling the throughput of this virtual router according to the specified setpoint. For UDP, we vary the rate of the flow sent by TG. Figure 5(a) shows the achieved throughput according to the throughput setpoint for different flows. In this figure, we see that the default configuration of XTC responds well for setpoint values greater than 20kp/s. Nevertheless, XTC is not operational for a setpoint below this value. This is due to the linearization of the Xen System model (5.2), which causes the system to work well around the operating point. As the operating point is 78.14kp/s, values smaller than 20kp/s are too far and so the system cannot achieve the expected Throughput Setpoint. In Section 7.2, we apply additional techniques to expand the operating range of the system to circumvent this issue.

We also measure the amount of oscillation experienced by the system according to the setpoint assigned. The oscillation is quantified by the Relative RMSE (Root Mean Square Error), which is defined as the ratio between the RMSE and the Achieved Throughput. The RMSE itself quantifies the oscillatory behavior of the system showing how the system response deviates



(a) Achieved Throughput.

(b) Relative RMSE.

Figure 5: Validation of XTC for UDP and TCP flows. The metrics are measured by assigning different values of Throughput Setpoint to XTC.

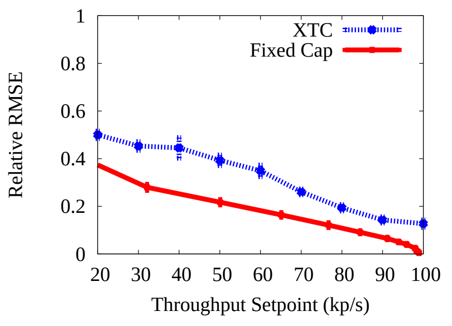
from the average throughput, in this case, the Achieved Throughput. Figure 5(b) shows the results of Relative RMSE as a function of the setpoint for each type of flow in the above experiment. We can see that the proposed throughput limitation has the drawback of adding oscillation to the system. As we show in the next results, however, the major part of this oscillation is due to the cap limitation in Xen. This means that the additional oscillation induced by XTC is small. Also, the oscillation increases with the packet dropping caused by the cap limitation. For example, with UDP flows, the oscillation increases when the setpoint for a given packet rate decreases, and more packets are dropped. For TCP flows, the packet loss can be viewed as the difference between the setpoint and the total capacity of the network, which is 60kp/s for TCP.

The next experiment compares XTC against the utilization of a static mapping function to correlate the cap value to the desired throughput, called Fixed Cap. To do this, we send a 64-byte UDP flow from TG to TR with 100kp/s and assign some fixed cap values, measuring the throughput achieved

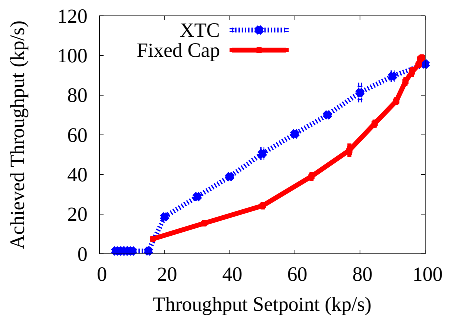
Table 1: Mapping used for the Fixed Cap.

| **Throughput Setpoint (kp/s)** | **Cap(%)** |
|---|---|
| 16 | 5 |
| 32 | 10 |
| 50 | 15 |
| 65 | 20 |
| 77 | 25 |
| 85 | 30 |
| 91 | 35 |
| 94 | 40 |
| 96 | 45 |
| 98 | 50 |

for each value. The mean values obtained for each fixed cap are considered as the Throughput Setpoint for the Fixed Cap experiment. This represents the same procedure that would be required to map a given fixed cap value to a desired Throughput Setpoint. Table 1 shows the mapping used in the Fixed Cap case. The measurements of interest are conducted the same way as done on the results with XTC. Note that, in this experiment, the Throughput Setpoint is always equal to the Achieved Throughput in the case of Fixed Cap, therefore we omit those results. Figure 6(a) shows the comparison between XTC and Fixed Cap in terms of the Relative RMSE. These results show that the major part of oscillation is caused by the limitation of throughput using cap and, as seen before, the oscillation is greater when the Throughput Setpoint is far from the packet rate sent by the source (100kp/s), even in the fixed cap case.



(a) Relative RMSE in normal condition.



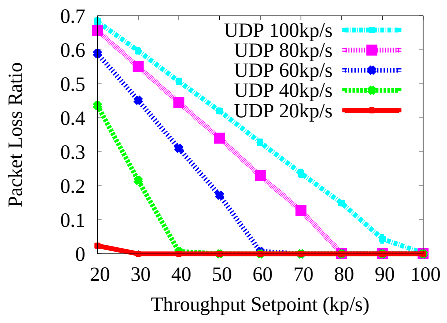
(b) Achieved Throughput in the presence of disturbance.

Figure 6: Comparison of Fixed Cap and XTC. The experiments quantify the oscillation added by XTC and the benefits of using a feedback control system instead of a simple mapping between cap and throughput.

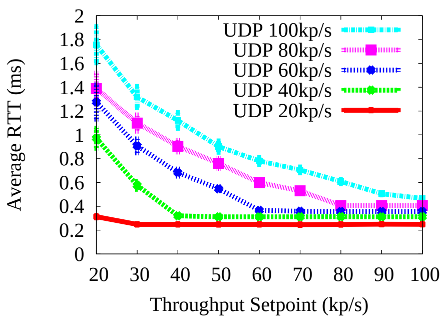
We also analyze the performance of XTC when the system is disturbed by an additional CPU-intensive task, such as the one needed to analyze IP packets with options set in the header. To simulate a disturbance, we perform the same experiment as before after starting a process in the virtual router. This process consumes 15% of the assigned CPU. As before, for the Fixed Cap case we use the mapping of Table 1. Obviously, for the Fixed Cap the Achieved Throughput will be smaller than the Throughput Setpoint because part of the CPU assigned is consumed by the disturbing process. Figure 6(b) shows the obtained results. They show that XTC rejects the disturbance introduced in the system, achieving the Throughput Setpoint. This experiment further justifies the utilization of a feedback controller instead of a simple mapping. As seen in Figure 6(a), Fixed Cap does not have the additional oscillations caused by the control loop. Nevertheless, Fixed Cap can not adapt to disturbances in the system.

To complete the system validation, we analyze other two performance metrics affected indirectly by XTC: RTT (Round-trip time) and packet loss. We perform the same experiment of Throughput Setpoint variation as before, while sending a `ping` probe. We thus measure the packet loss and the RTT experienced by the probe. The `ping` flow is started after 15 seconds of the experiment beginning and terminates 5 seconds before the experiment ends, and thus have a duration of 100 seconds. The rate used for `ping` is 0.1kp/s, not affecting the experiment. Figure 7 shows the obtained results for different UDP flow rates. Note that, as this initial XTC configuration works for a Throughput Setpoint greater than or equal to $20kp/s$, the results are shown from this value. Figure 7(a) illustrates the Packet Loss Ratio experienced by the probe. As XTC works by adjusting the *cap* of the virtual router (VR), the throughput limitation is done by inducing packet losses. Hence, every flow that passes through this VR will suffer packet losses, since XTC limits the aggregate throughput. The results show that the Packet Loss Ratio is proportional to the distance between Throughput Setpoint and the flow rate. As the Throughput Setpoint approaches the flow rate, the packet losses tend to zero because the XTC control is less effective.

