Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 353 (2020) 107–127

www.elsevier.com/locate/entcs

On Performance Evaluation of Security Monitoring in Multitenant Cloud Applications

Lelio Campanile[1] Mauro Iacono[2] Stefano Marrone[3]
Michele Mastroianni[4]

Dipartimento di Matematica e Fisica,
Università degli Studi della Campania "L. Vanvitelli",
viale Lincoln 5, 81100 Caserta (Italy)

Abstract

In this paper we present a modeling approach suitable for practical evaluation of the delays that may affect security monitoring systems in (multitenant) cloud based architecture, and in general to support professionals in planning and evaluating relevant parameters in dealing with new designs or migration projects. The approach is based on modularity and multiformalism techniques to manage complexity and guide designers in an incremental process, to help transferring technical knowledge into modeling practice and to help easing the use of simulation. We present a case study based on a real experience, triggered by a new legal requirement that Italian Public Administration should comply about their datacenters.

Keywords: Simulation; Multiformalism modeling; Multitenant; Cloud; Performance evaluation.

1 Introduction

As established by the Italian law n. 221 approved in 2012, to the Agency for Digital Italy (AGID - Agenzia per l'Italia Digitale) was assigned the task of carrying out a census of Public Administrations (PA) datacenters, in order to propose a rationalization plan. The main goal of this plan is to arrange rules aiming at consolidating the digital infrastructures of PAs, in order to achieve greater levels of efficiency and security in delivering services to citizens and firms.

The census highlighted high fragmentation and oddness in the PA ICT infrastructures, and arranged an evolutionary pathway to drive PAs towards a more

[1] lelio.campanile@unicampania.it
[2] mauro.iacono@unicampania.it
[3] stefano.marrone@unicampania.it
[4] michele.mastroianni@unicampania.it

https://doi.org/10.1016/j.entcs.2020.09.020

efficient and flexible use of ICT technologies in order to guarantee greater reactivity in the provision of services tailored to the needs of citizens and firms. This route is described in the "Three-years Plan for ICT", drafted by AGID in 2019 [1], and involves two activities:

- Rationalization of PA datacenters and consolidation of the less efficient datacenters in few selected quality centers;
- Study and definition of the evolutionary strategic model of PA cloud to be implemented following the rationalization described in the previous paragraph.

The strategic goals of this plan are:

- Increase the quality of services offered in terms of security, resiliency, energy saving and business continuity.
- Create a "PA-Cloud" environment, homogeneous from the contractual and technological point of view, by retraining internal resources existing in PAs or by resorting to resources of qualified external parties.
- Cost savings resulting from consolidating datacenters and migrating services to the cloud.

In order to fulfill the plan, in the next years most of the PAs must dismiss their datacenters and migrate their ICT services in the "PA-Cloud".

Due both to the limited time available, and the different characteristics in terms of price and performance of the various cloud providers, many PAs may approach the migration to the PA-Cloud environment by implementing a Virtual Datacenter using the hybrid model of deployment.

A hybrid cloud infrastructure is a combination of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technologies that enable data and application portability (e.g., cloud bursting for load-balancing between clouds) [6].

This means that the Virtual Datacenter of the PA is constituted by virtual machines placed in different (real) datacenters in a WAN environment, connected to each other by a high speed network infrastructure. In this case, setting up an effective monitoring system may be very difficult, due to the high latency and the need for data to cross various security, obfuscation and and adaptation layers.

We focus on the assessment of a case study, consisting in a Virtual Datacenter, potentially distributed across many different cloud providers, connected to the University network via a dedicated wavelength trail in an high speed DWDM network.

In a traditional datacenter, monitoring of network traffic is usually done using the "port mirroring" feature of network switches and/or using dedicated switch ports, called TAP (Test Access Port). In a Virtual Distributed Datacenter, this approach is useless: existing hardware TAP devices cannot be utilized for virtual network links in order to monitor traffic among Virtual Machines (VM) at a packet level [9].

The right approach for monitoring network traffic in such an environment is using the port mirroring service provided by Openstack [11]m that we assume as

reference cloud support. Port mirroring service involves sending a copy of packets entering or leaving one port to another port, which is usually different from the original and sends to another destination the copies of packets being mirrored. The behavior of a port mirroring service may be sketched as in Figure 1.

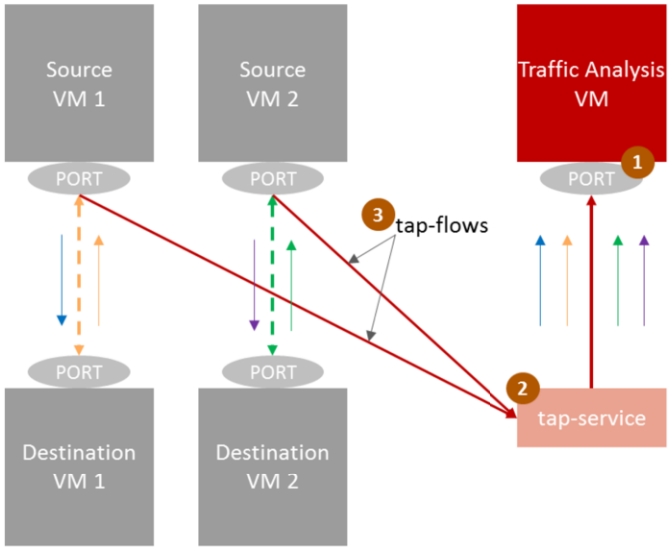

Fig. 1. Virtual port monitoring [11]

The port mirroring feature of a virtual switch can be a naive solution to provide packet level monitoring among VMs. However, using this feature in an environment that needs to treat a large volume of network traffic with low delay such as NFV (Network Function Virtualization) incurs performance degradation in packet switching capabilities of the switch and error-prone manual configurations [9].

Moreover, Tap-as-a-Service (TaaS) is an extension to the OpenStack networking service (Neutron). It provides remote port mirroring capability for tenant virtual networks. This service has been primarily designed to help tenants (or cloud administrators) to debug complex virtual networks and gain visibility into their VMs, by monitoring the network traffic associated with them. TaaS honors tenant boundaries and its mirror sessions are capable of spanning across multiple computing and network nodes. It serves as an essential infrastructure component that can be utilized to supply data to a variety of network analytics and security applications [11].

The behaviour of a Virtual TAP (VTAP) may be sketched as in Figure 2.

Our system consists of various Virtual Machines (VM) that may be hosted in the same (or in different) physical server(s), potentially hosted in different providers. For each VM a VTAP is supposed to be active, which data are sent to a monitoring system, located in the internal boundaries of University network.

