Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 97 (2016) 4 – 13

Procedia Computer Science

CLOUD FORWARD: From Distributed to Complete Computing, CF2016, 18-20 October 2016, Madrid, Spain

QoS Guarantees for Network Bandwidth in Private Clouds

Gaetano F. Anastasi[a], Massimo Coppola[a,*], Patrizio Dazzi[a,**], Marco Distefano[a,b]

[a]*Information Science and Technologies Institute, National Research Council (CNR), Via G. Moruzzi 1, 56124 Pisa, Italy*
[b]*Dept. of Computer Science, University of Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy*

Abstract

In modern datacenters, variability of network load and performance tipically leads to unpredictable application performance for the many tenants sharing the hardware. To overcome this problem, Cloud providers should supply network guarantees as part of their Quality of Service (QoS) offer to the customers. This paper focuses on providing QoS guarantees for the network bandwidth. Starting form a Service Level Agreement (SLA) specification we model an admission control test and adopt an endpoint-only solution for the provider, enforcing bandwidth guarantees through the Linux Traffic Control (TC) technology. Experiments are reported to show the effectiveness of Linux TC in the proposed approach.

Peer-review under responsibility of organizing committee of the international conference on cloud forward: From Distributed to Complete Computing

Keywords: Quality of Service; Network guarantees; SLA; Traffic Control; Cloud Computing

1. Introduction

Cloud Computing is nowadays one of the most popular computational paradigms, the primary goal of which consists in providing the common three main layer of service provisioning: the Infrastructure-as-a-Service (IaaS), the Platform as a Service (PaaS) and the Software as a Service (SaaS). The ultimate definition of Everything-as-a-Service (XaaS)[1] coined for Cloud Computing emphasizes the primary role of services as the foundation of this computational paradigm[2]. Cloud Computing implies adopting the paradigm of Service Oriented Architecture (SOA), the design methodology that relies on the *publish/find/bind* pattern, where services are the smallest software bricks of distributed applications. Services must obey to certain well-known interrelated design principles[3]. They must be highly-abstracted and loosely coupled, as well as flexible and autonomous, in order for them to be easily composable, reusable, and for allowing creation of added value and distributed computation from existing parts.

As customers rely on Cloud providers that supply their computing needs to meet production objectives[4], services cannot be solely delivered on a best-effort basis. Instead, Cloud providers must support specific Quality of Service

* Corresponding author. Tel.: +39-050-621-2992 ; fax: +39-050-315-2040.
** Corresponding author. Tel.: +39-050-621-3074 ; fax: +39-050-315-2040.
E-mail address: massimo.coppola@isti.cnr.it ; patrizio.dazzi@isti.cnr.it

Peer-review under responsibility of organizing committee of the international conference on cloud forward: From Distributed to Complete Computing
doi:10.1016/j.procs.2016.08.275

(QoS)[5,6] requirements coming from customers, and enforce them in spite of unforeseen events, notably including unwanted interaction between different customers' applications caused by contention for hardware resources. Cloud providers need to define and provide measurable QoS metrics and corresponding QoS guarantees to their customer, that are negotiated per application and consolidated into Service Level Agreements (SLAs)[7]. To satisfy the QoS requirements of all active SLAs, Cloud providers have to address two related aspects of the problem: on the one hand they have to promote QoS-based resource allocation mechanisms differentiating service requests based on their utility and on the other hand they must provide resource management mechanisms that can actually enforce the QoS guarantees offered to customers.

QoS guarantees related to time-sensitive applications, that may be inherently (soft) real-time, are especially important. As an example, in the case of core services, the deadline miss of a single service typically affects the overall system performance. Other services, like retrieving components' status or log information, may have more relaxed timing constraints. For these reasons, it becomes of paramount importance to provide a flexible and robust infrastructure for supporting QoS at all levels in the system. The underlying infrastructure must be flexible, allowing dynamic configuration of the QoS parameters, as well as be robust and tolerant to faulty components, so that if a service starts to misbehave it won't affect the QoS of other services concurrently delivered.

Our work focuses on the problem of guaranteeing a certain network bandwidth to services delivered by generic Service Oriented Infrastructures (SOIs), including Cloud services. In particular, this work addresses the problem of providing mutually-safe network QoS guarantees to distinct services by leveraging Linux TC (Linux Traffic Control)[8], a technology that has a wide range of applicability and is available in common Linux distributions. The main contribution of this paper includes:

- experimental evaluation of the performance of Linux TC in enforcing network bandwidth allocations, for supporting QoS guarantees offered by providers through SLAs during service provisioning;
- definition of quantifiable SLA parameters for supporting the allocation of different network bandwidth shares among different tenants;
- design and development of a resource management technique, based on admission control, for allocating network bandwidth shares to tenants and allowing to respect the QoS guarantees expressed in the SLA.

In the remainder of the paper, Section 2 discusses a QoS architecture that can constitute a possible framework for including the capability of providing network bandwidth guarantees. Section 3 focuses on describing the proposed approach for guaranteeing a certain network bandwidth during service provisioning. In Section 4 some experiments are performed for evaluating the effectiveness of the proposed approach. The related work is analyzed in Section 5 and Section 6 draws conclusions.