Figure 7(b) shows the Average RTT. For each round, the Average RTT is evaluated by averaging the RTT of all `ping` packets sent during that round. As can be seen, the larger the difference between Throughput Setpoint and the flow rate, the longer is the RTT. In other words, when XTC is more effective the RTT is higher. As CPU capacity of the VR is reduced, the VR will serve more slowly packets from its network interface, thus increasing the processing delay.



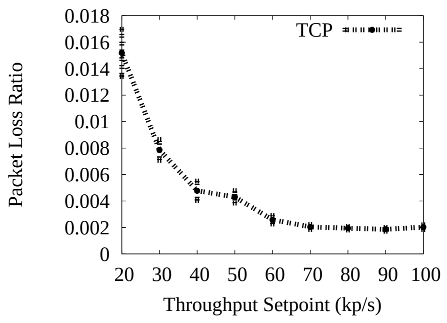
(a) Packet Loss Raio.



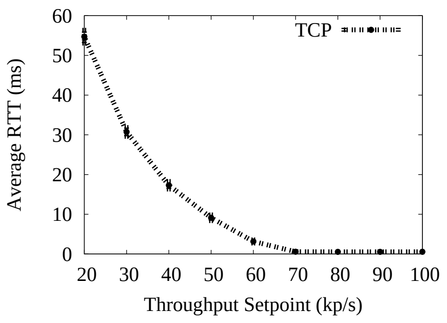
(b) Average RTT.

Figure 7: Performance metrics indirectly affected by XTC when controlling a VR with an UDP flow.

Figure 8 shows the Packet Loss Ratio and Average RTT when XTC controls a VR forwarding a TCP flow. The same remarks of the UDP case can be done for both performance metrics. Nevertheless, by looking at the y-axis of both Figure 8(a) and Figure 8(b), we can see that the `ping` probe suffers a very low packet loss comparing with UDP results. As TCP has congestion control, the flow rate is adjusted according to the available CPU capacity imposed by the *cap* control of XTC. Differently from UDP, which does not change its flow rate, the CPU capacity is enough to forward the flow. Consequently, the CPU capacity is not saturated and a probe with very low packet rate, as the `ping` sent at 0.1kp/s, will not suffer substantially loss. The Average RTT achieved with TCP is not comparable with the UDP results, since in the UDP experiment a lot of packets are discarded, meaning that the packets reach their deadline in the queues. In TCP this deadline is rarely achieved, as can be seen by the low Packet Loss Ratio.



(a) Packet Loss Ratio.



(b) Average RTT.

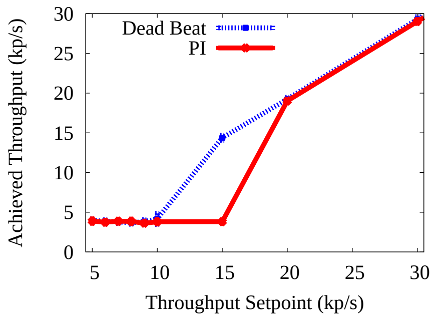
Figure 8: Performance metrics affected indirectly by XTC when controlling a VR with a TCP flow.

### *7.2. Operating Range Expansion*

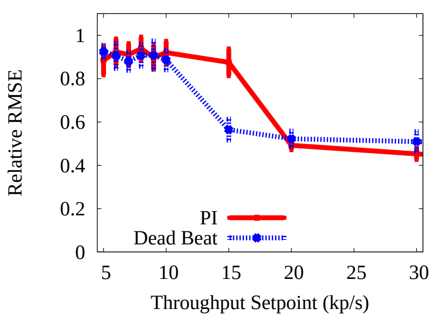
In order to expand the operating range of XTC, we propose two simple improvements, detailed below.

#### *7.2.1. Dead-beat Controller*

A dead-beat controller leads the system response to the minimum possible settling time. To achieve a better response of a linear system, all poles of the closed-loop transfer function must be in the origin of the $z$-plane. For instance, the closed-loop function of Equation 3, that characterizes the entire system controlled by XTC, must be $F(z) = \frac{1}{z}$ to have a dead-beat response. The position of the poles at the origin of the $z$-plane can help expanding the operating range of the system. This is because changes in the parameters $a$ and $b$ of the Xen System block may cause a displacement of the poles through the unit circle. If the poles are originally at the origin, it will be more difficult to perform any changes that could lead the poles to outside of the unit circle. This would cause system instability and would prevent



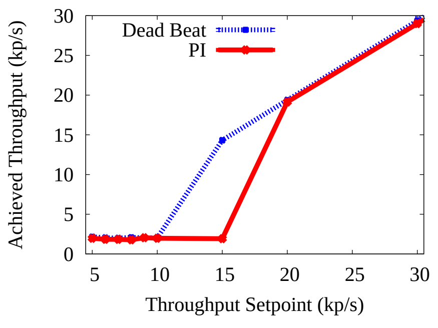
(a) Achieved Throughput.



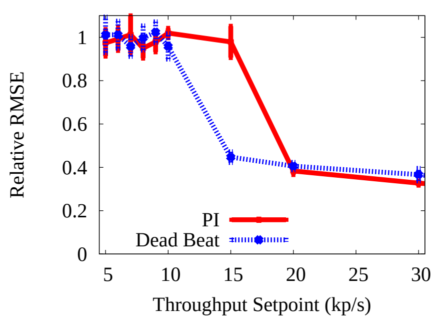
(b) Relative RMSE.

Figure 9: Comparison between PI and dead-beat controllers for an UDP flow. The experiment shows the operating range for each solution.

it from achieving the setpoint. After some algebraic manipulations we have found that, in order to have $F(z) = \frac{1}{z}$, the controller parameters must be $K_p = \frac{a}{b} = 1.452 \times 10^{-3}$ and $K_i = \frac{1-a}{b} = 14.419 \times 10^{-3}$. We evaluate the dead-beat controller by performing the same experiment of Section 7.1 for UDP and TCP flows, where we vary the Throughput Setpoint and measure the Achieved Throughput and Relative RMSE. In the UDP analysis, we only show the results for a 100kp/s flow because they present the worst oscillation, as shown in Figure 5(b). Figures 9 and 10 plot the results for UDP and TCP, respectively. We compare the results obtained with the dead-beat controller with the previous evaluated controller, called PI in these figures. The results show that with the utilization of the dead-beat controller, it is possible to decrease the minimum rate controlled from 20kp/s to 15kp/s for both UDP and TCP flows. The relative RMSE shows that the difference between the oscillations of the two controllers, considering the range where they both work, is negligible. Additionally, the dead-beat controller presents the same results as the PI controller for Throughput Setpoint values greater than the



(a) Achieved Throughput.



