Proclamation

I declare that this thesis was made by myself with assistance of my supervisor. All parts taken over word by word from literature or other publications are referenced. I approve publishing this thesis or any part of it with referencing author of original text.

In Prague at 2014-11-11 ....

Abstract

Contents

Part I

Theoretical

Virtualization

Virtualization is, in my opinion, the most important technology in data centers, because it caused significant progress in this field. It is not technology itself, so it should rather be called model than technology.

Definition of virtualization as stated in [2] says that ”virtualization is a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications or end users interact with those resources. The concept of virtualization is very broad and can be applied to devices, servers, operating systems, applications and even networks.” This definition gives description of the virtualization and can be applied to any type of virtualization.

The most common approach is virtualization of computers, because it is the oldest one and most widely used there days. It started in 1960s with mainframes as an attempt to employ resource sharing and this idea is still alive in current time. Virtual computer is logical representation of computer in software. [2] Virtual computers are usually called virtual machines (VM) and physical machine hosting VMs is called hypervisor. Rigorous term for physical hosting machine is host and hypervisor is software performing the virtualization, but word hypervisor is widely used in technical text for machine as well. It is possible and very advantageous to host many virtual machines on single physical computer, because it brings technical and economical benefits. Decoupling computer and it's software from hardware is important advantage, because it brings additional level of abstraction and gives ability to shift virtual machines between hypervisors. Economical benefit is quite obvious, since it is not necessary to buy single physical server for every service and electricity saving are also appreciable.

Another important type of virtualization is virtualization of networks. It is usually used together with computer virtualization, since it gives an occasion to separate network devices from network itself. Physical machines are not as flexible as VMs are, so plugging them into virtual network is not as beneficial as VMs, because there are still physical network cables, that can be hardly virtualized. There is a hot topic called Software Defined Networking (SDN) having potential to provide virtualization info physical network infrastructure, thus it may be good idea to integrate physical machines into virtual network as well.

Storage virtualization should also be taken into account, because it provides abstraction of the storage. Typical unvirtualized storage uses some physical device for storing data and metadata, but this approach is not flexible enough since it is usually limited to just one physical machine or group of machines connected to shared storage. It is necessary to find any method of storage virtualization, which would be able to provide any storage to any physical of virtual computer.

Service virtualization, memory virtualization, I/O virtualization or database virtualization are another types of virtualization. It is not necessary to mention all the types of virtualization because it is possible to virtualize almost everything and emerging of new types is quite probable.

Term virtualization is going to be used in further text as computer virtualization, another types of virtualization will always be denoted.

1.1 Types of virtualization

There are three different virtualization types and they vary by method used t add virtualization layer between guests and physical hardware. It it not possible to easily choose better or worse virtualization types, because it depends on intended usage, character of computing tasks and required operating system.

Architectures of computers, especially x86, are designed to run on physical devices, thus is in not easy to virtualize them. Access to hardware is controlled by priority levels called rings. Lowest priority is used by userspace applications and highest priority (ring 0) is reserved for operating system. It is necessary to insert virtualization layer between operating system and hardware, but there is not any ring with higher priority than operating system uses. This problem needs to be solved and it is not only one challenge. There are sensitive instructions incompatible with virtualization, because they use different semantics when they are not run in ring 0, as mentioned in [12].

1.1.1 Paravirtualization

Paravirtualization is type of virtualization with necessity of modifications in guest kernel. Modifications of kernel are necessary, because operating system uses non-virtualizable instructions that are trying to gain direct access to the hardware. These instruction need to be replaced with hypercalls that communicate directly with virtualization layer of hypervisor. [12] It is obvious, that guest operating system knows it is running virtualized.

Biggest advantage of paravirtualization is lower overhead compared to other types, because it is not necessary to translate instructions before running. However this advantages becomes less significant during time since there are already available processors optimized to run hardware assisted virtualization with less overhead. Main drawback of this type of virtualization is need for modifications done at an operating system, which is not always possible or allowed. Running modified OS also brings additional administration and thus additional cost.

It is possible to take a different look at paravirtualization and do not try to create entire virtual machine, but use operating system-level virtualization, where kernel allows to run multiple userspaces. These userspaces are called containers and therefore this approach is sometimes called container virtualization. It does not provide entire isolated virtual machine, but allows to run software packed in container. It is advantageous because there is almost none overhead in running software from container while maintaining sufficient level of container isolation. Container virtualization is applicable for situation, where whole virtual machine is not needed and then brings huge performance improvements since operating system layer is shared. Some says, that containers are going to bring next revolution info virtualization. For example Dustin Kirkland, Cloud Solutions Product Manager at Canonical wrote: "Linux containers, repositories of popular base images, snapshots using modern copy-on-write filesystem features. Brilliant, yet so simple. Docker.io for the win!" [13]. I think, that container virtualization may brings compelling advantages and I also like using it, but it is not suitable for every situation. It is still technically kind of paravirtualization and thus it is limited to provide only additional layer on host's operating system.

1.1.2 Full virtualization

Virtualization type capable or running unmodified operating system is called full virtualization. It utilizes runtime translation, which captures non-virtualizable commands and emulates them using hypervisor virtualization layer. Virtualizable instructions are executed directly on the hardware. Modification of "problematic" calls is carried by the hypervisor and it is the main difference compared with paravirtualization.

Most important benefit of full virtualization is it's ability to run guest operating system without any changes, so guest OS is not aware of being virtualized. This makes guest operating system fully abstracted from underlaying hardware, it is possible to multiple different operating system on single host and provides simple migration from physical to virtual machine. Drawback of this type is overhead caused by catching and translating non-virtualizable calls.