2. Reference Architecture

For our work we consider a QoS architecture[9] well-suited for providing QoS guarantees, including the network bandwidth ones we focus on. Such architecture is introduced to ease the presentation of this work, however we believe the proposed approach can be easily adopted in other architectures. At a glance, such architecture enables applications to negotiate QoS parameters related to services and it also guarantees such parameters during service provisioning. The architecture, as depicted in Figure 1, is organized on two different layers: the *QoS Negotiation Layer*, in which the QoS parameters are negotiated, and the *QoS Provisioning Layer*, which takes care of providing services while respecting negotiated guarantees.

In the QoS Negotiation Layer, SLAs are negotiated according to the WS-Agreement model[7], but in principle any other SLA framework can be used. Agreements are generated by interactions between providers and customers, in which the *Agreement Templates* are used by customers to generate proposals. The proposal, containing the desired QoS parameters, is inspected by the provider, which decides whether to accept or reject it. If the proposal is accepted an Agreement is created and notified to the requester. Otherwise, the Agreement is rejected and the service is not started. This layer performs admission control (i.e. the test of Eq. 2 as described in Section 3.2) to verify if the QoS requested can be guaranteed. In such case, the resources are reserved for the corresponding services.

The QoS Provisioning Layer actually provides services characterized by the QoS guarantees defined in the negotiation phase. The *Cloud Manager* component is responsible of receiving and handling service requests. In a Cloud, this

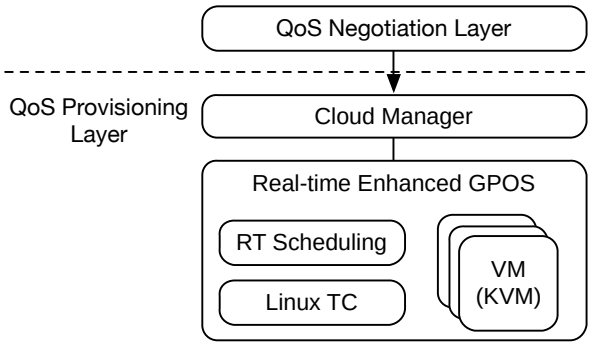

Fig. 1. Reference QoS Architecture.

```
<wsag:ServiceDescriptionTerm
        wsag:Name="server_parameters"
        wsag:ServiceName="httpd">
  <qos:ServerParams xmlns:qos="schemas.webqos">
      <qos:NetMinBandwidth
        unit="Mbps"> 19 </qos:NetMinBandwidth>
      <qos:NetMaxBandwidth
        unit="Mbps"> 19 </qos:NetMaxBandwidth>
  </qos:ServerParams>
</wsag:ServiceDescriptionTerm>
```

Fig. 2. XML Agreement fragment for negotiating bandwidth allocations.

role can be covered by an IaaS Cloud manager as OpenNebula[10]. Services are provided by means of a general purpose Operating System (GPOS) enhanced with real-time features to enforce QoS guarantees. Moreover, such OS must also provide virtualization capabilities for allowing customers to run their own VMs. Obviously, the virtualization layer adds complexity to the QoS management. Our previous work[11,12] shows that real-time scheduling techniques applied to the Linux scheduler allow to cope with some of the overhead and unpredictabilities experienced when executing multiple VMs on the same host. In particular, our approach is well suited for Type-2 hypervisors (e.g., KVM[13]) as they run on top of an unmodified host OS and thus existing tools and kernel modules can be reused without any modification.

3. Approach

The negotiation and the management of network QoS parameters are made of a few steps. Each client can negotiate certain network QoS parameters by sending contract proposal, as described in Section 3.1. The provider evaluates such parameters and performs the admission control test formalized in Section 3.2 to decide whether to accept or reject the proposal. Finally, the enforcement of the negotiated network parameters is detailed in Section 3.3.

3.1. SLA Specification

Customers can negotiate the network bandwidth QoS parameters by exploiting the agreement templates defined by providers. The QoS parameters defined in this work are `NetMinBandwidth` and `NetMaxBandwidth`.

- `NetMinBandwidth`. This parameter represents the minimum guaranteed network bandwidth allocated to a client. This is the bandwidth that has to be always guaranteed for the communications involving the provider and the customer.
- `NetMaxBandwidth`. This parameter represents the maximum network bandwidth that can be allocated to a client. If the provider has some unallocated bandwidth capacity, it could be assigned to a client up to reach this limit in the total amount of the reservation.

Customers can specify their required network QoS level by storing the corresponding parameters in an agreement proposal. An example can be found in the XML fragment of Figure 2, in which the client requests a guaranteed network bandwidth of $19Mbps$. We will use $1Mb$ for meaning $10^6 bit$ and $1Mib$ (mebibit) for meaning $2^{20} bit$ and thus reflecting the typical storage unit convention.