In this paper a modular modeling approach is proposed for the evaluation of performances of a service monitoring tool suitable for a virtual datacenter implemented in hybrid model and of the effects of workloads and configurations. The approach is

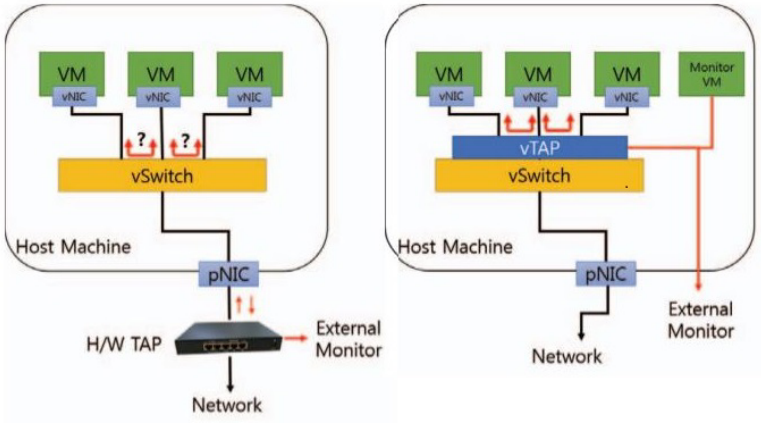

Fig. 2. TAP vs. Virtual TAP [11]

specially oriented to support professionals and policy makers in the design and evaluation process of different solutions. More specifically, we propose a model suitable to analyze the possible delays in the activation of security countermeasures when, as obliged by law in Italy now for Universities, all IT services provided must be run on (multiple) cloud tenants but kept under control by the local infrastructure, that only can and must implement the entry point to services. The model is modular and can be used for any number of tenants, taking into account all possible delays between the University gateway and all tenant infrastructure elements, considering stochastically variable workloads and considering realistic parameters from the measures taken on our University systems. The chosen model setup is a first step towards supporting the analysis of possible alternative solutions to implement our "to be" system, but we developed the model with a general approach and full parameterization to try and guide professionals in analogous projects and to allow a high possibility of reuse.

The paper is organized as follows: Section 3 provides an analysis of the system, Section 4 describes the modeling approach, Section 5 provides a short analysis of our case study, Section 6 draws conclusions and describes future work.

2 Related works

In the last five years, many and many papers had been written approaching issues related to Virtual Datacenters (VD). A relevant survey is [14], in which the different VD request embedding techniques are presented and compared in their functionalities, also discussing how these techniques fared in meeting the Service Level Agreements (SLA) by cloud providers for different Quality of Service (QoS) parameters. Another interesting survey is [16], in which a comprehensive study is presented of the state-of-the-art VM placement and consolidation techniques used in green cloud which focus on improving the energy efficiency. Another relevant paper is also [18], in which performances are analyzed of recently proposed datacenter network topologies, with focus on load-balancing problem. Another interesting reference is [17],

that presents a survey and taxonomy for server consolidation techniques in cloud datacenters. Special attention has been devoted to the parameters and algorithmic approaches used to consolidate VM onto hosts.

Regarding the issues related to cloud-based systems, and how traffic monitoring may be approached in a cloud multitenant environment, in [7] a detailed performance analysis is presented of the Network Function Virtualization (NFV) solutions at both the data and control/management plane, showing the potentials of native SDN adoption within OpenStack towards an integrated solution for production-level NFV deployments. [12] describes challenges to building cloud based fault monitoring systems, and use the output of a production system to understand how virtualization impacts multitenant datacenter fault management. In [3] an analytical queuing model is presented to evaluate the performance of a DPDK-based vSwitch. Such a virtual equipment is represented by a complex polling system in which packets are processed by batches, i.e., a given CPU core processes several packets of one of its attached input queues before switching to the next one. [13] introduces two technologies that can alleviate the network performance issues as faced by Neutron. Furthermore, two technologies, namely OpenDaylight (ODL) and Distributed Virtual Routing (DVR), are then presented together with a set of benchmarks which showcase their performance in a production environment. [9] shows the design and implementation approaches to vTAP using Open vSwitch with DPDK (Data Plane Development Kit) and an OpenFlow SDN (Software-Defined Networking) controller to overcome the problems. DPDK can accelerate overall packet processing operations needed in vTAP, and OpenFlow controller can provide a centralized and flexible way to apply and manage TAP policies in an SDN network.

In order to determine the parameters to be used in our simulation model we refer to literature. [4] describes the causes of latency in optical fiber metro networks, and several available latency reduction techniques and solutions are also discussed, concerning usage of different chromatic dispersion compensation methods, low-latency amplifiers, optical fibers as well as other network elements, and many experimental values are shown. In [5] performances of the networking environment are evaluated with reference to OpenStack, a widely adopted cloud infrastructure management platform. In particular, the paper provides an insight on how OpenStack deals with multitenant network virtualization and evaluates the performance of its main network components by measuring packet throughput in an experimental testbed. [15] presents an analysis of integration of SDN solutions and OpenStack. The architecture of OpenStack networking module or Neutron and plug-ins for communication with SDN is considered. Different types of SDN controllers are also considered in the paper. An experimental model of SDN-OpenStack integration was created according to the solution. Such characteristics of SDN-OpenStack solution as delay, maximum amount of UDP and TCP flows are obtained experiments. In [10] the impact is measured of virtual network on latency in a public cloud based on OpenStack, by measuring the throughput of VM and simultaneously capturing their packets on hosts.

3 System analysis

The modeling process has been based on the analysis of the various delay factors that affect the actual path of the information useful for implementing remote monitoring of each packet that flows through the systems, considering a generic tenant.

3.1 Application gateway

The system is composed of, at least, the entry point of user requests, a tenant that provides applications or the infrastructure to run them, and the interconnection network, that is generally a connection based on a optic fiber.

Requests may be generated by external users or by local users, but are anyway directed towards the application gateway of the organization, that is owned and controlled by the organization and is local to it. This gateway represents the only local computing resource and is naturally conveying all information that is delivered towards the tenants, so it is the most natural element of the system in which the analysis of potential security threats can be performed, as it is the most controllable and directly monitorable part of the system, and is not physically accessible by third parties. Consequently, in coherence with the chosen monitoring approach, all traffic that is logically internal to the system, including request replies and all requests that are generated toward internal software components in order to allow to satisfy requests, has to be sent to the gateway as well to be analyzed by the security monitor. In practice, working on the packet level, only a part of each packet is to be sent back to the monitor to enable the analysis. As the system may provide different services, incoming traffic may be of variable length and (logical) type and may generate a variable traffic towards the tenants. The monitor must examine security related information of each single packet that is transmitted into the system by each couple of software components, and raise and deliver alarms as soon as possible when a threat or a critical condition is detected. Alarms must be sent through the system to the interested nodes, to allow a local reaction according to the designed policies. The origin of requests is not relevant to the problem we are coping with, as, from the point of view of the application gateway, there is no difference between requests originated by the Internet or the local network of the organization.

The delay factors in this subsystem are basically concentrated into the service time and the architecture of the application gateway, the intensity of incoming requests and the volume of internal packets generated by the system, that in turn depends on the nature of the workload of each tenant and the number of tenants.

