Emulating an Infrastructure with EASE

Arup Raton Roy*, Shihabur Rahman Chowdhury†, Md. Faizul Bari†, Reaz Ahmed†, and Raouf Boutaba†
*Arista Networks
arup@arista.com
†David R. Cheriton School of Computer Science, University of Waterloo
{sr2chowdhury | mfbari | r5ahmed | rboutaba}@uwaterloo.ca

***Abstract*—In the last decade we have observed a tremendous adoption of distributed applications and a trend to host services in private or public clouds. However, service providers still need to own an infrastructure to test their applications or services. A similar problem is faced by network operators when they want to introduce a new service in their production network. It is very difficult to determine the behavior of a new application or service without deploying it in the production environment. Bugs or misconfiguration can cause service outage and trigger customer churn along with loss of reputation. There are several publicly available testbeds such as Emulab, GENI or OFELIA that allow users to lease physical and virtual resources for emulation. However, these testbeds do not provide performance guarantee. Acquisition of physical instances provides performance guarantee and isolation, but compromises overall system utilization. On the other hand, acquisition of virtualized instances lack guarantee and isolation resulting in an unrealistic emulation outcome. To address these limitations, we propose EASE, a next generation multi-tenant infrastructure emulator with an aim to maximize hardware utilization while providing performance guarantee, isolation and full-fledged support for SDN and NFV.**

I. Introduction

Emulation **a**s **a** **S**ervic**e** (EASE) is a distributed virtualized testbed that can emulate an entire infrastructures consisting of compute, storage and networking resources. The primary reasons behind developing a new testbed are providing performance guarantee, reproducible emulation environment, supporting more users and finally providing performance isolation between testbed users. None of the existing state-of-the-art testbeds, *e.g.*, Emulab [1], GENI [2], and OFELIA [3], support these features. We performed a simple experiment with Emulab to provide a motivating example.

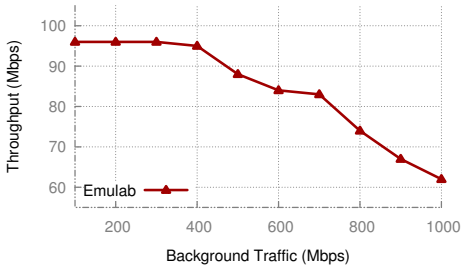


Fig. 1. Traffic throughput in Emulab

We deployed Internet2 [4] topology on Emulab using Xen virtual nodes. As per Emulab's policy, the maximum link bandwidth was set to 100 Mbps. We generated background traffic by randomly selecting some links and sending UDP packet bursts. Then we selected a fixed link (not part of the background traffic), and measured its maximum achievable throughput periodically while increasing the amount of background traffic. Fig. 1 shows the measured throughput with increasing background traffic. After the background traffic goes beyond 400 Mbps, throughput starts dropping sharply. This situation demonstrates that Emulab cannot provide performance guarantee even between the resources belonging to the same user. GENI and OFELIA were developed based on Emulab node [2], so they also suffer from similar problems.

EASE overcomes this problem by ensuring performance guarantee and isolation between testbed users. It allocates virtual resources such as Virtual Machines (VMs), Virtual Switches (VSs), and Virtual Links (VLs) for each user based on their emulation specification. Moreover, EASE uses the notion of *time dilation* to support more users than other testbeds with the same amount of physical resources. Specifically, it can slow down time for a VM or VS by altering the time management mechanism of the hypervisor and provide the illusion of more resource than what is actually allocated on the physical infrastructure. This way EASE can provide resource guarantee even if sufficient resources are not available.

The requirement for an infrastructure emulator such as EASE is driven by the increasing number of distributed applications and systems such as social networks, steaming services, smart-phone applications, *etc*. If a Service Provider (SP) is updating a service or deploying a new service, then there is no way to test the performance (or functionality) of the service beforehand without deploying it in the production system due to the following factors: (i) equipment purchase and maintenance cost and (ii) most SPs use cloud based services. In a real deployment, the service needs to work smoothly under different types of load and failure scenarios to avoid customer churn and loss of reputation. This brings in the compelling need for infrastructure emulators, allowing SPs to test a service under different emulated conditions.