(b) Relative RMSE.

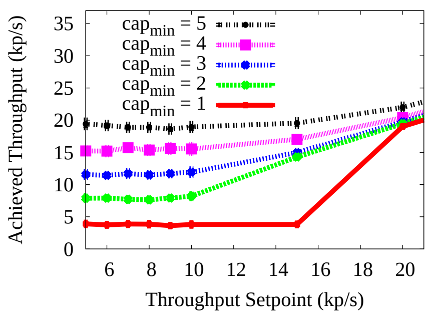
Figure 10: Comparison between PI and dead-beat controllers for a TCP flow.

ones shown in Figures 9 and 10.

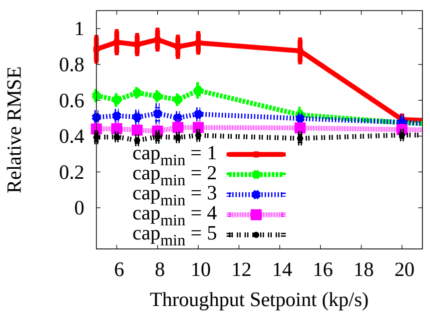
### *7.2.2. Minimum Cap*

Another approach to expand the operating range is to define a minimum cap value for the virtual router. In the experiments of Sections 7.1 and 7.2.1, the minimum cap was 1, which is the minimum cap configurable in Xen. Nevertheless, analyzing the evolution of the Achieved Throughput in the experiments of Section 7.1, we observe that for small setpoint values (i.e. values less than 20kp/s) the controller could not leave the initial state where $cap = 1$. Therefore, we investigate the assignment of a minimum value for cap to prevent the controller from being stuck in an undesired cap value. This value is hardware-dependent as well as the controller parameters $K_p$ and $K_i$. Nevertheless, it can be easily added to the initial training of the system used to define the controller parameters.

We perform the same experiment of Throughput Setpoint variation done before (Section 7.1), using the PI controller and different values of minimum cap, with an UDP flow at 100kp/s and a TCP flow. Figures 11 and 12



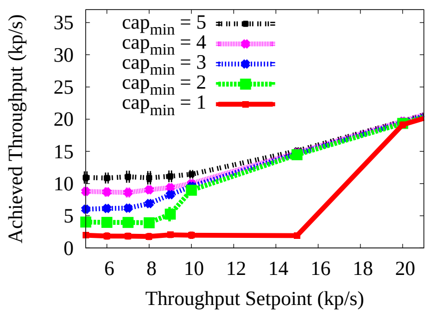
(a) Achieved Throughput.



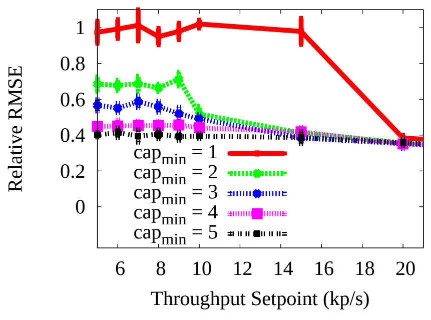
(b) Relative RMSE.

Figure 11: Comparison between different values of $cap_{min}$ for an UDP flow.

present the results for UDP and TCP, respectively. In these figures, each curve represents the results for a specific minimum cap. We call this minimum cap value $cap_{min}$. The results show that a minimum cap can be chosen in order to expand the operating range, but this value cannot be too high which would lead to an Achieved Throughput greater than the Throughput Setpoint. Analyzing the plots, we decide to use an intermediate value for the minimum cap ($cap_{min} = 3$), which linearizes the response of TCP flows according to the setpoint. This allows the limitation to approximately 6kp/s for TCP flows and improves the operating range of UDP flows up to 10kp/s. The Relative RMSE shows that the increasing of the minimum cap value does not add oscillation to the system, considering the point where the configuration with $cap_{min} = 1$ works (20kp/s). Additionally, independently of the minimum cap value, the performance remains the same for Throughput Setpoint values greater than the ones shown in Figures 11 and 12.



(a) Achieved Throughput.

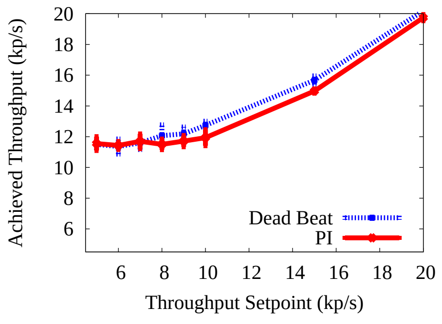


(b) Relative RMSE.

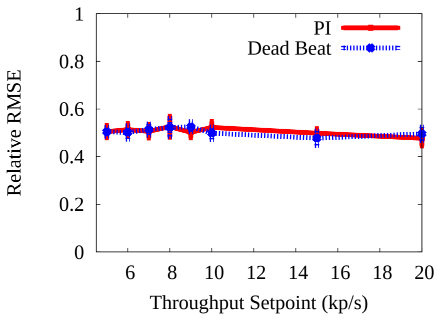
Figure 12: Comparison between different values of $cap_{min}$ for a TCP flow.

### *7.2.3. Dead-beat and minimum cap*

We combine a dead-beat controller with the chosen minimum cap value ($cap = 3$) to benefit from the approaches proposed in Sections 7.2.1 and 7.2.2. The same experiments are repeated to compare the results of this approach with a PI controller with minimum cap of 3. Figures 13 and 14 show the results for UDP and TCP flows, respectively. According to them, we conclude that the dead-beat controller for UDP flows has the same performance of the PI controller when the minimum cap is 3. Nevertheless, for TCP flows the dead-beat controller provides a response which is more linear in relation to the Throughput Setpoint. Although this improvement is apparently small, it can be representative when controlling a high number of virtual routers with small Throughput Setpoint values. Also, as commented before, the utilization of the dead-beat controller can make the system more robust to variations in Xen System block that were not foreseen. Thus, we use the dead- beat controller with minimum cap 3 in the remaining experiments.



(a) Achieved Throughput.

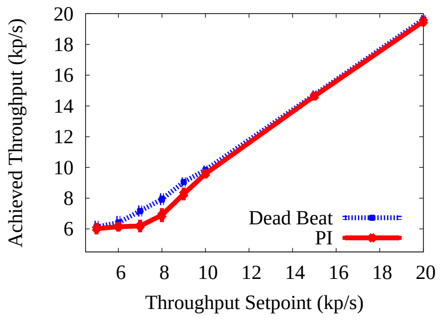


(b) Relative RMSE.