3.2. Admission Control

The provider performs admission control to decide whether to accept or reject a request of network bandwidth allocation. Such test can be formalized by introducing the following notation. Let B^{eth} be the maximum bandwidth of the provider network interface(s). The provider will be configured to assign a fraction of B^{eth} to the traffic directed to customers using the QoS framework. We will use the term *reserved network bandwidth utilization* (denoted by U^{res}, with $U^{res} \leq 1$) to represent such a value. Instead, the *unreserved bandwidth utilization* is denoted by $U^{oth} = (1 - U^{res})$. Denoting $C = \{c_1, c_2, \ldots c_n\}$ as the set of customers that have successful negotiated a network bandwidth reservation, the provider keeps in memory the pairs $\{U_c^{min}, U_c^{max}\}$ with $1 \leq c \leq n$. Instead, the network bandwidth utilization currently allocated to the customer c is denoted with U_c. Such a value is computed according to the policy of the provider. Eq. 1 shows an assignment that fairly distributes the spare bandwidth capacity.

$$U_c = min\left\{\left(U_c^{min} + \frac{U^{res} - \sum_{i=1}^{n} U_i^{min}}{n}\right), U_c^{max}\right\} \tag{1}$$

The minimum network bandwidth is requested through the parameter `NetMinBandwidth`, denoted by B_r, thus $U_r = B_r / B^{eth}$ indicates the minimum network bandwidth utilization that has been requested. The maximum network bandwidth is requested through the parameter `NetMaxBandwidth`, denoted by B_r^{max}. Thus, the maximum network bandwidth utilization requested is denoted by $U_r^{max} = B_r^{max} / B^{eth}$. A request can be accepted iff the admission test of Eq. 2 and the preliminary constraints $U_r \leq U_r^{max} \leq U^{res}$ hold.

$$\sum_{c \in C} U_c^{min} + U_r \leq U^{res} \tag{2}$$

3.3. Bandwidth Enforcement

The allocation of network bandwidth can be managed through the consolidated Linux kernel module called Linux TC. Such a module provides some mechanisms for rearranging traffic flows and scheduling packet transmissions, by means of *queuing disciplines*, *classes* and *filters*. A queuing discipline (qdisc) is a scheduler that rearranges packet queues and can contain classes. If such classes are terminal ones (leaf classes), they must contain also a qdisc responsible to send data from that classes. A filter is an object that permits to classify packets in output queues. For the purpose of our work, we will focus on the Hierarchy Token Bucket[14] (HTB) qdisc. It requires to configure each class with two parameters: *rate*, that is maximum rate a class and all its children are guaranteed; *ceil*, that is the maximum rate at which a class can send, if its parent has bandwidth to spare (default value is equal to *rate*).

In the proposed approach, HTB is used both in the root qdisc and in the internal classes. In particular, a parent class is created with parameters $rate = ceil = B^{eth}$, allowing children classes to borrow spare bandwidth capacity from it. Then, a leaf class is created and associated to each customer that has successfully negotiated a bandwidth reservation. Such class is configured using the negotiated parameters, by specifying $rate = B_n$ and $ceil = B_n^{max}$. The remaining traffic is managed by an additional leaf class configured with $rate = U^{oth} * B^{eth}$ and $ceil = rate$. In this way, unreserved traffic has a guaranteed bandwidth chosen by the system administrator and can be managed without compromising existing guarantees. Traffic control management via dynamic creation and deletion of HTB classes on network bandwidth reservations can be performed via user-space tools. Many of them are bundled with typical Linux distributions. In particular, we consider *tcng*[15] , which permits to write configuration scripts in a more flexible and user-friendly way than the *tc* tool. In the architecture described in Section 2, the Cloud Manager component is responsible for receiving information contained in the SLAs stipulated at the upper level and translate it for tcng. An excerpt of a configuration script that can be generated by the Cloud Manager and consumed by tcng is illustrated in Figure 3. It depicts a situation in which five network reservations are in place and all have been negotiated with both the `NetMinBandwidth` and the `NetMaxBandwidth`, on a value equal to $19Mbps$. Each network reservation

```
htb () {
  class (rate 100Mbps, ceil 100Mbps){
   $rewww1=class(rate 19Mbps) {sfq;};
   $rewww2=class(rate 19Mbps) {sfq;};
   $rewww3=class(rate 19Mbps) {sfq;};
   $rewww4=class(rate 19Mbps) {sfq;};
   $rewww5=class(rate 19Mbps) {sfq;};
   $other=class(rate 5Mbps, ceil 5Mbps){sfq;}}}
```

Fig. 3. tcng configuration for enforcing network bandwidth allocations

is characterized by a unique identifier.At the packet level, the traffic directed to a specific consumer is classified by means of filters. As an example, in the configuration of Figure 3 such filter is based on the provider port used by the client for requesting services. It is worth to notice that in the depicted situation, the system is configured with $B^{eth} = 100Mbps$, $U^{res} = 0.95$.