Our key contributions in this paper are as follows:

- EASE dilates time to emulate a virtual infrastructure on top of a physical infrastructure with insufficient resources.
- EASE provides guaranteed allocation of virtual resources with time dilation, and enables better resource sharing.
- EASE supports multi-tenancy by soft-partitioning the physical resources, while providing complete isolation between the users.

The rest of the paper is organized as follows. First, we briefly discuss the related works in Section II. Then, we list

978-3-901882-85-2

the testbed features and the design challenges in Section III. The overall architecture is presented in Section IV. After that we describe our techniques for resource provisioning (Section V), time dilation (Section VI), and resource optimization (Section VII). Next, performance evaluation of the system is presented in Section VIII. Finally, we conclude in Section IX with some future works.

II. Related Works

Emulab [1] allows users to acquire both physical and virtual resources. However, it has several limitations such as no performance guarantee and no out-of-the-box support for SDN. GENI [2] and OFELIA [3] are federated testbeds spread across a number of US and EU sites. They enable simultaneous experimentation with different applications and protocols by different users. However, they also suffer from similar problems as Emulab. Mininet [20], Maxinet [21] and EstiNet [22] are emulators for SDNs. Mininet emulates the entire network on a single machine, and thus fails to scale with network size and traffic volumes [10]. Maxinet proposes to distribute Mininet over multiple machines, but cannot provide resource guarantee. EstiNet also provides distributed emulation across multiple machines with an added feature of time dilation. However, no details is available regarding the technique used by EstiNet.

A couple of research works discuss methods to dilate time in hypervisors. One of the early works modifies the Xen hypervisor to dilate time of a VM during boot time [8]. This dilation factor remains fixed for the VM's lifetime. A more recent work [9] discusses how dilation factor for VMs running on `kvm-qemu` hypervisor can be changed to adapt to system load. However, this modified kvm hypervisor does not support separate `tdf` for each VM, thus making it unfit for our purpose.

III. Testbed Features and Design Challenges

EASE provides several features that makes it possible to have a simple and seamless emulation workflow. A brief overview of these features along with the challenges to implement them is presented in the remainder of this section.

A. Multi-tenancy

EASE supports multiple tenants, each running one or more emulations on the same physical infrastructure. In order to ensure proper isolation between the tenants we need to ensure the following forms of isolation:

Namespace Isolation: A user's namespace should be isolated from other users. It enables multiple users to use the same name for their virtual resources. For example, if a user names one of his VMs as `vhost-a`, this must not restrict other users to use the same name for their virtual resources.

Performance Isolation: Each user should have performance guarantee. Resource consumption by one user should not affect the perceived performance of another user.

Infrastructure Isolation: From a security point of view, a user should not be able to access the underlying physical infrastructure. A user should only be able to access the allocated virtual resources.

B. Resource Guarantee with Time Dilation

EASE utilizes time dilation to provide *perceived* resource guarantee if sufficient physical resources are not available. Time dilation is more suitable for CPU and throughput sensitive emulation compared to latency sensitive emulation since time dilation increases the execution time for the emulation. In order to implement this feature, we are faced with the following technical challenges:

Dilation of Virtualized Resources: EASE provides resource guarantee without hard partitioning the physical resources. To stretch the physical limits of the infrastructure, we use the notion of time dilation. EASE provides the users with a virtual notion of time. Each unit of virtual time corresponds to `tdf` (≥ 1) units of real time, hence, provides the users an illusion of scaled up resources. For example, with `tdf` = 2, it is possible to emulate a 100Mbps virtual link by allocating 50Mbps of physical bandwidth.