3.2 Interconnection network

The connection between the application gateway and each tenant is generally implemented by assigning a frequency on a optic fiber to the organization, that then can directly manage the connection. In practice, an optical path is established between the application gateway and each tenant, and the delay is given by the

traversal time of all fibers, all interconnection devices (if any) along the path and all frequency compensation devices. The traversal time of each packet depends on the number and size of packets that are sent between the application gateway and the tenant, as transmission time for a packet is approximately proportional to its size (plus overhead) and packets are queued for transmission. Fibers provide high bandwidth connections, so we expect their contribution to be not a bottleneck, but the delay they introduce in the transmission of security monitoring packets must be considered, and can in practice be measured on the field or obtained by analogous existing setups, as possible configurations offered by providers are quite standard and well known.

3.3 Tenant

We assume for tenants a standard architecture, based on a warehouse scale computing facility that provides standard cloud services. Typically, such a computer infrastructure is composed of a number of computing nodes, or hosts, that are interconnected by a high performance interconnection network, managed by virtualizing computing resources, storage and networking. Virtualization happens by defining Virtual Machines (VM), that run processes. We will consider processes as the relevant computing tasks that are executed on a tenant. VM are connected by Virtual Networking (VN), that might provide communications within the same host or between two hosts, by means of inter-host physical networking. In both cases, the traffic is principally managed by the VN service, that is actually composed of virtual routers that manage traffic inside each host. Virtual routers, analogously to physical ones, may be configured to replicate traffic on virtual ports for diagnostic reasons or other management applications: this mechanism is used to generate packets with the security related information to be sent to the security monitor. Applications that are hosted might operate individually or in collaboration, e.g. in a client-server fashion, or as parts of an application pipeline or a workflow that is orchestrated by a business process management system. Moreover, each process might provide or not a reply that has to be delivered through the application gateway to the user.

The delays introduced in the system are the effects of several factors, that might be considered: actual running time of the processes that implement the applications; in-host virtual networking; inter-host physical networking. The first factor can be obtained by observing existing logs for the same applications, in the case of a migration from a owned datacenter to a tenant, similar information on analogous applications, experiments directly performed on the tenant system or proper analytical models based on knowledge about the applications and the configuration of VM available at the tenant. The second and the third factor can be measured, or can be made public by the tenant, or can be approximately known from literature. An additional, very relevant, indirect contribution is given by the nature of the workload, in terms of number and distribution of process executions needed to satisfy a request, frequency and nature of replies in terms of generated packets, interactions between processes, mapping of processes on VMs and hosts.

4 Modeling approach

The general logic behind our modeling approach is to provide practitioners a reference for dealing with designing and dimensioning parameters in a multitenant system: consequently, we opted for a modular approach in which basic blocks can be replaced with more accurate versions, with different implementations or with blocks using different internal modeling formalisms, according to existing policies and available information.In the following we propose a reference approach, that we believe to be based on a level of details that allows capturing the main elements of the system with parameters that may be easily obtained from generally available experimental data extracted from existing system implementations.

The chosen modeling approach is based on a modular multiformalism modeling technique [2][8], that relies on Petri Nets (PN) and Queuing Networks (QN). The high level model consists of the composition of functional blocks with an internal behavior and input and output ports, that may be connected respectively with multiple output and input ports from other blocks by means of connectors. The semantics of connectors and ports allows multiple connectors from an out port to a single input port, indicating a confluence of packet traffic. Further details are out of the scope of this paper.

The high level model for the system is composed of blocks that represent the application gateway and, for each tenant, the tenant and its interconnection network, as in Figure 3. All interconnection blocks have the same internal parametric structure, as all tenant blocks.

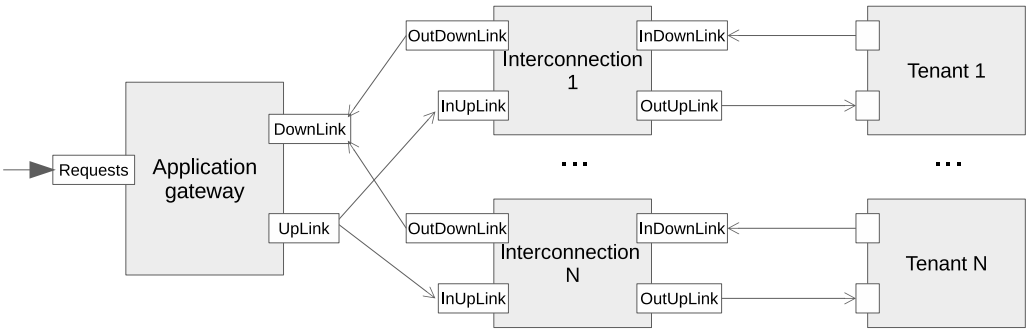

Fig. 3. High level block model of the system

The *ApplicationGateway* block has two inputs, *Requests* and *DownLink*, and one output, *UpLink*. Its internal structure is modeled by a QN submodel. The QN submodel is composed of a single queue, that is characterized by its service time *MonitoringTime*, receives jobs respectively of class *request* and *monitoring* from the two input ports *Requests* and *DownLink* of the block and produces jobs of two classes (*request* and *alarm*). Jobs are produced according to a class based, probability based policy: all incoming *request* jobs produce outgoing *request* jobs, while incoming *monitoring* jobs may generate *alarm* jobs with a parametric probability p_{alarm}. The white box model of the application gateway block is depicted in Figure 4.

The *Interconnection* block has two inputs, *InUpLink* and *InDownLink*, and two

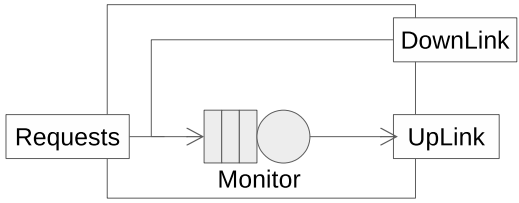

Fig. 4. White box model of the ApplicationGateway block

outputs, *OutUpLink* and *OutDownLink*. Its internal structure is in turn modeled by three blocks, namely *UpLinkFiber*, *DownLinkFiber* and *TrafficCharacterization*, that are connected as in Figure 5. Input *InUpLink* delivers *monitoring* jobs to *UpLinkFiber* and *request* jobs to *TrafficCarachterization*. Jobs produced by *UpLinkFiber* are directed towards port *OutUpLink*.

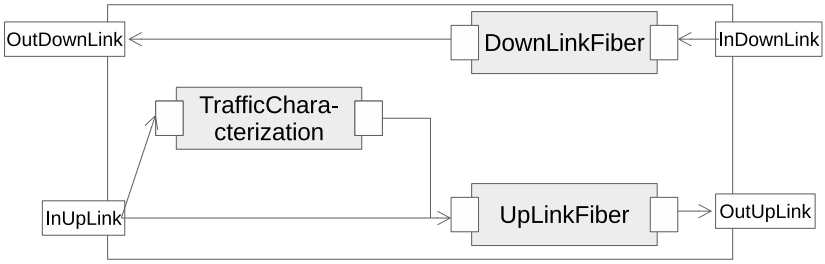

Fig. 5. Grey box model of the Interconnection block

Block *UpLinkFiber* has a single input and a single output, it models the effect of packets sent to the tenant by the uplink fiber connection and consists of a single queue with a service time that corresponds to the transmission rate of the actual available transmission channel, including possible compensations or effects of optical interconnections. The structure of block *UpLinkFiber* is described in Figure 6.