1.1.3 Hardware assisted virtualization

Full virtualization has significant overhead caused by binary translation, so CPU vendors introduced technologies capable of inserting virtualization layer between ring 0 and physical hardware. It speeds-up trap of privileged and sensitive calls to the hypervisor and it is not necessary to perform binary translation of to modificate kernel of guest operating system.

Benefit of this type is quite obvious, because it lowers virtualization overhead and thus provides better performance compared with full virtualization together with elimination of need for guest kernel modifications compared with paravirtualization. It is necessary to have a support in host's CPU is primary drawback of this type, but there is support in almost every processor in current marker.

Running unmodified guest operating system leaves all necessary translations of instructions on hypervisor layer, so I would be good to to introduce small changes to guest's operating system, which will reduce work left for the hypervisor but also do not need any significant changes in guest's kernel. This approach is called hybrid virtualization and it is subset of hardware assisted virtualization. Installation of additional drivers is required, but it is not necessary to apply any changes on whole kernel. These drivers are aware of virtualization and use virtualization layer directly without any translations made by the hypervisor. This method increases driver's IOPS and therefore it is usually used for virtualized network cards and storages. Driver able to deliver hybrid virtualization is *virtio* for KVM, Xen call it *paravirtualized device drivers* and VMWare *Guest Tools*.

1.1.4 Summary on types of virtualization

There were presented some virtualization general virtualization types and their pros and cons. There is not any universal virtualization type suitable for all use cases, thus is is always possible to decide on planned usage. It also depends whether it is required to run different kernel on single physical host or it is sufficient to share one kernel for all containers. Differences are compared in table 1.1.1.

We can divide types into two groups:

- One group provides guests with full virtual machine, every VM uses it's own isolated kernel and VMs are fulll or almost fully decoupled from hardware. Full, hardware assisted and hybrid virtualization belongs to this group.
- Members of second group are containers and paravirtualization. This group is specific by lightweight containers and host kernel shared by all running containers.

Virtualization is massively used even by czech IT companies. First group is used for example by *Wedos* for their virtual server hosting and related services. Second group is uses by *Seznam.cz* and they use LXC for web serves as well as for Hadoop cluster.

Table 1.1.1: Comparison of virtualization types

Type	**method**	**guest modif.**	**usage**
Paravirtualization	hypercalls by guest kernel	yes	same workloads and same OS
Full	translation of instructions	no	when full abstraction is needed
Hardware assisted	translation with help of hardware	no	same as full, but with compatible CPU
Hybrid	translations and driver changes	driver only	when possible to install additional drives

1.2 Advantages of virtualization

Most important advantages is decoupling software from physical hardware, at least in my opinion. It is possible to migrate virtual machines with running services between physical hosts without significant impact on service behavior. This brings amazing opportunity to adapt service environment on demand and scale the service.

It is possible to perform any hardware and software upgrades, because all running services may be temporarily migrated to other physical host. Virtual machines are much more easier to deploy than physical ones. It takes only a few seconds to create and run VM compared to at least hours to deploy physical machine. Deploy of virtual machine do not have to be performed by persons, because it is possible to employ an orchestration and scale up the service (add virtual machines) automatically. Reset of virtual machine is actually just software instruction in hypervisor, so it may by done remotely with ease.

Geographical backups or failover is much more easier to accomplish with virtualization approach. You can rent virtual machine from provider in foreign country and start your services in a few moments. It is huge simplification compared with running physical machine at foreign data center.

Virtualization brings also some economical and environmental advantages. Economical advantages are quite obvious, because it is no longer necessary to buy physical servers. Non-virtualizational approach requires one physical machine for every running server, but it is not longer necessary with virtualization. It is possible to run many virtual servers or containers on single physical machine. It is also possible to move even to the higher level of CAPEX cutting and rent virtual machine from provider and absolutely eliminate need for running any server machine. Renting virtual server increases OPEX, but they are more flexible and easier to control. Electrical consumption should also be taken into account, because single physical machine, even under higher load, will definitely consume less power compared with two or more similar machines.

I asked Petr Hodač, technical manager at SiliconHill and he stated, that they managed to reduce electricity consumption of whole server room by 19% iter alia due to deployment of virtualization. It produced also additional saving, because they need less UPS batteries and less cooling capacity, but savings on cooling are not included in mentioned savings.

However there are some drawback too. Failure of physical host causes failure of all virtual machines or containers running on this host. It is kind of single point of failure, but we can fight it with duplication of service nodes between different hosts, datacenters, providers or continents. Another disadvantage is hidden in additional virtualization level, since it is necessary to take care of hypervisors. I is not a real disadvantage, since traditional non-virtualized approach needs to take a care of many virtual machines.

Deployment of virtualization should always be well planned, because it can bring many amazing advantages, but it is also able to cause a disaster in case of poor system design or amateurish administration.

Cloud computing

It is possible find many services called ”cloud based” and it is important to agree on accurate definition of these services. It is quite clear, that cloud based service will use principle of cloud computing. Definition of cloud computing by NIST says, that ”Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) than can be rapidly provisioned and released with minimal management effort or service provide interaction.” [15]. This definition clarifies what cloud computing is, but says nothing about parameters and used technologies.

I think, that it would be more convenient to start definition from lower levels, which provides elementary parts, and get to the cloud service afterwards. This definition gives different look at cloud computing than NISTs, but it uses same conditions and therefore results are basically same. It focuses on currently used principles, which may change during time, so it may not be valid after some time, but it provides more technical overview on operation of cloud services.

Cloud computing services are nowadays heavily dependent on virtualization, because it allows to replace physical machines with virtual machines (VMs) or containers and brings a lot more flexibility than physical machine can ever provide.

Basic part of cloud computing system is virtual machine. Physical machine can also be part of the cloud system, but it is not able to deliver required rapid provisioning and it is not possible to deploy physical machine without service provider interaction. Virtual machine is elemental resource and also use some additional resources. These resources can be for example networking, which is used for interconnection between VMs as well as for reaching customers, storage used for system internal or customer data. It is important do employ some configuration management and orchestration, because it is able to deliver rapid provisioning of virtual machines and minimizes effort required for administration.