Per User Time Dilation and Time Synchronization: EASE provides a user with a consistent time dilation factor (`tdf`) for all of its resources. However, `tdf` can be different for different users. Moreover, `tdf` modification for one user should not impact the others. To this end, we have found two research works to dilate time in hypervisors [8], [9]. However, as discussed in Section II, they are not fit for our purpose.

It is challenging to adaptively change `tdf` of a user and provide different `tdf` to different users. This is further complicated by the fact that the `tdf` of a user's virtual instances distributed across different physical machines should also be consistent. For example, if timestamp calculation for a dilated VM does not consider the clock cycles spent to non-dilated parts (*e.g.*, memory, storage, *etc*.) then a user expecting to perceive a `tdf` of 2, might perceive a `tdf` of 1.98 for VMs on one physical machine and 1.97 for VMs on a different physical machine.

C. Automated Embedding

Once a user specifies the topology, type of devices, VM images, *etc*., EASE takes care of the embedding. It determines the optimal placement of the virtual resources and interconnects the VMs and VSs with IP tunnels.

A major challenge in automating the embedding process is to optimally place a user's virtual resources on the physical infrastructure while minimizing resource fragmentation and maximizing physical infrastructure utilization. Different bin packing problems reduce to such resource allocation problems, hence, they are NP-hard to solve. Therefore, we need to design an effective heuristic that can achieve the aforementioned goals within reasonable execution time.

D. SDN from the Ground Up

EASE supports SDN from the ground up. EASE creates the switching fabric using OVS [6], which allows EASE users to create an SDN network without any manual configuration unlike other testbeds.

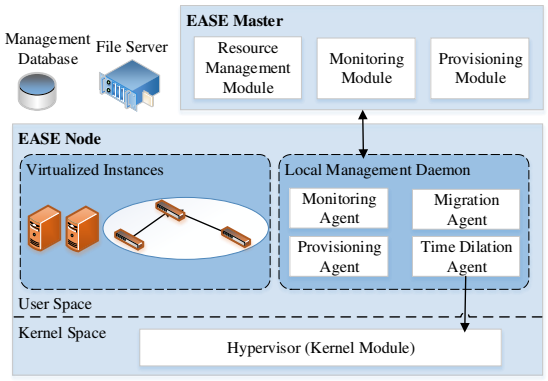


Fig. 2. EASE Architecture

E. Transparency

EASE deploys various specialized components to enable distributed deployment and emulation, which are kept transparent from the user. Users are given the illusion that only their requested infrastructure is being emulated. EASE provides resource guarantee by partitioning the virtual infrastructure and deploying it across multiple physical machines. To incorporate time dilation for the network, EASE deploys the switches inside a VM. In such partitioning, one virtual link might be mapped to a physical path, which brings forth technical challenges in neighbor discovery for the switching fabric. Transparency ensures that no modifications are required in a user's virtual switching fabric.

IV. System Architecture

In this section, we describe the architecture and different components of EASE. Fig 2 shows the overall architecture of EASE. We have two types of physical machines in our system: `EASE Master` and `EASE Node`. `EASE Node`s host the virtual instances (VMs, VSs and VLs) of a user's emulated infrastructure. The operations of `EASE Node`s are orchestrated by the `EASE Master`. The `EASE Master` and `EASE Node`s communicate via Remote Procedure Call (RPC). EASE also maintains *Management Database* and *File Server* for storing configurations, monitoring data, VM images and snapshots.

A. EASE Master

`EASE Master` consists of the following modules:

1) Provisioning Module: Provisioning module embeds a user's virtual resources. It takes an embedding request from a user that contains the topology, resource requirements, and Service Level Objectives (SLOs) (*e.g.*, availability, emulation duration). This module then determines suitable placements for the virtual resources by considering physical resource constraints. EASE deploys emulated topologies across multiple physical machines to achieve scalability.

2) Resource Management Module: Resource management module periodically runs to tune different parameters of the virtual resources for increasing hardware utilization. Specifically, it changes the virtual to physical resource mapping, adjusts the share of physical resources (CPU, bandwidth, *etc.*) and modifies different QoS parameters (*e.g.*, link delays).