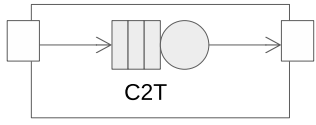

Fig. 6. White box model of the UpLinkFiber block

Block *TrafficCarachterization* allows to generate packets corresponding to different types of incoming *request* jobs, to evaluate the effects of different traffic shapes on the uplink channel. This block has one input and one output port, and is designed to allow a different internal structure according to the characteristics of the incoming request traffic of the system. In our case, to give a general reference, we hypothesize requests of two categories, short or long requests, in a mix that can be defined in probabilistic terms. A short request consists of a single packet, while a long request consists of a sequence of packets followed by a end packet. The number of packets is defined in turn in terms of a probability $p_{anotherpacket}$ of adding or not another packet after each packet sent. The block structure is defined by means of a PN, as in Figure 7.

Each incoming job from the input port generates a token in place *SendReq*, that has two output transitions, *ShortReq*, that characterizes short requests, and *LongReq*, that characterizes long requests. *LongReq* has an output place, *PayloadSent*, that has as output transitions again *LongReq*, to generate another packet, and *SendEnd*, that produces the end packet. Both *ShortReq* and *SendEnd* are connected to the output port to generate outgoing *request* jobs.

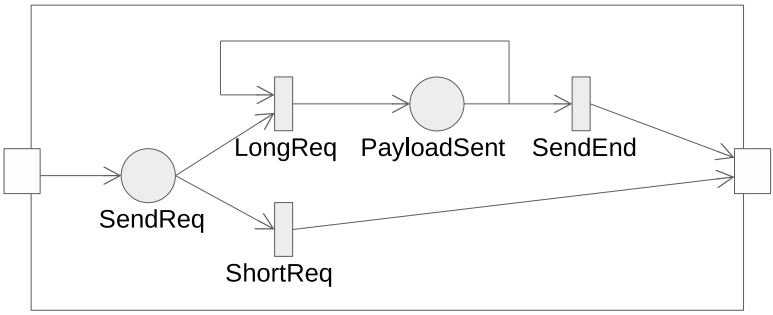

Fig. 7. White box model of the TrafficCharacterization block

Block *DownLinkFiber* has a single input, that receives packets of class *monitoring* and class *request*, and a single output. In this case, to show the flexibility of modularity, we decided to explicitly use the QN variant provided by Java Modeling Tools (JMT), the simulation environment used for model analysis in the next Section. This block models the effect of packets sent from the tenant back to the application gateway by the downlink fiber connection and consists of a single queue with a service time that corresponds to the transmission rate of the actual available transmission channel, including possible compensations or effects of optical interconnections, plus, to include the effects of *monitoring* or *request* class packets, a JMT fork and a JMT join element. The incoming *request* packets represent here replies corresponding to user requests, and we hypotesize that each reply requires n_{reply} packets to be sent to be delivered. The JMT fork produces one *monitoring* job per each incoming *monitoring* job and n_{reply} *request* jobs per each incoming *request* job, while the JMT join operates symmetrically, generating a single *request* job after all n_{reply} incoming *request* jobs have been received. The structure of block *DownLinkFiber* is described in Figure 8.

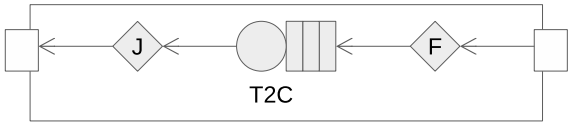

Fig. 8. White box model of the DownLinkFiber block

The *Tenant* block has one input and one output. Its internal structure is in turn modeled by two blocks, namely *Infrastructure* and *ApplicationBehavior*, that are connected as in Figure 9. This block inputs both *alarm* jobs and *request* jobs and delivers them to *Infrastructure*, and outputs *request* jobs and *monitoring* jobs from *ApplicationBehavior*. *Infrastructure* and *ApplicationBehavior* exchange *request* jobs.

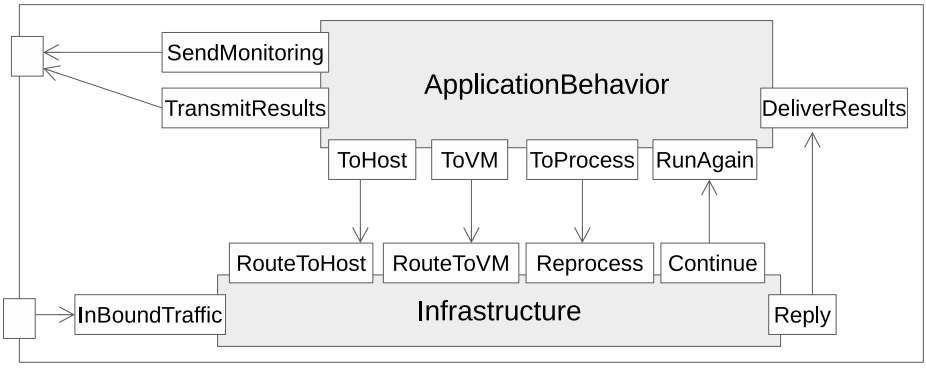

Fig. 9. Grey box model of the tenant block

Block *Infrastructure* represents the computing infrastructure of a tenant in terms of inter-host networking, in-host virtual networking and processing resources. It provides a simple implementation, in which all hosts are equal and there is a complete symmetry of resources and processes. The block has four input ports, namely *InBoundTraffic*, *RouteToHost*, *RouteToVM* and *Reprocess*, and two output ports, namely *Reply* and *Continue*. The internal structure of the block consists of three queues, namely *H2H*, *VM2VM* and *Process*. *H2H* inputs jobs from ports *InBoundTraffic*, that represents the new incoming service requests or alarms, and *RouteToHost*, that represents requests generated inside the tenant by previous requests that have already been processed on another host but ask for further operations, and models the time spent by packets in the inter-host physical networking subsystem. *VM2VM* input jobs from *H2H*, that represents the service requests or alarms received by a host, and *RouteToVM*, that represents requests generated inside the tenant by previous requests that have already been processed on the same host but ask for further operations, and models the time spent by packets in the in-host virtual networking subsystem. *Process* inputs jobs from *VM2VM*, that represents the service requests or alarms received by a VM, and *Reprocess*, that represents requests generated inside the tenant by previous requests that have already been processed on the same VM but ask for further operations, and models the process time of each request or alarm. Alarms processed by *Process* do not further produce effects in the system, while, after processing, requests do not have further effect with probability p_{end} and generate another request to be processed in the tenant with probability $p_{newrequest}$ by outputting a job to port *Continue*, while in the rest of the cases generate a reply towards the application gateway by outputting a job to port *Reply*. The structure of block *Infrastructure* is presented in Figure 10.