Virtual machines together provides the service, which is exposed to users via any kind of network. It doesn't matter whether customers access the service directly at virtual machines or via a proxy, but hiding worker VMs brings additional flexibility for migration and scalability.

Difference between cloud computing and bare virtualization is intelligence included in cloud, because it may be controlled automatically according to events or monitoring observed at cloud system. It is common to supply customers with configuration interface, which allows them to tune service parameters and provides user-friendly interface for administration. Bare virtualization does not offer any intelligence, even if it is equipped with shiny user interfaces with opportunity to scale virtual machines up or down, because all change performed manually.

Cloud computing is kind of hype these days, so it is often used just for marketing purposes and thus it is recommended to perform service analysis and do no absolutely trust every buzzword used in specification.

2.1 Deployment models

There are three scenarios possible for deploying cloud solutions. Models differs by ownership and subject responsible by administration of the system. Right solution depends on expected load, available budget as well as on expected classification of data. Public model and private model are mutually contradictory and last model called hybrid is combination of first two mentioned.

2.1.1 Private

Private model defines cloud environment build exclusively for single subject. Typical scenario is to build private cloud in datacenter owner by the subject, but it is not strictly required. There is common misunderstanding of term private, because it means private usage of cloud resources and not private ownership of cloud infrastructure. Private cloud may be leased from third-party provider and it also can be running on third-party hardware.

Running private cloud gives an advantage in elimination of any inter-tenant isolation problems and it is possible to adapt configuration to fit owner needs. There is a law, which forces sensitive data to be stored internally and with limited access, so it is necessary to build private or hybrid cloud for this kind of usage. It is not clear how to handle clouds and especially storages with law, because it this topic is wide and it is not possible to define rigid rules. There is currently running case with US judge ordering Microsort to provide data stored in Ireland. [17]

Drawback of private cloud is higher initial cost and probably also higher operational costs, but it depends on expected usage.

2.1.2 Public

Public cloud deployment model is based on resource sharing between the tenants. There is usually one subject called cloud provider and many customers (tenants) and these tenants buy resources and rights to use them. Resources are usually charged according to it's usage.

Billing per resource usage is called pay-per-use and it is interesting method of shifting costs between initial and operational. There are usually plans with various CPU, memory and storage options and final cost depends on real usage of resource. Pricing plan based on pay-per-use is favourable to services with low load with occasional peaks. Service under constant high load is not well suited for this payment model, because it does not bring any benefits. It is also to run scalable application, since unscalable will not be able to

Infrastructure is not dedicated and it is shared between tenants. Resources are shared, but must be strictly isolated, because it is unacceptable to allow any interference between tenants, unless they make an explicit request to allow it.

It is common to provide services with flexible parameters, for example Amazon calls it EC2 - Elastic Compute Cloud. Elasticity of provided services allows tenant to use more resources when needed and fall back to usual amount.

2.1.3 Hybrid

Hybrid cloud is model utilizing both, public and private, previous mentioned models. The goal is to combine advantages of both model and eliminate drawbacks. Public cloud is usually more cost effective, but may not be able to meet the security requirements and on the other hand private cloud can be designed to comply with users requests, but it is expensive. Hybrid designed solution can use private cloud part for confidential data and public cloud for less sensitive ones.

It is also possible to utilize cloud bursting in which system runs in private cloud and delegates part of load into public cloud. Lets describe it with application for collecting votes - sensitive part responsible for counting votes and generating results report will run in private cloud and public report will be saved to and served from infrastructure of public cloud. High level of security of counting votes is guaranteed and application is also able to deliver results to many subscribers as it can scale up into public cloud.

2.2 Service models

Purpose of cloud computing is to deliver the service and provide customers with tools to manage this service. Service models differs by level of control provided to customers and thus with areas of responsibility. I am going to call border between responsibility of customer and responsibility of provider as responsibility border. Responsibility borders according to service models are depicted at figure 2.2.1.

Some of service models leave almost all control of service and responsibility at provider side and other supplies customer with more control. It is necessary to select right service model according to expected service usage and required control level.

2.2.1 Infrastructure as a Service

IaaS is model with the most of configuration tasks left at customer's side. Customer is responsible for virtual machines and it's services, so it gives much more flexibility than other models and it is well-suited for services with extraordinary requirements.

Customer manages virtual machines as well as running services, so provider is responsible only for virtualization and underneath layers. It is even possible to run custom operating system, but provider usually offers prepared images with different operating systems. Prepared OS images are tested and modified to run well in cloud environment. There can be installed hybrid virtualization drivers, kernel tweaked to run virtualized and it is also good idea to remove useless drivers and software.

This model is good choice if special configuration is needed, but service deployment is more difficult because some expertise is required. IaaS can be used if customer require additional level of security, because virtual machine can use crypted volume and make data unreadable for provider. Unfortunately it is still possible for provider to acquire confidential from other sources, for example from memory, but it is much complicated to perform this.

Typical example of IaaS is Amazon Web Services and Active24's "Virtuální Privátní servery".

2.2.2 Platform as a Service

Border of responsibility of PaaS is located two layers higher compared to IaaS. Service provider is responsible for platform and all underlaying layers, thus provider takes care of same layers as in IaaS plus operating system and platform. Leaving operating system maintenance on provider's side may be beneficial, because provider can adjust operating system for virtualization and takes care about software updates.

Provider usually manages a lot of operating systems for many customers, so this updating and maintenance tasks may be automatized or executed in batch. Sharing operating system layer between customers with preserving adequate level of isolation can save many resource and make operating system administration even easier.