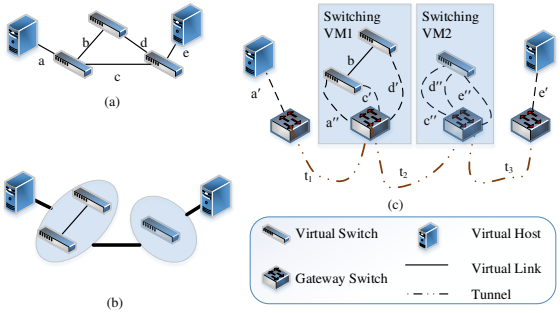


Fig. 3. Virtual Infrastructure Deployment

3) Monitoring Module: This module collects the system statistics periodically and stores them in the database. The statistics include the utilization of the physical and virtual CPUs and usage of the network.

B. EASE Node

`EASE Node` hosts the virtual instances (VMs, VSs, VLs) from different simultaneous emulations. It also runs a local management daemon which handles commands from the `EASE Master`. This daemon maintains several interfaces with different user processes, and the host kernel to provision, monitor and modify virtual instances. Next, we briefly explain each component:

1) Provisioning Agent: This agent spawns VMs mapped only to this physical machine using hypervisor specific APIs. It also creates the virtual switches and virtual links and configures them accordingly. Finally, it creates inter-host virtual links to connect all the partitions of a virtual infrastructure provisioned across different physical machines.

2) Monitoring Agent: Monitoring agent runs at a preconfigured time interval and gathers resource usage of a particular `EASE Node`. It monitors both physical and virtual resources and reports them to the `EASE Master`.

3) Migration Agent: This agent starts inter-host VM migrations when triggered by the `EASE Master`. Additionally, it also changes the vCPU assignment of a VM dynamically.

4) Time Dilation Agent: Time dilation agent keeps an interface with the host kernel to set the `tdf` of a VM dynamically. In case of inter-host migration, it also updates the virtual time from which the VM resumes its operation.

V. Emulation Provisioning

In this section, we explain how EASE ensures the connectivity between the virtual components of an emulation deployed across multiple physical machines. This will give a precise idea of the how both isolation and transparency are ensured by the design.

A. Network Partitioning

Recall that EASE leverages time dilation to utilize the underlying infrastructure by running different emulations with different `tdf`s. Therefore, both the switching fabric and the end hosts (user VMs) need to be dilated. This criteria restricts us to run the switches inside VMs.

Now, we can take two trivial approaches to put the switching fabric in a VM: (i) run the entire switching fabric inside a single VM, or (ii) deploy each switch (router) in a separate VM. Both of these approaches have their pros and cons. Coalescing the entire switching fabric, and running it in a single VM creates a large VM in terms of processing requirements. Fulfilling its resource requirement is harder when the physical resources in the infrastructure gets fragmented over time. One can solve this issue by deploying each switch in a separate VM. However, this solution results in higher numbers of VM to guest kernel packet traversals for inter-switch traffic compared to the first approach. So, we wanted to find a suitable trade-off between these two approaches.

We partition the switching fabric considering the processing requirements and inter-partition link bandwidth (Section VII). After the partitioning phase, we obtain a *reduced topology* for the emulation, which is also referred to as the *reduced graph* in the text. We place the subgraph of a switching fabric assigned to one physical host inside a single VM and connect the VMs using IP tunnels (Fig. 3).

B. Connectivity Establishment

We connect two VSs mapped to the same network partition using Linux IP Link. We choose tunnel to connect all VM to VS links and all inter-partition VS to VS links. The choice of tunnel provides us the desirable unicast forwarding for any traffic passing through the physical switching fabric.

In order to ensure transparency and hide the segments of an emulated link, we adopt a similar approach as in [10]. We create some stub switches, called *gateway switches*. A gateway switch encapsulates a packet (adds tunnel header) and unicasts it to the corresponding destination. When the gateway switch at the other end of the tunnel receives a packet, it strips out the header, and forwards it to the respective switch port. The entire process of encapsulation and decapsulation is kept hidden from the emulation to ensure transparency.