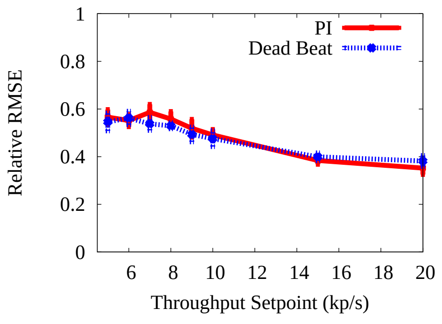
Figure 13: Comparison between the PI and the dead-beat controller with $cap_{min} = 3$ for an UDP flow.

## *7.3. Traffic Differentiation*

In this section, we show the ability of XTC to provide traffic differentiation between virtual routers. XTC can guarantee higher throughput to a virtual router by limiting the amount of resources used by the other ones. We conduct an experiment using six virtual routers, each forwarding a single TCP flow from TG to TR. First, we measure the throughput of each virtual router without XTC labeled "Without XTC" in Figure 15. In this figure, the $x$-axis represents each virtual router, whereas the $y$-axis represents the corresponding Achieved Throughput. We observe that the maximum throughput achieved without XTC in this scenario is 13kp/s and that the aggregate throughput is approximately 78kp/s, as shown in Table 2. This low aggregate throughput is a consequence of the Dom0 bottleneck. As an example of XTC differentiation, not possible without XTC, we conduct experiments to divide the aggregate throughput of 78kp/s, assigning less than 13kp/s to some virtual routers and more than 13kp/s to others. Hence, we aim at guaranteeing that the sum of the Throughput Setpoint of all virtual routers must



(a) Achieved Throughput.



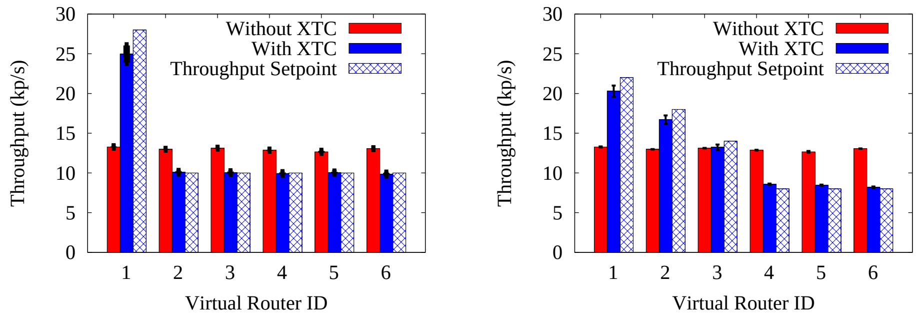
(b) Relative RMSE.

Figure 14: Comparison between the PI and the dead-beat controller with $cap_{min} = 3$ for a TCP flow.

be equal to the aggregated throughput (i.e. 78kp/s). Also, these values might be different between each other to show the traffic differentiation. Given that these requirements are respected, the Throughput Setpoint of each VR was chosen arbitrarily. We use two configuration examples to analyze the traffic differentiation:

- Configuration A: Throughput Setpoint at 10kp/s for five virtual routers and at 28kp/s for the remaining one.
- Configuration B: Throughput Setpoint at 22kp/s for one virtual router, at 18kp/s for another one, at 14kp/s for another router, and at 8kp/s for the remaining three routers.

Figure 15(a) shows the results of Configuration A, whereas Figure 15(b) shows the results of Configuration B. The bars labeled "With XTC" plot the Achieved Throughput measured when using XTC and the bars labeled as "Throughput Setpoint" are the Throughput Setpoint applied to each virtual router, indicated in the figure for reference. In both cases, we show that XTC can provide differentiation between virtual routers for the packet forwarding task, controlling the amount of processing power assigned to each router. Figure 15 shows that, for both Configurations A and B, the Achieved Throughput does not match exactly the Throughput Setpoint for some VRs because XTC presents a small loss in throughput when controlling parallel virtual routers. This loss affects more the VRs with higher Throughput Setpoint values, since these values are more difficult to achieve. The loss thus reduces the aggregate throughput when using XTC as compared when XTC is not used, as shown in Table 2. As this table shows, however, the maximum loss caused by XTC is approximately 5%, considering the extremes of the confidence interval.



(a) Configuration A. (b) Configuration B.

Figure 15: Traffic differentiation in a scenario with only TCP Flows. Different values of Throughput Setpoint are assigned to the virtual routers to show the ability of XTC to provide differentiation.

Figure 16 shows the Average RTT for each VR using the configurations A and B, and also when XTC is turned off. The RTT measurement is performed using the same methodology of Section 7.1. Note that, as pointed

Table 2: Aggregate Throughput with 6 Virtual Routers forwarding only TCP flows.

| **Configuration** | **Aggregate Throughput (kp/s)** |
|---|---|
| Without XTC | $77.97 \pm 0.07$ |
| A | $74.96 \pm 1.06$ |
| B | $75.49 \pm 0.85$ |

out before, XTC has the drawback of adding latency to the network. A smaller Throughput Setpoint causes a higher Average RTT, since XTC becomes more aggressive. However, by comparing the results of Figure 16 with those when one VR is being controlled (Figure 7(b)), we can conclude that the RTT does not increase substantially when we increase the number of VRs being controlled. For one VR, an aggressive limitation of 20kp/s produces an Average RTT of approximately 55ms, while for 6VRs the maximum Average RTT achieved was about 70ms. We do not show the results for packet losses since XTC experiences almost zero packet drops when controlling a router forwarding TCP flows. This behavior remains the same when parallel VRs are being controlled.

The next experiment shows the traffic differentiation when TCP and UDP flows coexist. We use a scenario similar to the previous one, but with two virtual routers. One router forwards a TCP flow while the other forwards an UDP flow at 100kp/s. The two flows send 524-byte packets. Figure 17 shows the Achieved Throughput of each router when XTC is off, labeled "Without XTC". As expected, the UDP flow has a higher throughput because it has no congestion control. The aggregate throughput in this experiment is approximately 100kp/s. We can use XTC to give more room to the virtual

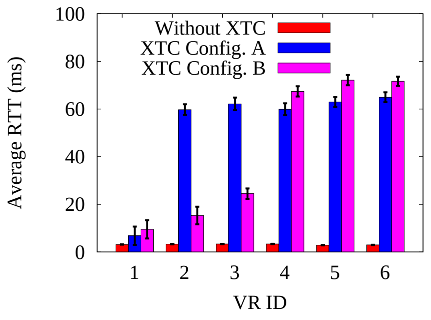


Figure 16: Average RTT for configurations A and B. As we reduce the Throughput Setpoint, XTC is more aggressive and thus the RTT is higher.