Customer using service according to this model runs his own software and does not take care of any lower layers. It is not necessary to do any administration tasks and more effort can be given to application development.

This model is well-suited for running applications without any special requirements. It makes service deployment faster and easier, but is more limited by used platform. Typical example is project Evia and Microsoft Azure.

2.2.3 Software as a Service

SaaS is model with none of administration tasks left at customer's responsibility. Software is hosted and maintained by service provider and customer is using the service. Service is accessed remotely via network (usually Internet). It is common mistake to say, that service is accessed as Web service, because any remote access can be used.

This model is right solution for customers looking for service without hassling with it's administration.

2.3 Networking

Networking is essential part of cloud computing because it is not be possible to access any services without networking. Every service in cloud computing system is accessed via network. Network is usually also used for communication between virtual machines, migrations, storage access and for many other tasks.

First part of networking to think about is physical layer. Various Ethernet versions are used for physical layer in cloud data centers. There are many versions with different link bandwidth and wiring, but 1G and 10G with twisted pairs or optical fibers is used most widely. There were 100M called Fast Ethernet but it does not make sense to use this for server link today, because of it's limited bandwidth and almost similar price compared to 1G.

It is common to insert two independent NICs into every server and connect them into independent ToR switches, because it improves fault-tolerance. There is usually one more NIC for remote management and some additional cards if SAN is used. Remote management can be connected using detached cable or shared with any of network cards. Separate cable brings more flexibility and fault-tolerance and shared cable reduces cabling effort and thus simplifies maintenance and improves cooling effectiveness. Both solutions are used.

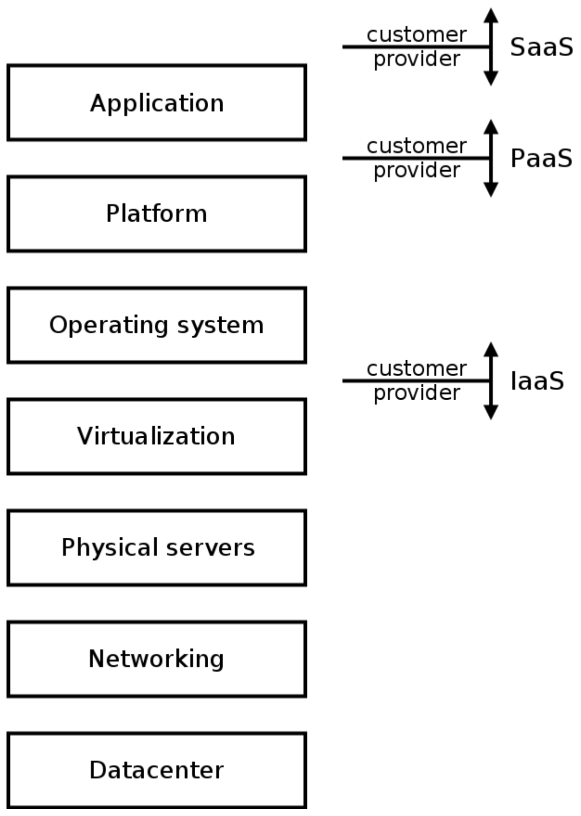


Figure 2.2.1: Service model responsibility

About 40 servers fits into traditional rack and these serverl need to be connected to network infrastructure. There are different topologies, but most common is variant of two or three tier hierarchical structure. [11] There is also approach called ”fabric” which implies non-blocking every-to-every mesh connection between switches. However fabric technologies are proprietary and limited to vendor.

Every rack contains about 40 servers and these servers are connected to switch called ToR. This switch is located in the rack and acts as access layer for servers. Servers are connected to at least two ToRs if additional fault-tolerance is needed. Upper network topology layers depends on data center size and scaling requirements. There can be distribution and core layer, collapsed core or some kind of fabric.

Physical topology must be adjusted to spread network layers between two or more datacenter networks. It is obviously not possible to spread physical layer, but data link layer and upper layers are possible to spread.

Another view on network topology is at network layer. Internet is based on TCP/IP so it is necessary to use this protocol family and assign IP addresses to servers, virtual machines and other network elements. There are two different versions of IP protocol:

version 4 is the older one, with 32 bit address space. This version is still used more than version 6 even though it’s address space is depleted and new version exists for more than 15 years.

version 6 is the ”new” one, uses 128 bit address space and different headers, thus it is incompatible with version 4.

Moder data center must provide both versions of IP protocol, because supporting only one versions is a huge limitation and can not be accepted for new services deployment. However there is a problem with obtaining IPv4 addresses, because available pool had already been depleted and all available addreses had been divided between RIRs. It is beneficial to make efforts to employ IPv6 protocol as primary one and try to limit the amount of required IPv4 addresses.

There are different ways how to use both versions concurently:

- Dual-stack is the simpliest and probably the most used solution. Each interface gets at least one IPv4 and one IPv6 address. Use of both versions causes additional maintenance effort because it is necessary to take care of two separate L3 networks.

- Tunelling IPv6 via existing IPv4 infrastructure with technologies like 6to4, 6rd or ISATAP is another way. This solution can be used for IPv6 deployment in networks with working IPv4, because it is used for transmission of all packets. Tunelling is usually focused on deployment in access networks, but deployment in data center network is also applicable, as described in [19].

- Translating IPv4 addresses into part of IPv6 addressing space is different approach than previous mentinoned, because it operates on IPv6-only networks. This technique does not require every box to have assigned an IPv4 address and thus is good for saving address space. Hovewer address translations may not be suitable for data center usage, because it does not preserve original IP addresses and makes customer tracking almost impossible. Further information can be found in [1].