4. Experimental Analysis

This section describes some experiments performed for evaluating the effectiveness of the proposed approach in providing network QoS guarantees for concurrent service requests. In this section we indifferently refer to consumers or tenants for indicating a generic software client. We specifically target private Clouds, where services are available only for members of a small-medium sized organization, connected in a LAN. In particular, we consider two different implementation of private Clouds, a generic SOA scenario and a virtualized one.

Our work focuses on analyzing the network traffic generated by hosts and it does not consider the impact of the network itself on the QoS. However, this latter problem has been widely addressed in the past. For instance, in the case of Switched Ethernet, network guarantees can be achieved by using the Flexible Time Triggered - Switched Ethernet (FTT-SE) protocol[16] and/or by enhancing switches with predictable scheduling capabilities[17]. Our work could be complemented with one of these approaches when the network itself is the bottleneck from a QoS point-of-view.

4.1. SOA Scenario

For the sake of simplicity, this testbed consists of only two machines, a client and a server, connected by a switch through $100Mbps$ Ethernet links. We create a multi-tenancy scenario by means of software but we believe that it could be easily recreated by connecting more machines in a cluster. The server machine acts as provider and it is featured by a $64bit$ Intel CPU running at $1.2GHz$ and has a GNU/Linux OS with a 2.6.28 kernel.The service requested by tenants is a rotation image service, that given a certain resolution r and a rotation angle α, returns the rotated image by sending it back through the network. The image to rotate is a gray-scale one ($8bit$ per pixel), provided by means of Ram-disks with different resolutions. To create a multi-tenancy scenario, the client machine creates (through the *ab - Apache Benchmarking* tool) 5 consumer tasks, and each one connects to the provider by using a different port and independently performs requests. The consumers request the service with parameters r = 512x512 and $\alpha = 20$. The rotated image, sent as response for each request, has a size of $419KiB$.

We are interested in measuring the satisfaction perceived by customers and thus all the results are collected by the client machine. In particular, we measured the transmission time of the service response by leveraging the report functionality of the *ab* tool. For each request, *ab* reports the so-called `wait` and `dtime` values, being respectively the time interval the consumer waits on the socket and the time interval that occurs between the request sending and the last byte received. The transmission time tt of the service response has been measured as the difference $tt = dtime - wait$.

We performed a set of experiments as described in the following. Each experiment has been repeated 20 times. For the reader's convenience the experimental configurations (explained throughout this section) have been summarized in Table 1, where 'Y' indicates the presence of a certain feature, 'N' the absence and '-' the meaningless for that configuration.

Table 1. Experimental Configuration Resume

	Traffic Control	Overloaded Network	Unreserv. Traffic Handling	Varying Rate
Experiment A	N	N	-	-
Experiment B	N	Y	-	-
Experiment C	Y	Y	Y	N
Experiment D	Y	Y	N	N
Experiment E	Y	Y	Y	Y

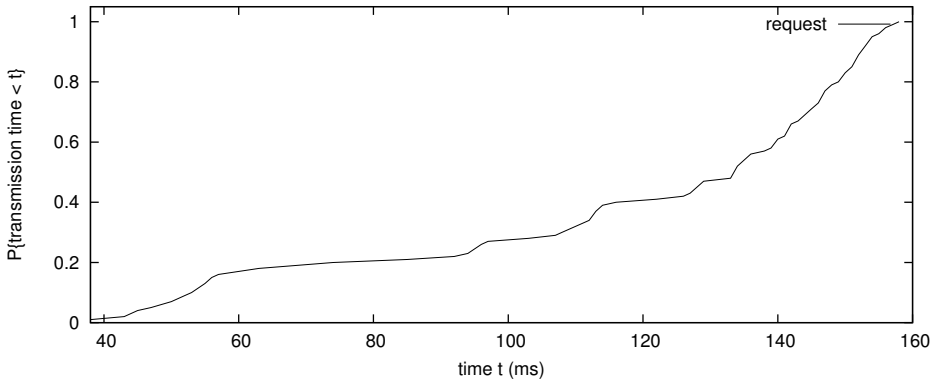

Fig. 4. CDF of service transmission time (Experiment A)

Table 2. Service transmission times (ms) for Experiments A-D

	min	avg	max	std.dev
Experiment A	38	117.95	158	36.63
Experiment B	124	195.3	226	26.28
Experiment C	186	191.3	193	1.51
Experiment D	222	230.6	237	3.03

In Experiment A, the service is concurrently requested by 5 tasks and we measured the service transmission times. The Cumulative Distribution Function (CDF) $C(x) = P\{\mathrm{tt} < x\}$ of the service transmission times has been plotted in Figure 4. It could be seen that there is a high variability in transmission times, mainly due to resource contentions in the network queue of the service provider. Thus, very different transmission times of the image are experienced by each consumer. Such variability can be also observed in the processing times of each request, due to the contention in accessing the CPU. Such problem has already been addressed in a previous work[9] by same authors and thus will not be further discussed.