router forwarding the TCP flow by limiting the virtual router with the UDP flow. We thus assign a Throughput Setpoint of 20kp/s and of 80kp/s to the virtual router forwarding the UDP and TCP flows, respectively. Figure 17 shows the Achieved Throughput labeled "With XTC" and the Throughput Setpoint of 20kp/s and 80kp/s labeled "Throughput Setpoint" for reference. We note that XTC is able to allow higher throughput to the router forwarding the TCP flow, although it is not enough to achieve the desired setpoint. The consequence is the reduction of the aggregate throughput from approximately 100kp/s to 68kp/s. As seen in Figure 17, the throughput increase in the TCP flow (16kp/s) is less than the throughput decrease in the UDP flow (49kp/s). This is because, for an UDP flow, the amount of forwarding capacity actually freed is not equal to the throughput reduction caused by XTC. The problem occurs due to the absence of congestion control in UDP. Since XTC controls only the packet rate injected by a virtual router in Dom0, it does not change the sending rate at the source. Consequently, even if the forwarding throughput is limited to 20kp/s by XTC (Figure 17), the rate that Dom0 tries to send to the virtual router is still 100kp/s for the UDP flow. Therefore, Dom0 wastes resources by trying to send 100kp/s to the incoming interface of the virtual router even if the rate actually forwarded is 20kp/s. TCP does not waste resources, because the congestion control of TCP in the source adjusts the rate according to the limitation imposed by XTC.

We conclude that a lot of Dom0 resources are wasted when controlling virtual routers that forward flows without congestion control. This absence of congestion control might cause a collapse in the infrastructure if the resource allocation system (Section 3) ignores misbehaving virtual routers (i.e. routers that are forwarding UDP flows with high rates). Consequently, the resource allocation system, that comprises XTC and other mechanisms, should detect those routers. Afterward, it can make appropriate decisions according to the type of flow. These decisions could set a very low Throughput Setpoint on XTC for this router or temporarily activate per-interface controllers on its network interfaces. Those per-interface controllers can limit the rate that Dom0 sends to a virtual router [18]. However, the utilization of this type of controller reduces the flexibility of the virtual routers, and can be considered a punishment to them (Section 3). Also, more radical decisions could be made, as turning off the misbehaving virtual router.

## *7.4. Fairness*

In this section, we show an example of how XTC can provide fairness by controlling the aggregate throughput of a virtual router. We define fairness as the Min-Max Ratio, which is the ratio between the lowest and the highest throughput. Thus, when the Min-Max Ratio is higher, the throughput of the

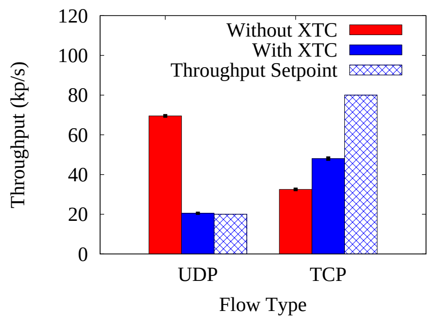
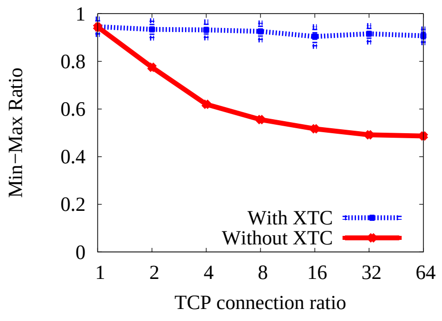
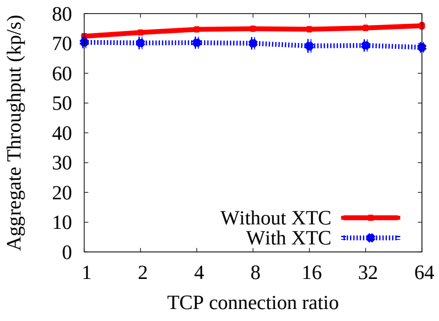


Figure 17: Traffic differentiation in a scenario where TCP and UDP flows coexist.

virtual routers tend to be more equal and so the distribution of Dom0 resources. In the next experiment we have two routers, one of them forwarding a different number of TCP flows between TG and TR. We consider a TCP flow as a single connection between source and destination. We fix a given virtual router with 1 TCP flow and for the other one we vary the number of flows. As TCP provides fairness between flows, we expect that each flow will receive an equal share of the total capacity. In this case, the aggregate throughput of the virtual router with more flows will be proportionally higher. Figure 18 shows the fairness comparison with and without XTC. Figure 18(a) shows the Min-Max Ratio as a function of the ratio between the number of flows forwarded by the two routers. Figure 18(b) shows the aggregate throughput with and without XTC. When using XTC, we equally divide the aggregate throughput achieved in the experiment without XTC between the routers. In Figure 18(a), without XTC, we observe an increasing unfairness between the virtual routers. This unfairness is not linear with the ratio of flows, probably because of the simple round-robin I/O scheduling used by Xen [3]. Therefore, the Xen scheduling cannot guarantee fairness.



(a) Min-Max Ratio.



(b) Aggregate Throughput.

Figure 18: Fairness Experiment. This experiment is done with different number of TCP flows in each virtual router to show the unfairness on Xen for this type of traffic and how XTC can reduce it.

Our results show that XTC can provide fairness since it controls the aggregate throughput of each virtual router, independently of the number of flows forwarded via their network interfaces. Figure 18(b) shows that the fairness increase comes at the cost of a small reduction in the aggregate throughput. This reduction, as seen in Section 7.3, is caused by the existence of multiple XTC instances.

## 8. Implementation

The above experiments used an experimental version of XTC that collects the throughput measurement based on the Iperf output. This method was employed to avoid any interference that a throughput meter implementation would cause. Also, as the experimental version uses the Iperf output of TR, it was only suitable to perform experiments with only one physical router. For a practical implementation, a throughput meter is needed at every XTC router. In this section we describe the practical XTC implementation, which is needed to perform experiments using more than one physical router, examining XTC scalability and overhead. This practical version of XTC can run in a production network.

## *8.1. Architecture*

Figure 19 depicts the architecture of XTC implementation. White boxes are XTC modules, gray boxes are components of Dom0, and the cross-hatched box is the Policing Mechanism. As mentioned in Section 3, the Policing Mechanism has a global view of the network and defines the Throughput Setpoint of each VR. The Policing Mechanism can request the activation or deactivation of XTC instances, etc. It is able to configure all physical routers and runs on the machine responsible for the system management. The XTC implementation is written in Python, all its modules run on Dom0 of the physical router, and have the following functions:

- **Throughput Meter:** This module periodically measures the throughput, in packets/second, of each virtual network interface by reading the `/proc/net/dev` file in Dom0.
- **Throughput Collector:** This module stores a map indicating which virtual network interfaces are assigned to each VR. Hence, it receives the throughput of each virtual network interface from the Throughput Meter and evaluates the aggregate throughput of each VR.
- **XTC Instance:** The XTC instance is the same XTC mechanism used in the above experiments, depicted in Figure 1. Each instance controls a specific VR, by periodically receiving its aggregate throughput (i.e. Achieved Throughput) and setting new cap values to the VR.