It is beneficial to deploy protocol IPv6 as primary one in my opinion, because it will become more and more needed during time. Hardware on current market usually support protocol IPv6 at least partialy, but there are still some hidden pitfalls. There may be problem for example with server's remote management, because it may not support IPv6 and thus is totally unusable on IPv6-only network. None of servers I have used for practical part of this thesis have support for IPv6 on IPMI. However there is currently only small demand and IPv6 deployment does not bring any direct profit. Even thought there are many problems and advantages are quite hidden, it is not possible to ignore this protocol and stay with old IPv4.

Virtual machine migration must be taken into account durring addressing schema design, because it plays crucial role in data center operation. Migrations are performed between hypervisors, i.e. physical servers, and these servers may be located in different racks, halls or even in different data centers. It is usually required to preserve IP address of virtual machine during migration process and thus adressing schema must be prepared to move single IP address around almost whole data center nwithout any significant configuration changes.

First solution for unlimited migrations while preserving IP address is L2 sharing between hypervisors. Data link layer is shared between all hypervisors and then virtual machine is located in same L2 network before and after migration. This solution does not scale well since it is not recomended to place more than a few hundreds of hosts into single L2 domain, so it may be necessary to divide single big L2 into many smaller networks. It is quite easy to employ this solution for hypervisors in same rack, but it is more difficult with more distant servers and even more when servers are located at different data centers.

Another way how to accomplish unlimited VM migrations is to use routinl for machine connectivity. This method uses temporary and fixed IP addresses. Hypervisor do not have to be in same L3 network and there is higher variability in addresses assigned to virtual machines. Temporary address is assigned according to VM's location and it changes during migration. Fixed address is routed to VM and this address does not change, because changes are made only in routing tables. Any routing protocol can be usel, e.g. OSPF, to provide correct routing of fixed address destination virtual machine. Is is neccesary to insert one record in routing table for every virtual machine, because \32 routes are being advertised. Huge routing table may be problem for data centers with many virtual machines. Higher layer is used to get more flexibility than lower level can offer. Main drawback of this solution is additinal complexity caused by routing and longer address swap, because it takes some time to propagate routing to new temporary address. It is not easy to perform live migration, because it is neccesary to change IP address of VM's interface and thus open sessions will terminate.

2.3.1 Overlays

Virtualization is used heavily these days, therefore it is necessary provide networking solution with at least same flexibility as virtualization offers. Multitenancy, VM migrations, fast reconfiguration and rapid deployment are most missing features of physical networks these days. It is currently possible to migrate virtual machines without service interruption, but there is still not clear solution how to perform migration across whole data center with maintaining IP address. Overlay networking is one of proposed solutions and it is supposed to bring additional abstraction layer capable of decoupling network from physical hardware. Technologies capable to build overlay network are VXLAN, STT and NVGRE.

Data center network need to be robust enough so parallel paths are used to provide redundancy and avoid outage caused by single link failure. It is necessary to avoid loops on L2 network because there are loops caused by redundant paths and L2 network does use anything like TTL field. Spanning Tree Protocol (STP) can be used for avoiding loops in L2 networks, but there are two mayor problems. First is need to adjust STP if VLANs are used because it is necessary to build special tree for each VLAN. Second problem with using STP is utilization of parallel links. STP keep only one of parallel link running and the others are disables to avoid topology loops. It is not optimal solution because links utilization is low and it is impossible to increase connection bandwidth by adding parallel links. Upgrading to higher link speed is limited and does not make economical sense. 10G cards are becoming affordable, but 40G cards are still expensive.

Servers housed in rack are usually connected to switch called ToR and this switch is learning their MAC addresses. Common ToR switch have 24 or 48 ports so it should be able to learn addresses of these serves. However virtualization techniques let us to run many virtual machines on single physical server so number of MAC addresses can increase significantly. Amount of address may grow even more because each virtual machine can have more than one interface and ToR switch must be capable of learning all addresses.

Public cloud solutions tend to serve many tenants and it is necessary to avoid unwanted interaction between them. It is quite easy to guarantee this on virtualization layer but much harder to accomplish on network layer. Tenant isolation can be performed on Layer 3 or Layer 2. VLANs are often used for isolation on Layer 2, but this solution suffer from insufficient scaling issues because only 12 bit VLAN identifier is used and it provides only 4096 different tags. This might be enough for smaller solution but it is not sufficient for huge cloud system. Each physical server can host up to 100 virtual machines/containers[1] owned by different tenants so 1 server can consume about 100 VLAN tags. All available tags can be depleted by 40 high performance servers which can fit in just one rack. Isolation at Layer 3 is does not provide sufficient scaling as well. It is necessary to provide unlimited migration facility between hypervisors in different racks and sometimes is required to spread Layer 2 between all tenant's virtual machines. Layer 3 isolation is not capable of this.

VXLAN

Virtual eXtensible Local Area Network is an overlay scheme with multitenancy and domain isolation. It is defined on Layer 3 and uses encapsulation as tunneling mechanism.

Most important think is encapsulation since it provides VXLAN domain isolation and defines overlay network. Block called VTEP is responsible for encapsulation and tunnel organization. It analyzes every packet received from VM and prepend outer header with label. This label is called VXLAN Network Identifier (VNI) and it is used to isolate domains so virtual machines in different domains are not allowed to communicate directly with each other. Encapsulated packet is send to destination VTEP as UDP packet. Destination VTEP unpacks packet, check whether there is any virtual machine in VXLAN domain and deliver this frame. NVE is another term for VTEP and it was introduced in [7] as part of general network virtualization framework.

It is necessary for VTEP to be able to find destination VTEP for every encapsulated packet. This can be solved by data plane learning during forwarding as specified in [8] or by acquiring this information from orchestrator. Getting information from orchestrator in my opinion much better because it avoids additional actions. Some kind of orchestrator or at least information system must be present in every data center and this system already knows location of all virtual machines as well as addresses of VTEPs. I think that it does not make sense to perform learning during forwarding because required informations are already saved in orchestrator. Orchestrator can directly distribute forwarding rules to all VTEPs or VTEP can ask orchestrator on-demand via any kind of API. Overlay unicast traffic can be forwarded directly to destination VTEP without any additional learning or even flooding.