C. Traffic Forwarding and Isolation

Any two partitions of the network might have several Virtual Links (VLs), mapped on tunnels. A tunnel is identified by the IP addresses of the machines where its two end points reside. In order to uniquely identify the packet of VLs, a unique VL id is assigned to each inter-partition VLs.

When a gateway switch receives a packet from an inter-partition VL, it tags the packet with the VL id along with the encapsulation. At the receiving end, this tag helps the gateway switch to forward packet to the corresponding VS. The id and the corresponding forwarding rules are configured statically during the deployment of the emulation. This feature enables an emulation to use its own preferable IP address and *hostname*.

VI. Timer Management

A. Per User Time Dilation

Each EASE user is provided with a uniform `tdf` across all virtual resources. We were faced with the following challenges in this regard: (i) time dilate the user's VMs with user provided VM images, (ii) `tdf` synchronization for all virtual resources running on multiple physical machines.

We modified the hypervisor's time management subsystem to provide a dilated view of time to a user's virtual instances. We modified `kvm` hypervisor to emulate the execution of `rdtsc` instruction from x86 processors instead of running the instruction on hardware. To emulate `rdtsc` instruction we maintained the following information in a dictionary (indexed by the process ids of a VM's vcpus) called `tdf_dict`: (i) `tdf`, (ii) last seen virtual (`pvtsc`), and (iii) real (`prtsc`) time stamp counters of a VM. When a VM issues a `rdtsc` call and the current real time stamp counter is `crtsc`, we return a virtual time stamp counter (`cvtsc`) to the VM calculated using the following equation: $cvtsc = pvtsc + (crtsc - prtsc)/tdf$

B. Virtual Time Synchronization

The next issue in time dilation is to make sure all the virtual resources belonging to a user across different physical machines have the same view of time dilation. Inconsistency in the view of dilation can arise because all the physical machines might not execute the time dilation change command at the same global time. Therefore, all the VMs may not start dilating time at the same moment.

Distributed global time synchronization has been shown hard to solve in the literature. In this paper, instead of providing a full proof solution, we take an engineering approach to mitigate the impact of asynchrony. When commands are issued from EASE master to the EASE nodes to change the `tdf` of VMs, that command also contains a time in the future, which tells the EASE node when to execute the command. In this case, the inconsistent states of a user's VMs depend on the time differences between the physical machines.

VII. Resource Management

Our resource management scheme aims to provide guaranteed resources for an emulation running on EASE. Additionally, we also bound the emulation turnaround time by allocating enough virtual resources. In what follows, we describe our embedding process for an emulation request and setting an initial `tdf` for an emulation.

The initial mapping phase takes care of a newly arrived emulation. A user provides EASE the resource requirements for the infrastructure that he wants to emulate. This request contains the network topology, hosts and their resources (*e.g.*, number of vCPUs, Memory, Disc, *etc.*), and links with their bandwidth and propagation delay. The user needs to specify a lease time which is interpreted as the amount of time needed to run the emulation. Moreover, user also provides a maximum `tdf` value that he is willing to tolerate. From this `tdf` value and the lease time, we can deduce the turnaround time within which the emulation must be finished. Our embedding algorithm allocates resource and set an initial `tdf` in presence of a bound on the turnaround time. However, such resource allocation problems are typically NP-hard to solve. Therefore, we propose a heuristic algorithm to solve it.

A. Heuristic Algorithm

Our heuristic solution (Algorithm 1) takes a virtual infrastructure topology $G = (H, S, L)$ and maximum tdf $\mathcal{T}$ as input, where H, S, and L are the set of VMs, VSs, and VLs, respectively. Each node $n \in N = H \cup S$ has a resource requirement c_n^r for each resource type $r \in R$. Moreover, each VL l has a bandwidth requirement b_l. Our goal is to find a partition of G, map the virtual instances, and set the best possible initial tdf for emulation. If a feasible embedding is not found, we reject the request. This algorithm uses procedures **Partition**, and **FindMaapping**, to embed G.