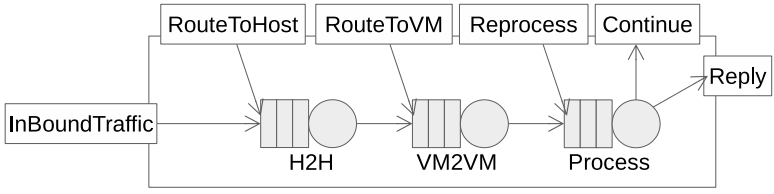

Fig. 10. White box model of the Infrastructure block

Block *ApplicationBehavior* represents the logic according to which processes interact in order to serve a request. It has been modeled as a separate block to allow the isolated design of different replaceable blocks, according to the actual behavior of the workload: in this case, we chose as example a logic that may sequentially generate further internal requests in the tenant, but it is similarly possible to implement behaviors that mimic workflow-like workloads or other organizations. The block has two input ports, namely *RunAgain*, that generates and routes a new internal request, and *DeliverResults*, that inputs jobs representing replies to be delivered to the application gateway [5] . The block has five output ports, namely *TransmitResults*, that sends *request* jobs representing results towards the *Interconnection* block, *SendMonitoring*, that provides *monitoring* packets to the *Interconnection* block corresponding to internally routed requests, *ToProcess*, that delivers requests to a process in the same VM, *ToVM*, that delivers requests to a new VM, and *ToHost*, that delivers requests to a new host. The internal structure of the block is depicted in Figure 11. The internal structure is implemented by a PN model consisting of two separated sections: the first is composed of place Results, that is marked with a token each time port *DeliverResults* receives a job in input, and transition Reply, that generates a job through port *TransmitResults* when firing; the second routes token produced in place *Repr* when a job arrives through port *RunAgain*. Routing happens according to probabilistic criteria: with probability p_{ExitVM} a token in place *Repr* is delivered to transition *ExitVM* to be further rerouted, or otherwise to transition *ReprSameVM*, that generates a job through port *ToProcess* when firing; transition *ExitVM* generates a token in place *OtherVM*, enabling with probability $p_{ExitHost}$ transition *ReprSameT* to produce a job through port *ToHost* when firing, otherwise transition *ReprSameH* is enabled to produce a job through port *ToVM* when firing. Transitions *ReprSameVM*, *ReprSameT* and *ReprSameH* also enable port *SendMonitoring* to generate a *monitoring* packet when firing.

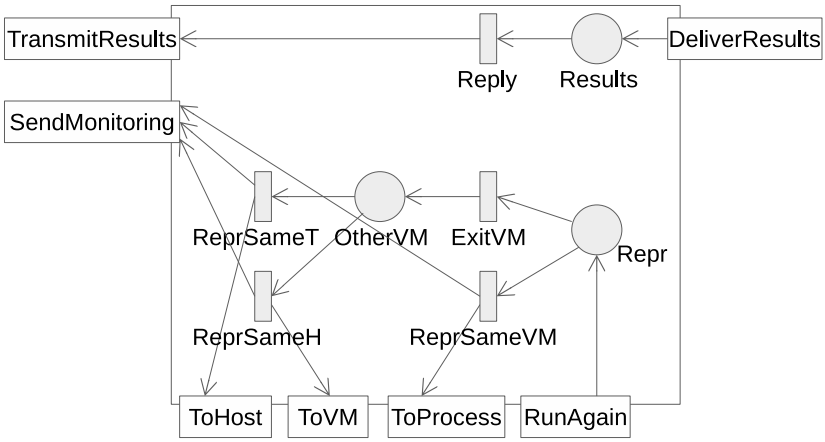

Fig. 11. White box model of the ApplicationBehavior block

[5] This function, and the part of this block that implements it, might also have been moved into the *Infrastructure* block with no essential difference, but we preferred here to use single formalism blocks for the sake of simplicity.

5 Evaluation

To define our analysis strategy for the model, let us first refer to Figure 12.

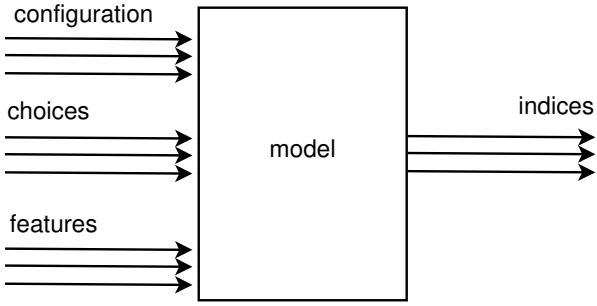

Fig. 12. Input/output view of the simulation model

We introduce three kinds of parameters against whose values the model is analyzed:

- `features`: they collect the variables that are related to some characteristics of the system/environment that cannot change. As an example, in this category we have the parameters as the physical delay in the optical fiber which we deal with as given and unchangeable;
- `choices`: they collect the parameters that can be changed at design-time; as example, by changing the allocation of the services on different virtual machines, the ratio of the packet exiting from the service that remain in the same virtual machine change;
- `configuration`: it collects the set of the parameters that may be changed at run-time by re-configuration actions. As an example, virtual machines can be moved on the different hosts by means of migration technique; in this way, the ratio of the packets exiting from a virtual machine that are directed to the same or to another host may change accordingly.

Table 1 details some of the key parameters also showing the reference value used in the simulations.

Name	Model Element	Property	Reference Value
Features			
ARR_RATE	Requests	Job Arrival Rate	exp(100) [j/s]
ALARM_RATE	Monitor	Routing probability	0.05
FIBER_TIME	C2T-T	Service Rate	exp(1E3) [j/s]
Choices			
MON_SERV	Monitor	Service Rate	exp(2E3) [j/s]
MON_NUM	Monitor	Number of servers	1
LONG_SHORT_RATIO	LongReq-T/ShortReq-T	ratio between the weights	1/9
Configuration			
STAY_EXIT_VM_RATIO	ReprSameVM-T/ExitVM-T	ratio between the weights	1/9
TENANT_NUM	-	number of different tenants	1

Table 1
Model Parameters

Starting from these parameters, the model is analyzed to explore possible project

variants. The quality of the solution is measured by one or more `indices`. Table 2 reports the indices used in this paper.

Name	Description
TH_VM	Throughput of the VM2VM-T station
TH_PRO	Throughput of the Process-T station
TH_HOST	Throughput of the H2H-T station
RT_PRO	Reponse time of the Process-T station

Table 2
Analysis indices

The presented model has been implemented as a simulation model for Java Modeling Tools, that is shown in Figure 13 in a single tenant configuration.