There is traffic called BUM - Broadcast, Unknown unicast and Multicast which not easy to handle by VXLAN. This kind of traffic needs to be delivered to more than one host in a single VXLAN domain and thus it is necessary to send encapsulated packet to many VTEPs at the same time. Multicast should be used ad

[1]Server node2.brg/vpsfree.cz is currently running 117 VPSs with hardware: Supermicro X9DR3-F, 2x Xeon E5 2630Lv2, 256GB DDR3, 8x 2TB WD2002FAEX, 2x Intel DC S3700 200GB

described in [8], but it needs mapping between VXLAN VNI and multicast address. This mapping should be managed by orchestrated by management layer. It would be beneficial to have technology for delivering BUM traffic without need of multicast because it brings additional complexity and it is not allowed in global Internet. Unknown unicast can be mitigated with getting information about addresses from orchestrator as described in previous paragraph because the orchestrator knows all addresses and there will no longer exists any unknown traffic. Sending encapsulated broadcast and multicast to many VTEPs can be achieved by multicast in underlay network or any advanced delivery methods can be uses. It is possible to send this traffic as unicast between source VTEP and every other VTEP. However this solution is suboptimal thanks to packet duplication and higher network bandwidth usage. Only one advantages is that multicast in underlay network is not required. It is also possible to select on node as ”router” for encapsulated VXLAN traffic between VTEPs but it is technically similar to building multicast tree and thus it might be easier to deploy multicast in underlay. There is proprietary technology called IBM DOVE with is very similar to VXLAN but does not require multicast.

I think that VXLAN is quite promising technology for network virtualization in data center. It brings much more flexibility that traditional VLAN approach and it can be called as en evolution of VLAN. Principle of overlay network is building virtual network on top of physical infrastructure. Benefits were described in previous articles and main drawback is lack of cooperation between overlay and underlay network. Multicast might be also quite problematical to arrange.

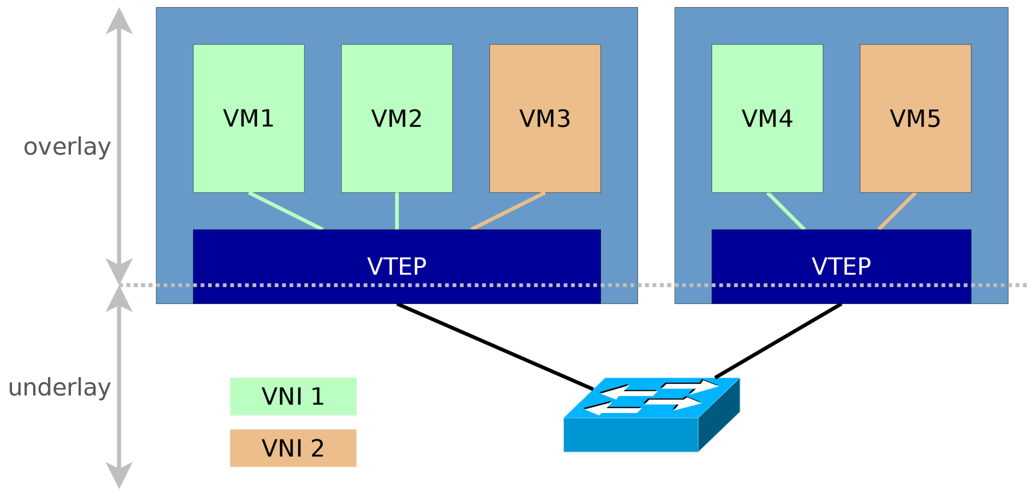


Figure 2.3.2: Model of VXLAN topology

NVGRE

Network virtualization using GRE is another overlay technology for multi-tenant data centers. It is very similar to previously mentioned VXLAN because it uses same topology scheme with different encapsulation mechanism. I am not going to provide detailed description of protocol as with VXLAN but main differences will be mentioned. Further information and current draft version can be found in [9] [6] [4].

The biggest difference is encapsulation mechanism. VXLAN uses own approach but NVGRE uses GRE. Using quite old encapsulation standard may be beneficial because some boxes already support it and there is no need to make significant changes to physical infrastructure. Details about encapsulation mechanism can be found in [6] and [4] describes header extensions used to carry VSID. VSID is identifier for virtual subnet isolation, it is analogue of VNI used by VXLAN with same length of 24 b.

Outer header of packet with encapsulated payload is sent to destination NVE thus usage of ECMP may be suboptimal. There may be lack of entropy in outer header because destination address is same for all virtual machines residing at same NVE. This problem should be solved in ECMP hashing procedure by integrating VSID into sources for hash generation.

Multicast and broadcast traffic within overlay network is handled using multicast in underlay network. There is also defined N-Way unicast which do not depend on multicast: "In N-Way unicast, the sender NVE would send one encapsulated packet to every NVE in the virtual subnet. The sender NVE can encapsulate and send the packet as described in the Unicast Traffic Section 4.3. This alleviates the need for multicast support in the physical network." [9] However this solution is suboptimal because there is unwanted packet duplication and thus it is better to deploy multicast and use it as carrier mechanism.

Definition of NVGRE [9] is still labeled as draft, last version was published on 2014-11-05 and new updates are expected. Proposed draft is simple and there are still mayor problems waiting to be solved. For example there is not any method how to distribute locations of addresses within overlay network. Document [9] says: "This information can be provisioned via a management plane, or obtained via a combination of control plane distribution or data plane learning approaches. This document assumes that the location information, including VSID, is available to the NVGRE endpoint." It is obvious that this need to be solved before deployment in production use.