The Partition procedure (Algorithm 1) groups some VSs together and forms a reduced graph (Section V) from G. It also sets the least possible tdf for embedding. This procedure first sorts the VSs according to the their resource requirements. We consider the bandwidth of incident VLs as a measure of a VS's processing need. We also use an empirical value w to take the link type and protocol into account. The following equation provides the maximum processing requirement c_s for a VS s in terms of the fraction of CPU core, where, $u_{sl}, v_{sl} \in \{0, 1\}$ determine if a VS s is endpoint of a VL $l = (u, v)$:

$$c_s = w \sum_{\forall l \in L} (u_{sl} + v_{sl}) b_l \tag{1}$$

Then, we use First Fit heuristic [11] to place a VS to a partition. The capacity of each partition is considered in terms of CPU. This approach eventually creates VMs of switches with less resource requirement and provides more flexibility to embed these VMs later in the stage. This step only considers packing efficiency and does not take the VL bandwidth into account. After this step, the algorithm iterates through all inter-partition links to minimize inter-partition bandwidth by collocating their end point VSs.

Algorithm 1: Emulation Embedding

```
function Embedding(G, 𝒯)
    T ← ∞, (T_min, G^R) ← Partition(G)
    if T_min > 𝒯 or FindMapping(G^R, 𝒯) = false then
        Reject Request and return false
    else
        T = min. T ∈ [T_min, 𝒯] : FindMapping(G^R, T) = true
    Set initial tdf to T
    return true
```

After the aforementioned step, we get the partition of the network. However, the bandwidth allocation on some virtual interfaces might be higher than their capacity. When this situation occurs, this partitioning will only work when time is appropriately dilated, *i.e.*, the tdf is above a required value. The Partition algorithm returns this minimum tdf along with the reduced graph.

Then the embedding algorithm finds the final mapping using binary search on tdf (Algorithm 1). FindMapping first scales CPU and bandwidth with the tdf parameter T. It then computes the resource demand of a VM as a weighted sum of all of its resource requirements. The weight w_r of a resource r is determined based on the relative scarcity of r. Then it selects VMs in a non-increasing order of their resource requirement. The intuition here is that it is easier to accommodate resource demanding VMs during the earlier stage of embedding.

Algorithm 2: Partition Emulation Request

```
function Partition(G)
    b_if ← Interface bandwidth of a VM
    Sort S by (1) in non-increasing order
    Partition the network using First Fit
    forall the inter-partition link do
        move the link if it minimizes the bandwidth and does not violate resource constraint
    Generate G^R
    Find max. external bandwidth b_max for each partition
    T_min ← b_max / b_if
    return T_min, G^R
```

Algorithm 3: Find Embedding

```
function FindMapping(G^R, T)
    Scale all the resources r' ∈ R' using T
    N^R ← Sort nodes of G^R according to Σ_{∀r∈R} w_r c_n^r
    repeat
        Choose the best node i ∈ N^R, N^R ← N^R − {i}
        Pick the best feasible host h̄
        if There is no feasible host then return false
    until N^R ≠ ∅
    return true
```

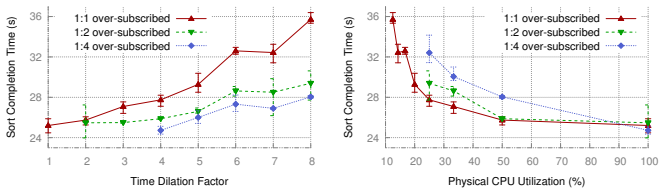
After VM selection, the physical machine for this VM is determined. The algorithm first lists all feasible physical machines that can satisfy its resource requirements. It considers two criteria for this purpose: 1) how many neighbors of the selected VM it contains, and 2) its residual resources. Specifically, it ranks a physical machine higher if it already hosts some neighbors of the VM. This criteria minimizes the allocation of external link bandwidth. On the other hand, selecting a physical machine with minimum residual resource minimizes the fragmentation. Our algorithm selects the physical machine as a weighted sum of them.