The behavior pointed out in Experiment A is emphasized when other tasks concurrently use the network, as reported by the following Experiment B. In this case a bandwidth eager task was created for sending UDP packets of size $12.5KB$ every $1ms$, trying to saturate the available bandwidth of $100Mbps$. The bandwidth eager task, illustrative of a possible misbehavior inside the provider, was running while consumers requested the service as of the previous experiment. The collected transmission times are greater than those reported in Experiment A, as can be seen by comparing the first two columns of Table 2 (the 95% confidence values are always below the 2.2%). In particular, adding the bandwidth eager task, the average transmission times increased by the 65%.

Experiment C consists in repeating the previous experiment when the service provider is configured by using the traffic control script of Figure 3, which depicts a situation in which a bandwidth allocation of $19Mbps$ has been assigned to each of the 5 consumers. It is worth to note that each consumer can negotiate a different bandwidth share

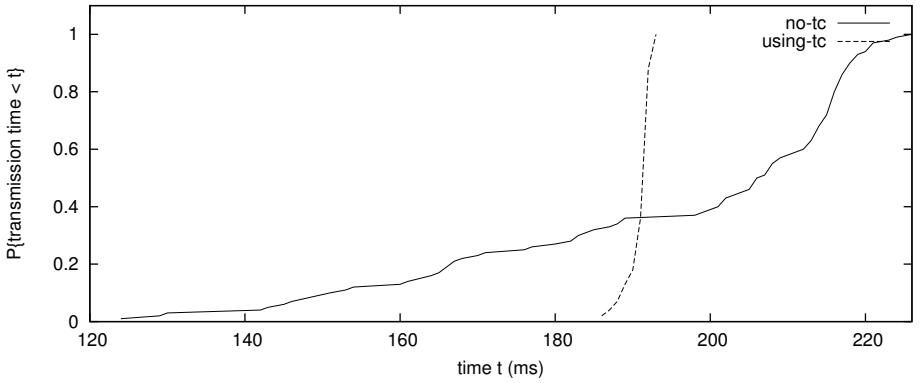

Fig. 5. Comparison of the CDFs of service transmission times for overloaded network (Experiments B, C)

with respect to others and we have only chosen such configuration for the sake of simplicity. The results show a substantial flattening of the transmission times in comparison with Experiment B, due to the drastic reduction in the standard deviation of the transmission time (95% less). The flattening of the transmission times by using traffic control can be appreciated in Figure 5, that shows CDFs of collected results: the label *using-tc* identifies transmission times obtained with the traffic control activated, *no-tc* with traffic control deactivated.

The flattening of the service transmission time enhances the system with a certain degree of determinism and allows each consumer to experience the same QoS level across various requests over time. By repeating Experiment C without the bandwidth eager task, we obtained a mean value of $182ms$ vs $191.3ms$ and a standard deviation of $0.66ms$ vs $1.51ms$. This suggests that the transmission times are not influenced by concurrent requests of other tenants, but they are minimally affected by the bandwidth eager task. This points out some limitations of the Linux TC packet scheduler in handling overloads and suggests the adoption of some precautions, e.g. redefining the *unreserved bandwidth utilization* U^{oth} in a manner that $U^{res} + U^{oth} \neq 1$ (see Section 3.2). Indeed, by progressive lowering U^{oth} such that $U^{res} + U^{oth}$ ranges from 1 to 0.95 showed an improvement of up to $8ms$ in the mean transmission time.

Experiment D has been performed for pointing out a particular behavior of Linux TC. It consists of using the script of Figure 3 without specifying the `other` class, assigned to unreserved traffic. In this way, Linux TC does not find any matching rule for the UDP traffic, that is left out of control. The results, reported in the fourth column of Table 2, show that transmission times of reserved traffic are always flat in comparison of those of Experiment B, but the mean value is greater than the value of Experiment C, because the unreserved traffic is not shaped. Another experiment (Experiment E) has been performed for verifying the precision of the HTB algorithm in allocating network bandwidth for each class. By verifying such precision at different grain levels ($1Mbps$, $0.1Mbps$, $0.01Mbps$) we measured an error up to 1.4% w.r.t. ideal values calculated by using the function $f(bw) = S/bw$, representing the transmission time as a function of the bandwidth bw (with $S = 419KiB$ be a constant equal to the rotated image size sent through the network).

From this set of experiments we can extrapolate some lessons learned. First, the resource management component can safely allocate network bandwidth shares at a minimum grain of $10Kbps$ per tenant, that we believe is sufficient in most cases. Second, the enforcement mechanisms is quite strong and a misbehaving service cannot affect the bandwidth allocation of other services. Third, to provide strong guarantees, the admission control test must be configured with the bandwidth utilization $U^{res} + U^{oth} < 1$. In particular, we observed that the best trade-off was obtained with a value $U^{res} + U^{oth} = 0.95$. By using such configuration, the `NetMinBandwidth` guarantee was provided with a minimal error (1.3%) w.r.t. the ideal value.