STT

Last but not least technology used for building overlay network is Stateless Transport Tunneling (STT). It is designed to meet common requirements as allow overlapping of tenant's address space, decouple virtual network from physical infrastructure and allow unlimited virtual machine migration.

Basic principle is still same - some box (usually called NVE) encapsulates packets from overlay network and send it through underlay network to other NVEs. However STT introduces completely new encapsulation method. TCP-like header is used as an encapsulation header but there is no three-way handshake or sessions because packets are processed different way. Header is used only as a storage for metadata about encapsulation. Field called Context Identifier is assigned to every flow and it is used as a generalized form of virtual network identifier. [3] It is beneficial to use this generalization because there is space for future services. Space reserved for Context Identifier is 64 bits long so there is really enormouns amount of combinations available.

It is important for every overlay technology to support ECMP because efficient flow distribution between multiple paths can be used for underlay network in data center. First important requirement is to route each packet belonging to single flow same way and it is accomplished by using same ports and addresses for these packets.

Second requirement is to provide enough of entropy for uniform flow distribution. Packet's source port is function of inner header and thus it provides entropy data for ECMP mechanism.

Using almost standard TCP segmentation for encapsulation is advantageous because it may bring significant performance improvement. Segmentation offloading is heavily used these days so it can be used to speed-up encapsulation process. The most important advantage of STT is providing new functions and using of existent hardware techniques.

However I can see some problems with deploying STT. First and the most important is changing meaning of TCP header field since this will probably cause problems in middle boxes. It will be necessary to adjust configuration of state firewalls to allow STT because TCP headers are expected to behave different than is is used. Defining document [3] is still in draft version and it is already expired. Last version is #06 at time of writing this paragraph (2014-13-11) and this version expired on 2014-10-17. It is possible that this technology will be used in future but I do not think it is going to be used as overlay technology very much these days.

2.3.2 Hop-by-hop network virtualization

It is possible to use new technologies, e.g. SDN, and build different kind of virtual network called hop-by-hop. Hop-by-hop virtual network is totally different than previously described overlay networking since it does not use any encapsulation and data path is established by joining independent links between hops.

Hop is responsible only for forwarding data unit to next hop and whole flow is directed by a controller. Controller is software appliance responsible for communication with physical boxes, distributing routes and analyzing packets received from forwarding plane. It is usually tightly collaborating with orchestrator.

There is different perspective on network since control plane is separated from forwarding plane so physical devices are used only for fast packet transfers and data plane is responsible for network control. Every decision is performed in controller or orchestrator and propagated to forwarding plane through data plane. It is obvious that the orchestrator should not be physically centralized because it would create single point of failure so it is better to use any distributed solution.

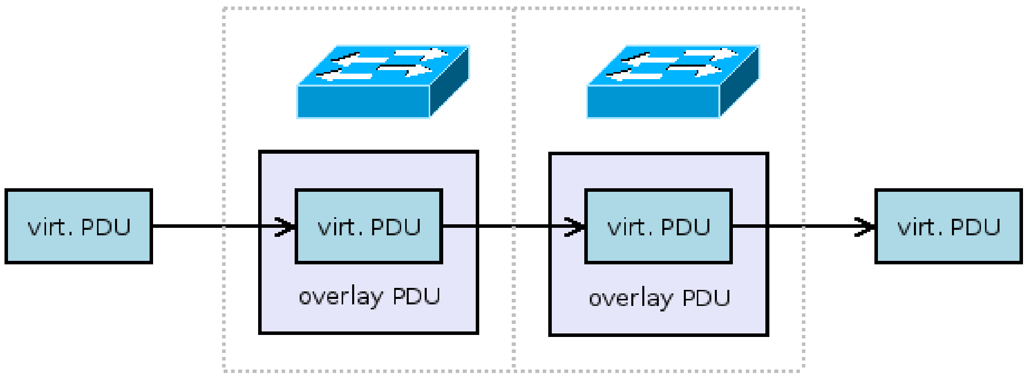


Figure 2.3.3: Overlay virtual network

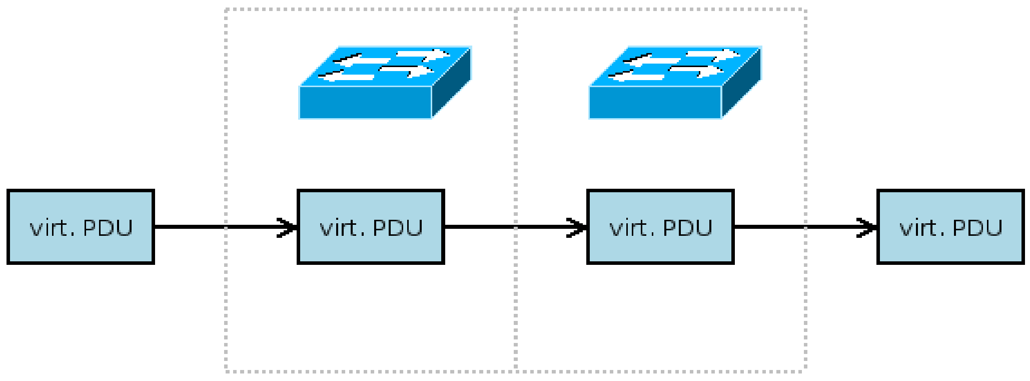


Figure 2.3.4: Hop-by-hop virtual network

2.3.3 Load balancing and high availability

Load balancing is an essential part of service operation because there it is required to improve scalability and availability better than single machine approach can ever achieve. It quite common in the past that one service had been served just by one machine. However this solution is suboptimal since it is totally unscalable and it is impossible to provide high-availability solution.