VIII. Evaluation

EASE is a work in progress. However, we have implemented some features of EASE in a prototype and demonstrate the impact of time dilation in Section VIII-A and EASEś capability to provide resource guarantee in Section VIII-B.

A. Impact of Time Dilation

1) Setup: In this scenario, we show the impact of time dilation on the completion time of an emulation. We also demonstrate how other load factors in the system affect completion time of an emulation under different tdfs. Our setup consists of four VMs, each with a single virtual CPU, spawned on a physical host with four physical CPU cores.



(a) (b)

Fig. 4. Impact of (a) Time Dilation and (b) CPU Over-subscription on Emulation Completion Time

As a workload we sort 128MB of randomly generated data on these VMs. As the performance metric, we measure the virtual time required to complete the sort. Note that, changing the `tdf` changes the view of CPU speed to the VMs. In order to conduct reproducible experiments, the VMs should have the same view of CPU speed throughout their lifetime. Therefore, when we change the `tdf` we also restrict the maximum allowed physical CPU utilization for the VMs using `cpulimit` [14] tool. `cpulimit` limits the physical CPU utilization of a process by sending `SIGSTOP` and `SIGCONT` signals. This has an impact on the performance of a process as well. To demonstrate `cpulimit`'s impact we run the experiment with different over-subscription ratios for physical to virtual CPUs. A $1 : k$ over-subscription rations means k virtual CPUs are pinned to 1 physical CPU.

2) Results: Fig. 4 presents the results on the impact of `tdf`. Ideally, for a given over-subscription ratio, the virtual time required to complete a job should remain almost the same for different `tdf`s. Also, the best possible sharing scenario, *i.e.*,when each virtual CPU is pinned to a separate physical CPU, should yield the minimum job completion time. However, as we can see from Fig. 4(a), the results are counter-intuitive, *i.e.*,more over-subscription gives faster job completions and for a given over-subscription, the completion time increases with increasing `tdf`. Such increase is due to the impact of `cpulimit`. With increasing `tdf`, we are lowering the maximum physical CPU share cap on the virtual CPUs. As a result, the probability of `cpulimit` sending `SIGSTOP` signal to the virtual CPU is increasing. For example, for a given over-subscription ratio, a `tdf` value of 2 puts 50% cap on the virtual CPUs. However, for the same over-subscription ratio we need to put 25% cap on the virtual CPUs, therefore increasing the probability of sending `SIGSTOP` to the process corresponding to the virtual CPU.

To better understand the combined impact of over-subscription and `tdf`, we represent the results in Fig. 4(b). Fig. 4(b) presents the job completion time against different physical CPU utilization for different over-subscription ratio. Consider a case, when the `tdf` is 2 with an over-subscription ratio of $1 : 1$. In this case, a physical CPU hosts one virtual CPU and allows a maximum of 50% total utilization. If we want to achieve the same level of physical CPU utilization with an over-subscription ratio of $1 : 2$, we have to set `tdf` to 2 and cap the maximum allowed physical CPU for each virtual CPU to 25%. Therefore, there will be higher probability of `cpulimit` interrupting the virtual CPUs. Therefore, for a given physical CPU utilization, higher over-subscription will put lower cap on the utilization of individual virtual CPUs, thus increasing the overhead due to `cpulimit`. For the same reason, we observe a decreased job completion time for higher over-subscriptions.

B. Resource Guarantee for Emulation

1) Setup: We demonstrate the better resource isolation capability of EASE compared to state-of-the art emulation testbeds. We conduct the same experiment as described in Section I. We deploy the same topology with 100Mbps link capacity and play traffic between randomly selected links. We measure the utilization of a fixed link for different levels of background traffic.

2) Results: Fig. 5 shows the results on resource isolation. Compared to Emulab, throughput does not significantly drop when the same experiment runs on EASE.