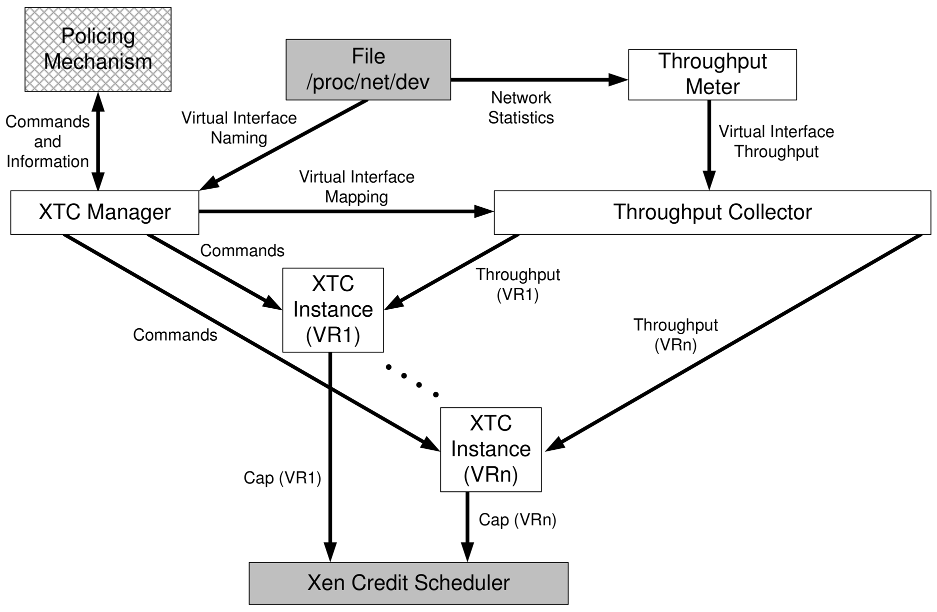


Figure 19: Architecture of the practical implementation of XTC in a single physical router.

- **XTC Manager:** This module communicates with the Policing Mechanism through a TCP socket. Based on the requests of the Policing Mechanism, the XTC Manager can activate XTC for a specified VR, can change its Throughput Setpoint on-the-fly and can also pause and deactivate an XTC Instance. Also, it returns to the Policing Mechanism information about the XTC instances, such as their status and current Throughput Setpoint. The XTC Manager keeps track of which virtual network interfaces are assigned to each VR and provides this information to the Throughput Collector. The mapping information is gathered by reading the `/proc/net/dev` file, since the name of every virtual network interface contains the ID of the virtual router it is assigned to.

Note that the XTC implementation access only Dom0 components, being transparent to the VRs.

## *8.2. Scalability analysis*

To provide a scalability analysis of XTC, we investigate its effects on added latency and packet loss as a data flow crosses multiple XTC routers in series. We add two physical routers to the testbed depicted in Figure 2. Hence, the testbed has a line topology in the following sequence TG-TF1-TF2-TF3-TR, where TF3 is the TF machine of Figure 2. The machines TF1 and TF2 are identical and equipped with an Intel Xeon X5570 2.93 GHz processor and Gigabit network interfaces. Since XTC is running on each physical node, the TC machine is not used. All machines run Debian Linux kernel version 3.2, and the physical routers (i.e. TF1, TF2 and TF3) run Xen version[3] 4.1. An XTC implementation is running on each physical router. The XTC parameters are the same used before for the traffic forwarder TF3. For the new physical routers TF1 and TF2, whose hardware is different from TF3, we evaluate their XTC parameters ($k_p$ and $k_i$) following the methodology of Section 5 and using a dead-beat controller. The Policing Mechanism is performed by the TG machine, but the only function implemented is to start XTC for each VR and set its Throughput Setpoint. The communication of the Policing Mechanism with the XTC Manager is done

[3]We have used in Section 7 the Xen version 3.4.2 to produce results comparable with our previous work. For the scalability experiment this requirement is not needed since the results are completely new and some machines are different compared to the previous work. We thus choose to use the latest stable Xen version (Xen 4.1) and kernel version (3.2) available for Debian at the time the experiments were conducted.

through a control network, which consists of a switch connected directly to each physical router. Consequently, the control traffic is separated from the experimental traffic.

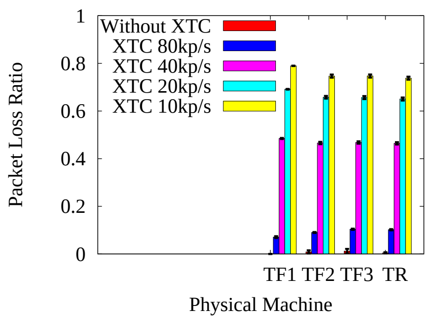
The goal of this analysis is to observe the behavior of XTC across multiple hops in the network in terms of packet loss and latency increasing. The experiment consists of sending an UDP flow from TG to TR, that is forwarded by virtual routers running on TF1, TF2 and TF3. Consequently, the virtual topology is also a line with one VR in each physical router. While sending the flow, we also send a ping probe to a given node of the network using the same methodology of Section 7.1. The experiments are done by probing a given VR or the TR. When probing each node, we use XTC with a Throughput Setpoint value assigned to all VRs, or with XTC turned off. During each experiment, all VRs are configured with the same Throughput Setpoint because it is useless to configure them with different values, since only the XTC with the lowest value would be effective. When all the instances have equal values, the XTC in TF1 will be the most effective since this machine is the first one that the flow crosses. However, the other instances will also be useful due to the XTC oscillation: when TF1 limits the throughput in a value higher than the Throughput Setpoint, the other machines can still limit at the specified value.

Figure 20 shows the scalability results for a 90kp/s UDP flow. This value was chosen since the maximum UDP throughput achieved in this testbed was close to 90kp/s. The VRs achieve the Throughput Setpoint in all experiments. For each probed VR, the x-axis specifies in which physical router it resides. Also, we show results for the TG machine, which represents the last hop traversed by the data flow. The experiments were performed with XTC configured in various Throughput Setpoint values. Note that, as we analyze the performance hop by hop, the values of Packet Loss Ratio and Average RTT of a given node are cumulative with the previous nodes (e.g. the losses achieved in TR accounts for the losses of TF1, TF2 and TF3). The only exception is the Packet Loss of TF1. Since this machine is the first one that the flow crosses, it wastes a lot of resources sending to the VR packets that will be discarded by XTC. The other VRs will thus receive packets in a rate close to the Throughput Setpoint. Consequently, TF1 will lose more packets than the others when responding to ping probes.