5.1 Varying Arrival Time

The following analysis is a sensitivity analysis, produced varying packet arrival time (in the range 1.0 - 200 j/s) and observing what happens in the infrastructure (see Figure 10). The first graph represents the response time in queue Process (Figure 14), the other Figures represent respectively the throughput of queue Process (Figure 15), queue VM2VM (Figure 16) and queue H2H (Figure 17). In Figure 14 a sharp drop can be observed in performances when the UserRequest arrival rate rises up to 160 j/s: the response time of queue Process increases rapidly. Correspondingly, in Figure 15, which shows the throughput of queue Process, at the same abscissa, a sharp saturation of the throughput is visible. Conversely, Figure 16 represents the queue associated with processes communicating on the same physical host but in different VMs, and, similarly to Figure 17, representing the host to host communication queue, shows an almost linear trend, as the fraction of processes that cause the execution another process on another VM or another host as a consequence of their own execution is small.

5.2 Varying Service Time

The following analysis is a sensitivity analysis, produced varying the service time of queue Process. Service time is varied from 100% to 200% in steps of 10% with respect to nominal parameters. The first graph represents the response time in queue Process (Figure 18), the other Figures represent respectively the throughput of queue Process (Figure 19), queue VM2VM (Figure 20) and queue H2H (Figure 21). In Figure 18 a sharp drop can be observed in performances when the user request arrival rate rises up to 160 j/s: the response time of queue Process increases rapidly. Correspondingly, in Figure 19, at the same abscissa, a sharp drop of the throughput is visible. Figure 20 and 21, instead, show bounded fluctuations (less than 4%) at whatever ratio.

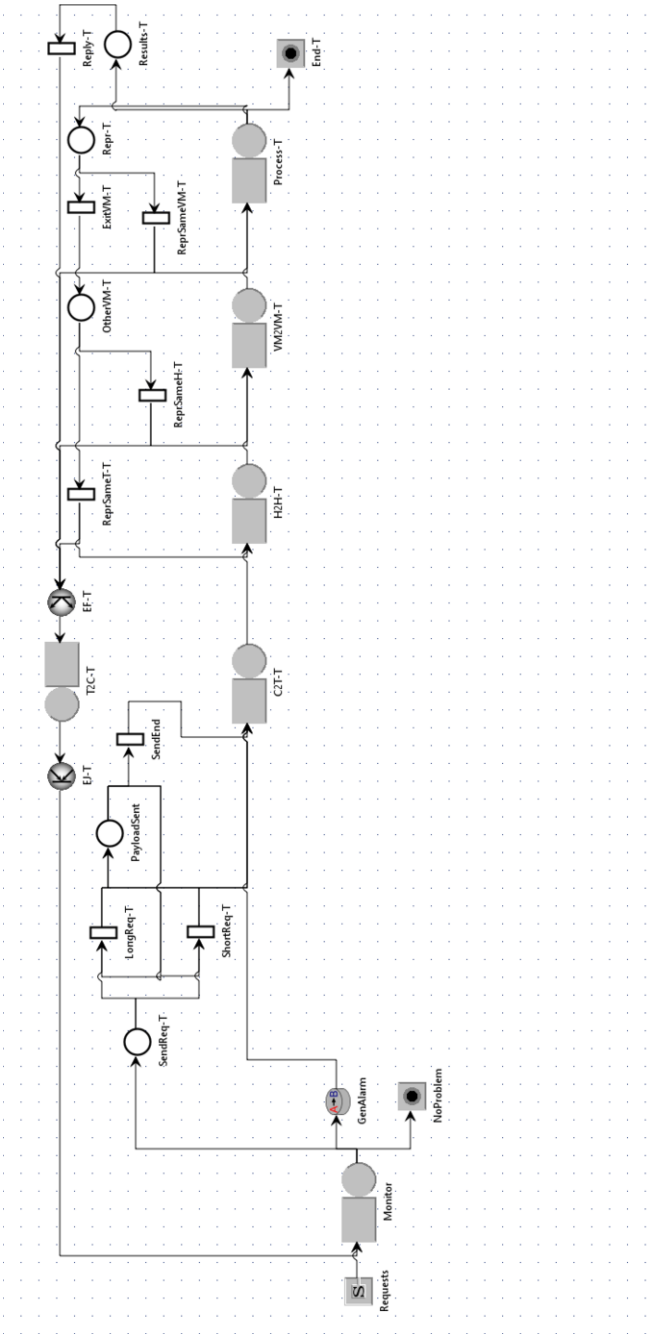

Fig. 13. Simulation model for a single tenant configuration

5.3 Varying the weights of transitions longReq and shortReq

Another test campaign was conducted varying the transition longReq/shortReq. With reference to Figure 7, this means that the weight between single packet (a short request), and a sequence of packets followed by an end packet (a long request) is varied from 10%-90% and 90%-10% in steps of 10%. Results are shown in Table 3.

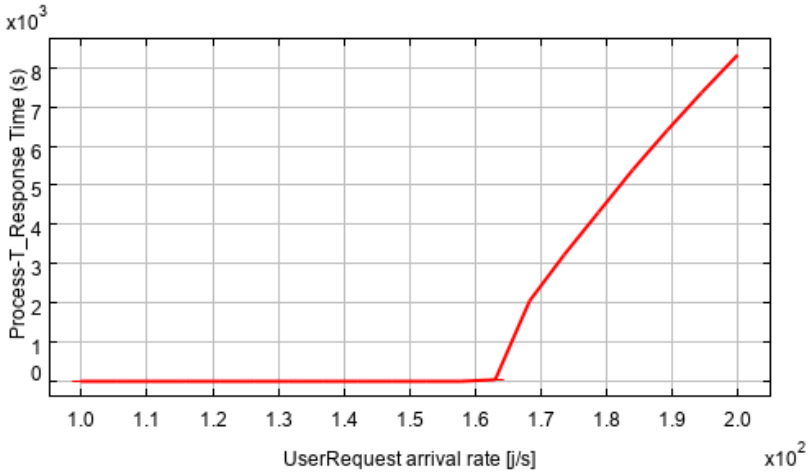

Fig. 14. Response time - queue Process

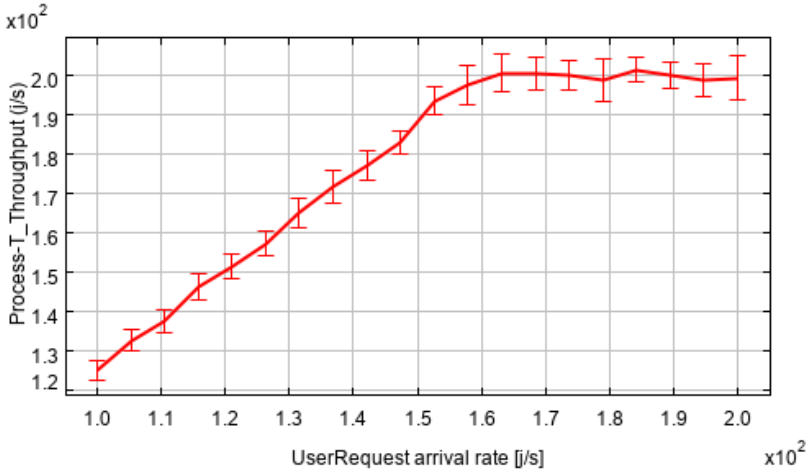

Fig. 15. Throughput - queue Process

Results in Table 3 show that the ratio of the weights of transitions longReq/shortReq has a very limited effect in the throughput of queue Process, due to the chosen parameters, that are realistic, so are such to keep the system stable. Instead, the effect of variations of this ratio is more relevant for queues VM2VM and H2H, because they are obviously more sensitive to incoming traffic.