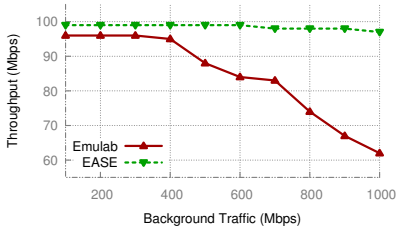


Fig. 5. Resource guarantee in the presence of background traffic

IX. Conclusion

In this paper, we have presented the design and implementation of EASE: a distributed virtualized multi-user testbed for infrastructure emulation. EASE provides features such as performance guarantee, reproducibility and performance isolation. It is worth mentioning that this is a work in progress. Once complete, service provides, network operators and researchers can benefit from EASE to experiment with new services or ideas that require playing with an infrastructure.

As a future work, we plan to improve the time dilation of the emulated network. Currently, we observe some overhead introduced by `cpulimit` when we are capping the maximum CPU utilization for time dilated VMs. We plan to investigate further into this matter. Moreover, we also plan to investigate the impact of time dilation on latency sensitive emulations. We also plan to improve the networking performance by running the emulated network directly on physical machines instead of on time dilated VMs. Finally, we also plan to study the impact of time dilation on non-dilated resources.

Acknowledgment

We thank our shepherd Guillaume Doyen and the anonymous reviewers of CNSM 2016 for their valuable feedback. This work was supported by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

REFERENCES

[1] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 255–270, 2002.

[2] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, no. 0, pp. 5 – 23, 2014.

[3] M. Suñé, L. Bergesio, H. Woesner, T. Rothe, A. Köpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda *et al.*, "Design and implementation of the ofelia fp7 facility: the european openflow testbed," *Computer Networks*, vol. 61, pp. 132–150, 2014.

[4] "Internet2 Research Network Topology and Traffic Matrix," http://www.cs.utexas.edu/~yzhang/research/AbileneTM/.

[5] "What's Behind Network Downtime? Proactive Steps to Reduce Human Error and Improve Availability of Networks," http://www-05.ibm.com/uk/juniper/pdf/200249.pdf.

[6] "Open vswitch," http://openvswitch.org/.

[7] "Quagga routing suite," http://www.nongnu.org/quagga/.

[8] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker, "To infinity and beyond: time warped network emulation," in *Proc.of ACM SOSP*. ACM, 2005, pp. 1–2.

[9] H. W. Lee, D. Thuente, and M. L. Sichitiu, "Integrated simulation and emulation using adaptive time dilation," in *Proc. of ACM SIGSIM PADS*. ACM, 2014, pp. 167–178.

[10] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, "Design and Management of DOT: A Distributed OpenFlow Testbed," in *Proc. of IEEE/IFIP NOMS*, 2014, pp. 1–9.

[11] D. S. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, 1974.

[12] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *Proc. of ACM SIGCOMM*. ACM, 2012, pp. 431–442.

[13] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proc. of IEEE INFOCOM*. IEEE, 2014, pp. 289–297.

[14] "Cpulimit tool," https://github.com/opsengine/cpulimit.

[15] "Bro intrusion detection system," https://www.bro.org/.

[16] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *Proc. of ACM CCS*. ACM, 2004, pp. 2–11.

[17] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *Proc. of PAM'13*. Springer, 2013, pp. 31–41.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[19] "Reproducable network research." [Online]. Available: https://reproducingnetworkresearch.wordpress.com/2014/06/03/cs-244-14-bro-network-intrusion-detection-system-performance-analysis/

[20] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proc. of HotNets 2010*. ACM.

[21] P. Wette, M. Draxler, and A. Schwabe, "Maxinet: distributed emulation of software-defined networks," in *Proc. of IFIP Networking*, 2014, pp. 1–9.

[22] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "Estinet openflow network simulator and emulator," *Communications Magazine, IEEE*, vol. 51, no. 9, pp. 110–117, 2013.