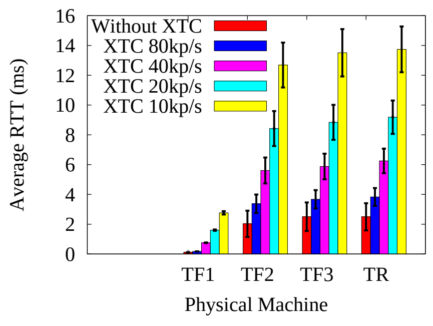
Both the results of Figure 20 show that the Packet Loss Ratio and the Average RTT increase as we decrease the Throughput Setpoint, as already observed in Section 7.1. Figure 20(a) shows that the Packet Loss Ratio is constant across the network, and thus the number of hops executing XTC does not reduce the performance concerning the Packet Loss Ratio. Figure 20(b) shows the results for the Average RTT. For all XTC values, we can observe a high increase in RTT from the first router to the second. However, it does not change significantly after that and settles around 14ms. Hence, XTC does not add latency as we increase the number of nodes in the network.

## *8.3. Overhead discussion*

As observed on the above results, XTC introduces overhead in terms of RTT. Nevertheless, this overhead does not represent a scalability issue because RTT converges to a constant value as we increase the number of physical routers. In addition, XTC produces packet drops when a router is forwarding packets at a rate higher than the maximum permitted.



(a) Packet Loss Ratio.



(b) Average RTT.

Figure 20: Scalability Experiment with a 90kp/s UDP flow.

Regarding the XTC implementation, all blocks are based on simple operations. Hence, they do not add significant performance overhead. The control loop of XTC, for instance, executes every second, demanding negligible CPU processing. The exception is the Throughput Meter, which periodically gather statistics from the `/proc/net/dev` file in Domain 0. These statistics are collected every 0.1 second further adding CPU processing overhead. This was not an issue for our experiments though. Even using commodity PCs (Section 4), our XTC implementation works fine with only two CPU cores allocated for Domain 0: one for packet forwarding and another for all XTC and kernel tasks. It is worth mentioning that allocating two cores for Domain 0 is not an issue since nowadays even commodity PCs are multi-core.

## 9. Related Work

Resource control of virtual networks can be done globally or locally. Global control is responsible for creating virtual links and defining the location of each virtual router in the network, providing functions such as virtual to physical network mapping and virtual router migration between physical routers. On the other hand, local control provides resource allocation between virtual routers within a physical node. Hence, global control decisions assume that a local control is active within each physical router. XTC is a local controller used to share Dom0 resources.

In the context of global control, a lot of effort has been given to the virtual to physical network mapping, also known as virtual network embedding. This mapping is needed to maximize the number of virtual networks supported in the physical network, considering infrastructure constraints such as bandwidth, memory, and CPU. Network embedding is considered as an optimization problem that generally is NP-hard and thus is tackled by approximate algorithms in the literature. For example, in [19] an algorithm is proposed to solve a multi-objective mixed-integer optimization problem in the context of virtual network mapping. Beyond the most classical constraints listed above, some works add other constraints in order to provide a better network embedding. In [20] the authors consider that, in order to instantiate a virtual router, a disk image must be transferred from a server to a physical router. As a consequence, they consider the location of this server in the network and choose the mapping in order to optimize also the time to transfer the disk image. The mapping can also consider application-specific requirements. For example, in [21] the authors consider the network embedding for multicast video.

Approaches that provide local control are scarce in the context of virtual networks, but well investigated in the context of resource allocation in servers [22] [23]. Anwer *et al.* [18] address the network bottleneck problem of Xen and propose a per-interface controller. They use a controller built on a NetFPGA [24] to provide fairness between virtual machines. This controller is used to drop packets on each network interface before they reach the hypervisor. If a virtual machine is receiving packets at a rate higher than permitted, the control mechanism blocks the extra amount of packets, ensuring that they do not arrive at the hypervisor. As a consequence, this control can avoid wasting Dom0 resources. The tradeoff, however, is the requirement for specific hardware, the NetFPGA. Also, as a per-interface controller, it reduces flexibility for the virtual network administrator, as discussed in Section 1. Nevertheless, this controller can operate in parallel with a software-based mechanism, such as XTC, in order to avoid problems with flows that do not have a congestion control at the source, as seen in Section 7.3. Fernandes and Duarte [25] propose a local resource allocation system for Xen to manage Dom0 resources. This mechanism monitors resource utilization and punishes misbehaving virtual routers. The punishment is done by discarding the packets that have the misbehaving router as source or destination. Differently from our work they only consider the plane separation scenario. As explained in Section 1 we choose not to use this scenario in order to address the extreme case of Dom0 bottleneck.

## 10. Conclusion

Network virtualization is the basis for a future Internet pluralist architecture. When using Xen, we have a network bottleneck in Dom0, which represents a key challenge for the utilization of virtualized routers. Our results show that the traffic forwarded by a virtual router can interfere with another one using Xen's default implementation. To minimize this problem, we have proposed a traffic control mechanism (XTC) to orchestrate the amount of Dom0 resources used by each virtual router. The main idea of XTC is to perform throughput control by controlling the amount of CPU given to each virtual router. The adjusted throughput is important to allow control per virtual router, differently from classical approaches that only provide per interface control. Our experimental results show that XTC provides differentiation between virtual routers and adapts to system changes. Also, we demonstrate that XTC can provide fairness between virtual routers, which is not possible using Xen only. Finally, we show that, although XTC adds latency and packet loss to the network, these metrics are not increased substantially when we increase the number of hops running XTC.

Our future work includes building a policing mechanism that will define the Throughput Setpoint of each XTC instance. The setpoint must be configured according to an SLA (Service Level Agreement) defined by the administrator of each virtual network. Also, the Throughput Setpoint might be dynamically adjusted, according to the resource utilization at a given period and to the virtual router behavior. Finally, the policing mechanism must be able to detect misbehaving routers and make decisions in order to punish them, such as reducing its Throughput Setpoint, enabling per-interface controllers, or even turning them off.

## Appendix A. System Linearization

The linearization applied in this work consists in choosing an operating point and approximating the nonlinear function as linear around this point.