5.4 Varying the weights of transitions exitVM and sameVM

The last test campaign was conducted varying the transition exitVM/sameVM. With reference to Figure 11, this means that the weight between processes requiring to run other process on the same VM (sameVM), and processes requiring to run other process on another VM (exitVM) is varied from 10%-90% and 90%-10% in steps of 10%. Results are shown in Table 4.

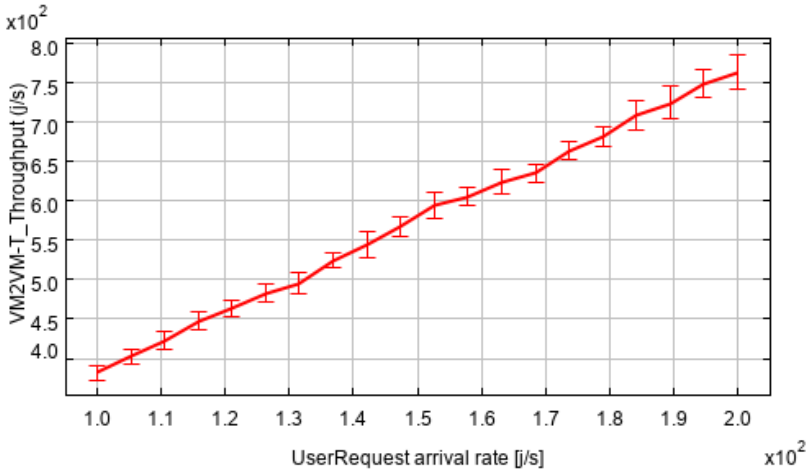

Fig. 16. Throughput - queue VM2VM

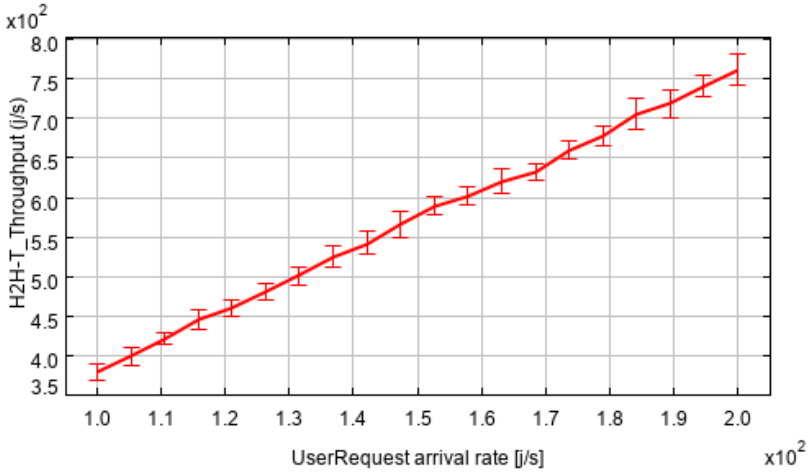

Fig. 17. Throughput - queue H2H

longReq/shortReq ratio	Process-T Throughput [j/s]	VM2VM-T Throughput [j/s]	H2H-T Throughput [j/s]	Process-T Response Time [s]
10%/90%	126.19	386.00	384.21	1.30E-02
20%/80%	123.95	459.56	456.95	1.29E-02
30%/70%	125.99	694.43	704.87	1.16E-02
40%/60%	123.53	615.86	613.03	1.25E-02
50%/50%	124.54	692.61	690.59	1.18E-02
60%/40%	124.91	702.56	698.91	1.18E-02
70%/30%	126.26	711.71	719.73	1.17E-02
80%/20%	124.62	700.03	698.27	1.19E-02
90%/10%	124.31	687.25	692.23	1.20E-02

Table 3
Varying the weights of transitions longReq and shortReq

Fig. 18. Response time - queue Process

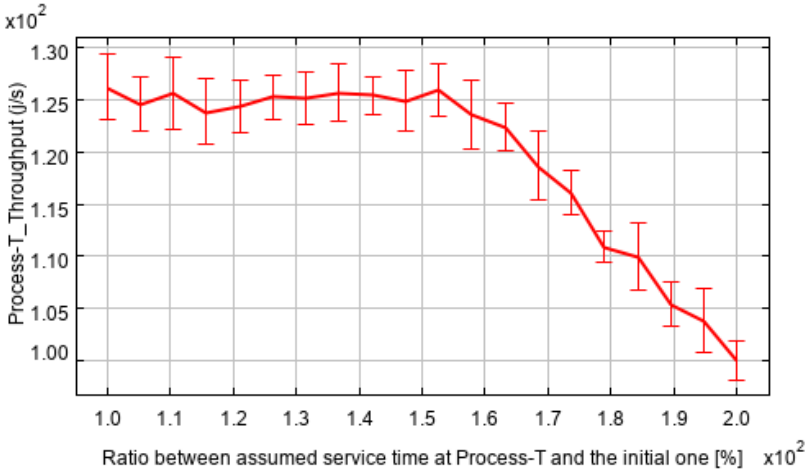

Fig. 19. Throughput - queue Process

Results in Table 4 show that the throughput of queues VM2VM and H2H drops in absolute value (circa 50%) with reference to data shown in Table 3. This is because the growth of exitVM ratio causes that correspondingly more packets return to queue VM2VM to be reprocessed, and the number of jobs/second significantly decreases compared to the previous case. In fact, the ApplicationBehavior block (described in Figure 11) behaves like a "packet multiplier", rising the number of processing requests entering the data center and causing a rise of the throughput for queues VM2VM and H2H. In this case, altering the ratio but with the same probability of generating a reprocessing condition does not alter the throughput of queue Process because its effect is just the alteration of the path that internally generated requests will follow to get to queue Process in the model, actually only impacting on the time they will re-enqueue.