4.2. Virtualized Scenario

By using the methodology already described in Section 4.1, we repeated experiments A, B, C in a different testbed. In particular, we were interested to evaluate the capability of providing bandwidth guarantees when the service

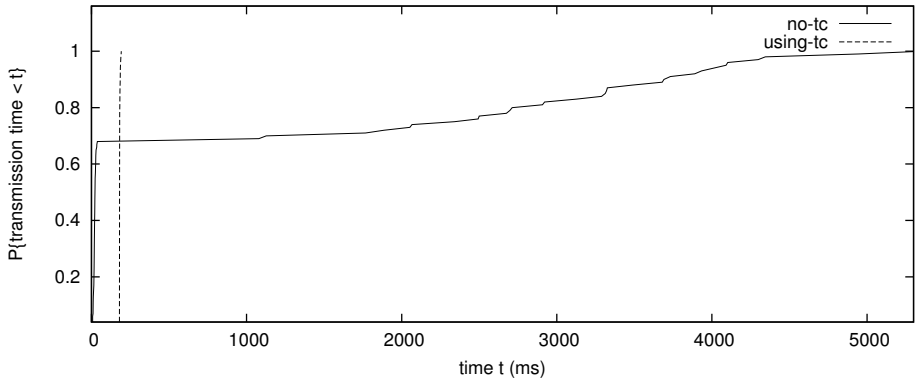

Fig. 6. Comparison of the CDFs of service transmission times for overloaded network in a virtualized scenario

provider software (server) is deployed on a VM. By leveraging the KVM hypervisor on Linux, we deployed the VM to act as a server in a host equipped with a 3.20GHz Intel Xeon CPU and 16 GB of RAM. We assigned to the VM the following resources: 2 CPU cores, 2 GB of RAM and a $100Mbps$ NIC. The service consumer software was executed on another host (acting as a client) with a 2.53 GHz Intel Xeon CPU and 112 GB of RAM. The hosts were connected in a private LAN.

Qualitatively, we observed the same behavior of the previous experiments, that can be resumed as follows:

- the service transmission times are highly variable when there is no traffic control, especially when the bandwidth eager task is in place;
- applying traffic control is effective, even inside a VM, for flattening the transmission times of the service response (and this kind of determinism is not affected by a possible software misbehavior).

In Figure 6 we plot CDFs of collected results for overloaded network in a virtualized scenario: the label using-tc identifies transmission times obtained with the traffic control activated, no-tc with traffic control deactivated.

If we compare the results of Figure 6 with its counterpart represented by Figure 5, we notice that not using traffic control in a virtualized environment can increment the risk of providing low QoS levels. In fact, transmission times grow up to seconds with a probability of the 31%, while they remained in the order of hundreds of ms in the non-virtualized scenario. Instead, by leveraging traffic control inside a VM, we obtain a flattened transmission time, ranging from $180ms$ to $192ms$. It is interesting to note that we obtained very similar results in the non-virtualized scenario, where the service transmission times range from $186ms$ to $193ms$.

5. Related Work

In recent years there has been a growing interest in providing network bandwidth guarantees for Cloud Computing. In fact, the capability of providing bandwidth guarantees may be appreciated by customers that want to run time-sensitive (i.e., soft real-time) applications or just having lower bound predictions of application performances [18]. Some approaches, like NetShare [19] and Seawall [20] ensure fair sharing of congested links (respectively in a per-tenant basis and in a per-source basis) but they do not strictly guarantee a minimum bandwidth to each tenant.

Other approaches address instead the problem of providing bandwidth guarantees, as our work does. Among them, Oktopus [21] provides predictable bandwidth guarantees by addressing VM placement. This work uses a hose model to represent the tenant's bandwidth request but it only considers homogeneous request through VMs. The work by Zhu et al. [22] extends the Oktopus model allowing static reservation for heterogeneous bandwidth requirements. However, these works do not use any reclaiming mechanism for unused bandwidth reservations. Approaches like Gatekeeper [23] and EyeQ [24] assume that the core of the network is congestion-free and they can provide bandwidth guarantee only in that case. LaCurts et al. [25] propose to leverage historical data of bandwidth requirements to predict the future needs of a tenant. However, they assume that an admission control mechanism and a method for enforcing guarantees at the

host level already exist. SecondNet[26] proposes a centralized allocation algorithm focused on allocating Virtual Data Centers (an abstraction for resources allocation) on physical clusters. It provides a solution for VM-to-VM bandwidth guarantees where tenants requests are depicted through a matrix contains the complete pairwise bandwidth between VMs. FairCloud[27] provides an allocation policy for bandwidth sharing and bandwidth guarantees for any VM on the underlying physical network topology. FairCloud has the drawback of requiring expensive hardware support in switches. For overcoming this issue, same authors have proposed ElasticSwitch[28], a solution to provide minimum bandwidth guarantees that can work with commodity unmodified switches. This solution can be implemented inside hypervisors and it does not need any centralized external controller. ElasticSwitch provides minimum bandwidth guarantees through dynamic reservations, by dynamically increasing the rate-limit when there is no congestion.