Considering a non-linar system which the output $\tilde{y}(k+1)$ depends on the input and output at the $k^{th}$ sample, given respectively by $\tilde{u}(k)$ and $\tilde{y}(k)$. Hence, the function can be represented as:

$$\tilde{y}(k+1) = f(\tilde{y}(k), \tilde{u}(k)). \tag{A.1}$$

To perform the linearization, we choose and operating point $\bar{y} = f(\bar{y}, \bar{u})$, in which the input and output are constant. For example, in this work the $\bar{y}$ is the mean value of the Achieved Throughput over the region of interest and $\bar{u}$ is the input value that leads to $\bar{y}$. We thus model the signals of the linearized system as offset values from the operating point. Formally, the input and output of the linearized system are described respectively as:

$$u(k) = \tilde{u}(k) - \bar{u} \tag{A.2}$$

$$y(k) = \tilde{y}(k) - \bar{y} \tag{A.3}$$

We thus rewrite Equation A.1, replacing $\tilde{u}(k)$ and $\tilde{y}(k)$ using Equations A.2 and A.3:

$$\bar{y} + y(k+1) = f(\bar{y} + y(k), \bar{u} + u(k)). \tag{A.4}$$

To linearize the system, we apply the Taylor series expansion in function $f$ resulting in:

$$\bar{y} + y(k+1) = f(\bar{y}, \bar{u}) + \left(\frac{\partial f}{\partial y}\right)_{\bar{y},\bar{u}} y(k) + \left(\frac{\partial f}{\partial u}\right)_{\bar{y},\bar{u}} u(k) + ... \tag{A.5}$$

Using the fact that $\bar{y} = f(\bar{y}, \bar{u})$, we cancel the terms $\bar{y}$ and $f(\bar{y}, \bar{u})$ in Equation A.5. Also, using this equality, the partial derivatives in Equation A.5 became constant values. We can thus write $\left(\frac{\partial f}{\partial y}\right)_{\bar{y},\bar{u}} y(k) = a \times y(k)$ and

$\left(\frac{\partial f}{\partial u}\right)_{\bar{y},\bar{u}} u(k) = b \times u(k)$. Ignoring the other terms of the Taylor expansion, the linearized system is thus given by:

$$y(k+1) = a \times y(k) + b \times u(k). \tag{A.6}$$

Finally, Equation A.6 is a linear approximation of Equation A.1. We consider that this approximation is valid for values close to the operating point $f(\bar{y}, \bar{u})$. However, in this work, the nonlinear system is almost linear as observed in Section 5.2. Hence, the approximation is valid even for values far from the operating point as seen in Figure 4 of that section. More details about the linearization method applied in this work can be found in [17].

## Acknowledgments

The authors would like to thank CAPES, CNPq, CTIC, FAPERJ, and FINEP for their financial support to this work. We thank also Eduardo V. L. Nunes, Marcelo D. de Amorim and Victor T. da Costa for their valuable help.

## References

[1] Chowdhury, N., Boutaba, R.. A survey of network virtualization. Computer Networks 2010;54(5):862–876.

[2] Carapinha, J., Jiménez, J.. Network virtualization: a view from the bottom. In: Proceedings of the 1st ACM workshop on Virtualized Infrastructure Systems and Architectures (VISA 2009). 2009, p. 73–80.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S.,, , Harris, T., et al. Xen and the art of virtualization. In: ACM Symposium on Operating Systems Principles (SOSP 2003). 2003, p. 164–177.

[4] Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.. kvm: the linux virtual machine monitor. In: Linux Symposium; vol. 1. 2007, p. 225–230.

[5] VMware ESXi - accessed in february 2013. http://www.vmware.com/products/vi/esx/; 2013.

[6] OpenVZ user's guide. http://download.openvz.org/doc/OpenVZ-Users-Guide.pdf; 2013.

[7] Lantz, B., Heller, B., McKeown, N.. A network in a laptop: rapid prototyping for software-defined networks. In: ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets 2010). 2010, p. 19:1–19:6.

[8] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., et al. Openflow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review 2008;38(2):69–74.

[9] Sherwood, R., Gibb, G., Yap, K., Appenzeller, G., Casado, M., McKeown, N., et al. Flowvisor: A network virtualization layer. Tech. Rep.; Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium; 2009.

[10] Fernandes, N.C., Moreira, M.D.D., Moraes, I.M., Ferraz, L.H.G., Couto, R.S., Carvalho, H.E.T., et al. Virtual networks: Isolation,

performance, and trends. Annals of Telecommunications 2010;66:339–355.

[11] Wang, Y., Keller, E., Biskeborn, B., van der Merwe, J., Rexford, J.. Virtual routers on the move: live router migration as a network-management primitive. In: ACM SIGCOMM Conference on Data Communication (SIGCOMM 2008). 2008, p. 231–242.

[12] Couto, R.S., Campista, M.E.M., Costa, L.H.M.K.. XTC: A Throughput Control Mechanism for Xen-based Virtualized Software Routers. In: IEEE Global Telecommunications Conference (GLOBECOM 2011). 2011, p. 1–6.

[13] Ongaro, D., Cox, A., Rixner, S.. Scheduling I/O in virtual machine monitors. In: ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2008). 2008, p. 1–10.

[14] Libvirt: The virtualization API - accessed in february 2013. http://libvirt.org/; 2013.

[15] Almesberger, W., et al. Linux network traffic control - Implementation overview. White Paper available in http://diffserv.sourceforge.net/; 2001.

[16] Wang, Z., Zhu, X., Singhal, S.. Utilization and SLO-based control for dynamic sizing of resource partitions. Ambient Networks 2005;3775(1):133–144.

[17] Hellerstein, J., Diao, Y., Parekh, S., Tilbury, D.. Feedback Control

of Computing Systems. WILEY-INTERSCIENCE; John Wiley & Sons; first ed.; 2004.

[18] Anwer, M., Nayak, A., Feamster, N., Liu, L.. Network I/O fairness in virtual machines. In: ACM SIGCOMM workshop on Virtualized Infrastructure Systems and Architectures (VISA 2010). ACM; 2010, p. 73–80.

[19] Fajjari, I., Ait Saadi, N., Pujolle, G., Zimmermann, H.. Adaptive-VNE: A flexible resource allocation for virtual network embedding algorithm. In: IEEE Global Telecommunications Conference (GLOBECOM 2012). 2012, p. 2664–2670.

[20] Alkmim, G., Batista, D., da Fonseca, N.. Mapping virtual networks onto substrate networks. Journal of Internet Services and Applications 2013;4(1):3. URL `http://www.jisajournal.com/content/4/1/3`.

[21] Miao, Y., Yang, Q., Wu, C., Jiang, M., Chen, J.. Multicast virtual network mapping for supporting multiple description coding-based video applications. Computer Networks 2013;57(4):990–1002. doi: http://dx.doi.org/10.1016/j.comnet.2012.11.013.

[22] Kjaer, M., Kihl, M., Robertsson, A.. Resource allocation and disturbance rejection in web servers using SLAs and virtualized servers. IEEE Transactions on Network and Service Management 2010;6(4):226–239.

[23] Padala, P., Shin, K., Zhu, X., Uysal, M., Wang, Z., Singhal, S., et al. Adaptive control of virtualized resources in utility computing environments. ACM SIGOPS Operating Systems Review 2007;41(3):289–302.

[24] Lockwood, J., McKeown, N., Watson, G., Gibb, G., Hartke, P., Naous, J., et al. NetFPGA–an open platform for gigabit-rate network switching and routing. In: IEEE International Conference on Microelectronic Systems Education (MSE 2007). 2007, p. 160–161.

[25] Fernandes, N.C., Duarte, O.C.M.B.. XNetMon: A network monitor for securing virtual networks. In: IEEE International Conference on Communications (ICC 2011). 2011, p. 1–5.