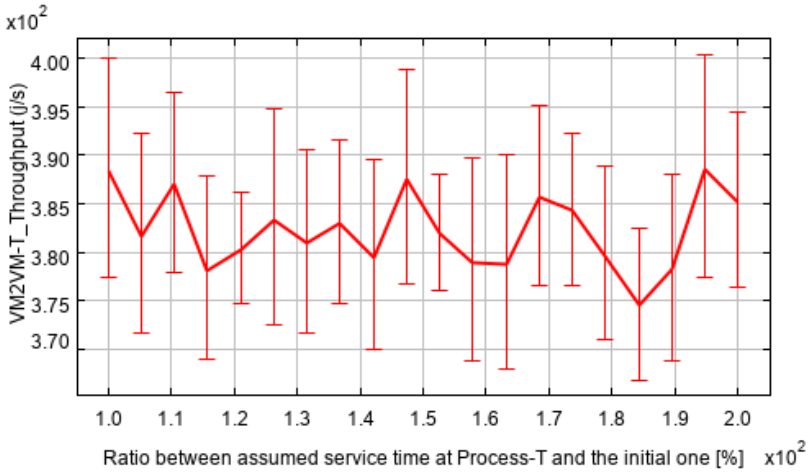

Fig. 20. Throughput - queue VM2VM

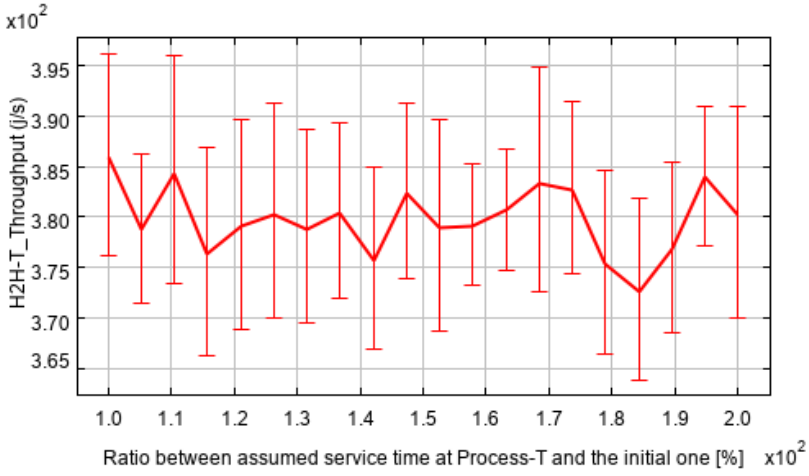

Fig. 21. Throughput - queue H2H

exitVM/sameVM ratio	Process-T Throughput [j/s]	VM2VM-T Throughput [j/s]	H2H-T Throughput [j/s]	Process-T Response Time [s]
10%/90%	125.82	383.19	381.09	1.32E-02
20%/80%	126.71	391.95	387.69	1.30E-02
30%/70%	124.38	387.60	380.36	1.28E-02
40%/60%	125.33	386.96	382.52	1.34E-02
50%/50%	125.59	394.54	382.59	1.31E-02
60%/40%	123.08	386.82	373.52	1.29E-02
70%/30%	124.69	395.78	380.34	1.31E-02
80%/20%	124.36	402.96	379.61	1.30E-02
90%/10%	125.70	406.50	386.35	1.32E-02

Table 4
Varying the weights of transitions exitVM and sameVM

6 Conclusions and future work

In this paper we presented a practice oriented modular multiformalism approach to support performance evaluation and planning in designing cloud based systems, with a focus on multitenant opportunities. The approach aims at supporting practitioners in their activity, and is consequently based on a many-steps modeling process, involving in this case QN and PN as modeling formalism and Java Modeling Tools as evaluation tools. Future work includes the development of incremental models following the future activities of the design team, the evaluation of multitenant scenarios or scenarios with distributed monitoring and the proposal of alternate modeling blocks for workflow based applications, asymmetric datacenter architectures and applications and edge computing based solutions.

References

[1] AGID (2019).
URL https://docs.italia.it/media/pdf/pianotriennale-ict-doc-en/stabile/pianotriennale-ict-doc-en.pdf

[2] Barbierato, E., M. Gribaudo, M. Iacono and A. Jakobik, *Exploiting CloudSim in a multiformalism modeling approach for cloud based systems*, Simulation Modelling Practice and Theory (2018).

[3] Begin, T., B. Baynat, G. Artero Gallardo and V. Jardin, *An accurate and efficient modeling framework for the performance evaluation of dpdk-based virtual switches*, IEEE Transactions on Network and Service Management **15** (2018), pp. 1407–1421.

[4] Bobrovs, V., S. Spolitis and G. Ivanovs, *Latency causes and reduction in optical metro networks*, , **9008**, International Society for Optics and Photonics (2014), pp. 91 – 101.

[5] Callegati, F., W. Cerroni, C. Contoli and G. Santandrea, *Performance of network virtualization in cloud computing infrastructures: The openstack case*, in: *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014, pp. 132–137.

[6] Dillon, T., C. Wu and E. Chang, *Cloud computing: Issues and challenges*, in: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 27–33.

[7] Foresta, F., W. Cerroni, L. Foschini, G. Davoli, C. Contoli, A. Corradi and F. Callegati, *Improving openstack networking: Advantages and performance of native sdn integration*, in: *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[8] Gribaudo, M. and M. Iacono, *An introduction to multiformalism modeling*, in: M. Gribaudo and M. Iacono, editors, *Theory and Application of Multi-Formalism Modeling*, IGI Global, Hershey, 2014 pp. 1–16.

[9] Jeong, S., D. Lee, J. Li and J. W. Hong, *Openflow-based virtual tap using open vswitch and dpdk*, in: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–9.

[10] Lee, C., K. Asano and T. Ishihara, *The impact of software-based virtual network in the public cloud*, in: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 494–499.

[11] Openstack, *Openstack security guide*, in: *OpenStack Security Guide*, 2019.
URL https://docs.openstack.org/security-guide/index.html

[12] Roy, A., D. Bansal, D. Brumley, H. K. Chandrappa, P. Sharma, R. Tewari, B. Arzani and A. C. Snoeren, *Cloud datacenter sdn monitoring: Experiences and challenges*, in: *Proceedings of the Internet Measurement Conference 2018*, IMC '18 (2018), pp. 464–470.

[13] Saghir, A. and T. Masood, *Performance evaluation of openstack networking technologies*, in: *2019 International Conference on Engineering and Emerging Technologies (ICEET)*, 2019, pp. 1–6.

[14] Sivaranjani, B. and P. Vijayakumar, *A technical survey on various vdc request embedding techniques in virtual data center*, in: *2015 National Conference on Parallel Computing Technologies (PARCOMPTECH)*, 2015, pp. 1–6.

[15] Tkachova, O., M. J. Salim and A. R. Yahya, *An analysis of sdn-openstack integration*, in: *2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S T)*, 2015, pp. 60–62.

[16] Usmani, Z. and S. Singh, *A survey of virtual machine placement techniques in a cloud data center*, Procedia Computer Science **78** (2016), pp. 491 – 498, 1st International Conference on Information Security & Privacy 2015.

[17] Varasteh, A. and M. Goudarzi, *Server consolidation techniques in virtualized data centers: A survey*, IEEE Systems Journal **11** (2017), pp. 772–783.

[18] Zhang, J., F. R. Yu, S. Wang, T. Huang, Z. Liu and Y. Liu, *Load balancing in data center networks: A survey*, IEEE Communications Surveys Tutorials **20** (2018), pp. 2324–2352.