The latter approaches generally provide guarantees by proposing complete (and thus complex) solutions for building virtual networks between VMs belonging to one tenant. Instead, our work focuses on a single physical host for enforcing bandwidth guarantees. It requires the co-location of VMs belonging to the same negotiated application but it does not require the overhead of setting up additional virtual networks.

Our work leverages the traffic control mechanisms of Linux TC[8] for allocating outgoing network bandwidth. We are aware of one work[29] in Cloud Computing that leverages Linux TC for network bandwidth reservations, however it does not provide any extensive evaluation of the mechanism as we do. Linux TC embeds several packet scheduling mechanism and thus it can be used in different contexts and for different purposes. As an example, the work by Vila-Carbó et al.[30] analyzes the use of Linux TC for real-time transmission, while we focus on bandwidth reservation.

6. Conclusion

Cloud customers rely on providers to supply all of their computing needs. As a consequence providers' services are instrumented to support QoS requirements specified by customers using SLAs formal agreements. In this paper we focused on network bandwidth guarantees, providing a SLA specification and modeling admission control tests. We enforce such guarantees by leveraging Linux TC. To validate our approach, we conducted a set of experiments showing the effectiveness of Linux TC in providing network bandwidth guarantees. In the future we plan to extend this work in two main directions. On the one hand we will explore different resource allocation strategies; on the other hand, we may extend this contribution by supporting an advanced reservation mechanism[31]. We plan to leverage a simulator we developed[32] for reproducing scenarios in which multiple independent providers are aggregated by brokers[33].

Acknowledgments

The authors acknowledge the support of Project FP7-257438, Contrail (Open Computing Infrastructures for Elastic Services), Project FP7-256980 NESSoS (Network of Excellence on Engineering Secure Future Internet Software Services and Systems) and Project FP7-318786 Midas (Model and Inference Driven, Automated Testing of Services Architectures).

References

1. Banerjee, P., Friedrich, R., Bash, C., Goldsack, P., Huberman, B., Manley, J., et al. Everything as a service: Powering the new information economy. *Computer* 2011;**44**. doi:10.1109/MC.2011.67.
2. Vouk, M.A.. Cloud computing – issues, research and implementations. *CIT Journal of Computing and Information Technology* 2008;**16**(4). doi:10.2498/cit.1001391.
3. Huhns, M., Singh, M.. Service-oriented computing: key concepts and principles. *Internet Computing, IEEE* 2005;**9**(1):75 – 81. doi:10.1109/MIC.2005.21.
4. Buyya, R., Yeo, C.S., Venugopal, S.. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*. 2008, p. 5–13. doi:10.1109/HPCC.2008.172.
5. Abeni, L., Buttazzo, G.. Integrating multimedia applications in hard real-time systems. In: *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*. IEEE; 1998, p. 4–13. doi:10.1109/REAL.1998.739726.
6. Palopoli, L., Cucinotta, T., Marzario, L., Lipari, G.. AQuoSA-adaptive quality of service architecture. *Software Practice and Experience* 2009;**39**. doi:10.1002/spe.v39:1.

7. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., et al. Web Service Agreement Specification (WS-Agreement). 2007. URL: http://www.ogf.org/documents/GFD.107.pdf.
8. Hubert, B., Graf, T., Maxwell, G., van Mook, R., van Oosterhout, M., Schroeder, P.B., et al. Linux Advanced Routing and Traffic Control Howto. 2004. URL: http://lartc.org/lartc.html.
9. Cucinotta, T., Mancina, A., Anastasi, G.F., Lipari, G., Mangeruca, L., Checcozzo, R., et al. A real-time service-oriented architecture for industrial automation. *Industrial Informatics, IEEE Transactions on* 2009;**5**(3):267–277. doi:10.1109/TII.2009.2027013.
10. Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer* 2012;**45**(12):65–72. doi:http://doi.ieeecomputersociety.org/10.1109/MC.2012.76.
11. Cucinotta, T., Anastasi, G.F., Abeni, L.. Respecting temporal constraints in virtualised services. In: *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*; vol. 2. 2009, p. 73–78. doi:10.1109/COMPSAC.2009.118.
12. Cucinotta, T., Anastasi, G.F., Abeni, L.. Real-time virtual machines. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium, Work In Progress Session*. 2008, p. 1–5.
13. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.. kvm: the Linux Virtual Machine Monitor. In: *Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07)*. Ottawa, Ontario, Canada; 2007, p. 225–230. Mirrored at https://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf.
14. Devera, M.. HTB Linux queuing discipline manual - user guide. 2002. URL: http://luxik.cdi.cz/~devik/qos/htb/userg.pdf.
15. Almesberger, W.. Linux traffic control - next generation. In: *Linux-Kongress 2002, 9th International Linux System Technology Conference*. 2002, p. 95–103. Available at http://tcng.sourceforge.net/doc/tcng-overview.pdf.
16. Marau, R., Almeida, L., Pedreiras, P.. Enhancing real-time communication over cots ethernet switches. In: *Factory Communication Systems, 2006 IEEE International Workshop on*. 2006, p. 295 –302. doi:10.1109/WFCS.2006.1704170.
17. Varadarajan, S., Chiueh, T.. Ethereal: a host-transparent real-time fast ethernet switch. In: *Network Protocols, 1998. Proceedings. Sixth International Conference on*. 1998, p. 12 –21. doi:10.1109/ICNP.1998.723721.
18. Oberle, K., Stein, M., Voith, T., Gallizo, G., Kbert, R.. The network aspect of infrastructure-as-a-service. In: *Intelligence in Next Generation Networks (ICIN), 2010 14th International Conference on*. 2010, p. 1–6. doi:10.1109/ICIN.2010.5640923.
19. Lam, V.T., Radhakrishnan, S., Pan, R., Vahdat, A., Varghese, G.. Netshare and Stochastic Netshare: Predictable Bandwidth Allocation for Data Centers. *SIGCOMM Comput Commun Rev* 2012;**42**(3):5–11. doi:10.1145/2317307.2317309.
20. Shieh, A., Kandula, S., Greenberg, A., Kim, C., Saha, B.. Sharing the data center network. In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*; NSDI'11. Berkeley, CA, USA: USENIX Association; 2011, p. 309–322.
21. Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.. Towards predictable datacenter networks. In: *Proceedings of the ACM SIGCOMM 2011 Conference*; SIGCOMM '11. New York, NY, USA: ACM. ISBN 978-1-4503-0797-0; 2011, p. 242–253. doi:10.1145/2018436.2018465.
22. Zhu, J., Li, D., Wu, J., Liu, H., Zhang, Y., Zhang, J.. Towards bandwidth guarantee in multi-tenancy cloud computing networks. In: *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE; 2012, p. 1–10.
23. Rodrigues, H., Santos, J.R., Turner, Y., Soares, P., Guedes, D.. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. In: *Proceedings of the 3rd Conference on I/O Virtualization (WIOV'11)*. Berkeley, CA, USA: USENIX Association; 2011, p. 6–6.
24. Jeyakumar, V., Alizadeh, M., Mazières, D., Prabhakar, B., Kim, C., Greenberg, A.. EyeQ: Practical Network Performance Isolation at the Edge. In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association; 2013, p. 297–312.
25. LaCurts, K., Mogul, J.C., Balakrishnan, H., Turner, Y.. Cicada: Introducing predictive guarantees for cloud networks. In: *Proceedings of the 6th USENIX conference on Hot Topics in Cloud Computing*. Philadelphia, PA: USENIX Association; 2014, p. 1–6. URL: https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/lacurts.
26. Guo, C., Lu, G., Wang, H.J., Yang, S., Kong, C., Sun, P., et al. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In: *Proceedings of the 6th International COnference*; Co-NEXT '10. New York, NY, USA: ACM. ISBN 978-1-4503-0448-1; 2010, p. 15:1–15:12. doi:10.1145/1921168.1921188.
27. Popa, L., Krishnamurthy, A., Ratnasamy, S., Stoica, I.. FairCloud: Sharing the Network in Cloud Computing. In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*; HotNets-X. New York, NY, USA: ACM. ISBN 978-1-4503-1059-8; 2011, p. 22:1–22:6. doi:10.1145/2070562.2070584.
28. Popa, L., Yalagandula, P., Banerjee, S., Mogul, J.C., Turner, Y., Santos, J.R.. ElasticSwitch: Practical Work-conserving Bandwidth Guarantees for Cloud Computing. In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*; SIGCOMM '13. New York, NY, USA: ACM. ISBN 978-1-4503-2056-6; 2013, p. 351–362. doi:10.1145/2486001.2486027.
29. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*; EuroSys '07. New York, NY, USA: ACM. ISBN 978-1-59593-636-3; 2007, p. 275–287. doi:10.1145/1272996.1273025.
30. Vila-Carbo, J., Tur-Masanet, J., Hernandez-Orallo, E.. An evaluation of switched ethernet and linux traffic control for real-time transmission. In: *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. 2008, p. 400 –407. doi:10.1109/ETFA.2008.4638424.
31. Konstanteli, K., Kyriazis, D., Varvarigou, T., Cucinotta, T., Anastasi, G.. Real-time guarantees in flexible advance reservations. In: *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*; vol. 2. IEEE; 2009, p. 67–72. doi:10.1109/COMPSAC.2009.117.
32. Anastasi, G.F., Carlini, E., Dazzi, P.. Smart Cloud federation simulations with CloudSim. In: *Proceedings of the First ACM Workshop on Optimization Techniques for Resources Management in Clouds*; ORMaCloud '13. New York, NY, USA: ACM. ISBN 978-1-4503-1982-9; 2013, p. 9–16. doi:10.1145/2465823.2465828.
33. Anastasi, G., Carlini, E., Coppola, M., Dazzi, P.. QBROKAGE: A genetic approach for QoS Cloud brokering. In: *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. 2014, p. 304–311. doi:10.1109/CLOUD.2014.49.