It is necessary to employ service load balancing because load and service demand is still increasing and only properly designed load balancing solution can meet all of the requirements. Common requirements are

- low latency - request should be processed without any significant delay
- high availability - service should stay up during partial infrastructure failure
- scalability - infrastructure should be ready to increase resource when the load is enormous

There are many different ways how to deploy load balancing and they differ by flexibility and functions available. We can be say that load balancing performed at higher levels is more flexible, but the best solution are combination between two or more technologies on different layers. Appropriate solution depends also on access method because there are different balancing possibilities for example for HTTP API, remote terminal service and video streaming service. Method presented in following text will be general as well as access method specific, but right use case will be always mentioned.

Load balancing is closely related with scaling. There are two types scaling - scaling-up and scaling-out. Scaling-up is accomplished by using more powerful resources, e.g. using interfaces with higher line rate or upgrading server. It is easier, achievable faster and does not require load balancing, but it is quite easy to reach limit of scaling-up. 1G Ethernet NICs are very common, 10G are a bit more expensive but still possible to buy and 100G are very rare and extra expensive. It is also necessary to take economical aspect into account since performance improvement and price function are not equally steep. Scaling-out is other possible scaling schema and it is accomplished by adding many parallel workers with common capacity. This approach is more favourable from economical view because performance growth is almost linear and technical benefit lies in redundancy. However it is necessary to use load balancing to distribute workload across nodes. I think that best scaling solution lies somewhere between so I would recommend to slowly scale-up and use scale-out for massive increase of performance.I think that best scaling solution lies somewhere between so I would recommend to slowly scale-up and use scale-out for massive increase of performance.

Load balancing methods can be divided into two group whether they provide session persistence or not. Session persistence mean that one client is always routed to same computing node. It is required if there is a client's information, called session, available only on this computing node and session would be lost in case of redirecting to another node. Application can be designed with taking load balancing into account and thus it does not require session persistence. However session persistence is usually needed for load balancing of services designed without load balancing capabilities.

DNS based approach

DNS load balancing is first possible solution because it is take place before establishing session. It is relative easy to deploy and application redesign may not be necessary. Basic implementation can be round-robin DNS which is carried out by assigning many AAAA or A records for service host name. Client selects one record after resolving host name and use it thus basically performs load balancing already at user's device. This method is really simple but lacks any advanced management options. First problem is with high availability because it is not possible to quickly remove host from zone in case case of failure. There is field called TTL assigned to every record in zone and this field defines how long can be this record cached, maximum time between change in zone and propagation to all client should be TTL a SOA. However there are Internet Service Providers ignoring this standard so it is possible that some client will still get wrong records even after TTL expiration. Sample zone file with AAAA and A records and TTL 6 minutes is in figure 2.3.5.

More advance variant of DNS based load balancing is modification of zone performed by authoritative DNS server. There is usually just one AAAA/A record for service hostname, but returned IP address can be different for every query. This method may use geolocation and return IP address of the nearest server according to user's position, however user can use different recursive servers and geolocation can be very inaccurate. Technically this method is only variation of previously mentioned with better control of distribution and there is still same problem with TTL. This method is used by web portal Seznam.cz for load balancing between primary and secondary data center. They use TTL 5 minutes and also experienced problem with incorrect caching however I am not allowed to publish any detailed information.

Another problem with DNS load balancing, especially failover, is DNS pinning. It is mechanism implemented in web browser to make DNS rebinding attacks more difficult. This attack is based on pushing faked DNS record to client and then forward all traffic to attacker's IP address. Browser with pinning implemented "pins" first resolved IP address and use it even after TTL expiration so it basically prevent load balancing mechanism to switch client to another computing node. Further information can be found in [18].

Figure 2.3.5: Example zone file for DNS load balancing

```
app.example.com 360     IN      AAAA    2001:db8::1
app.example.com 360     IN      AAAA    2001:db8::2
app.example.com 360     IN      AAAA    2001:db8::3
app.example.com 360     IN      A       192.0.2.1
app.example.com 360     IN      A       192.0.2.2
app.example.com 360     IN      A       192.0.2.3
```

Application level load balancing

One of the most flexible method is application load balancing. Is is performed on Layer 7 so it is possible to differentiate in all lower layers. This solution is beneficial because application is able to decide on exact mapping between customer connection and working node. Customer is connected to balancing part of application at first. This part (group of nodes) is responsible for redirecting or forwarding request to computing node. It is possible require login before redirecting and then forward request according to information required. Every information about customer is already available, like IP address and login name, so computing node can be selected and it is also very simple to achieve session persistence. Balancing procedure is depicted in figure 2.3.6.

Advantage of this method is direct connection between client and computing node, so balancing part is not overloaded with translating requests between users and computing nodes. Direct connection eliminates bottlenecks because there is not any central authority responsible for load balancing. Technically there is central authority in load balancing part, but it can be redundant and balanced using other method, e.g. DNS load balancing. However it is necessary to expose computing nodes to users network and thus some may say that is insecure. I think that exposing computing nodes to outside word is not security hazard, because security should be provided by proper application design and network security. Obscurity is not good security approach in my opinion.

Anycast load balancing

It is possible to use anycast routing for load balancing and distribute workload between nodes. Model situation is depicted at figure 2.3.7 Only anycasted IP address is propagated to outside world, so every incoming packet have this destination address. This address is also assigned to local interfaces of computing nodes and advertised to local router using any routing protocol, e.g. OSPF.

An incoming packet is delivered to local router and this router performs lookup and selects destination address according to it's actual routing table. This is advantageous because it is possible to assign priority to routes propagated by computing nodes and node is almost immediately remove from routing table in case of node failure.

However this solution is not capable to provide session persistence because packet can be routed to different computing node every time. There would be a bottleneck in topology described in figure 2.3.7 but this method can be adjusted to eliminate

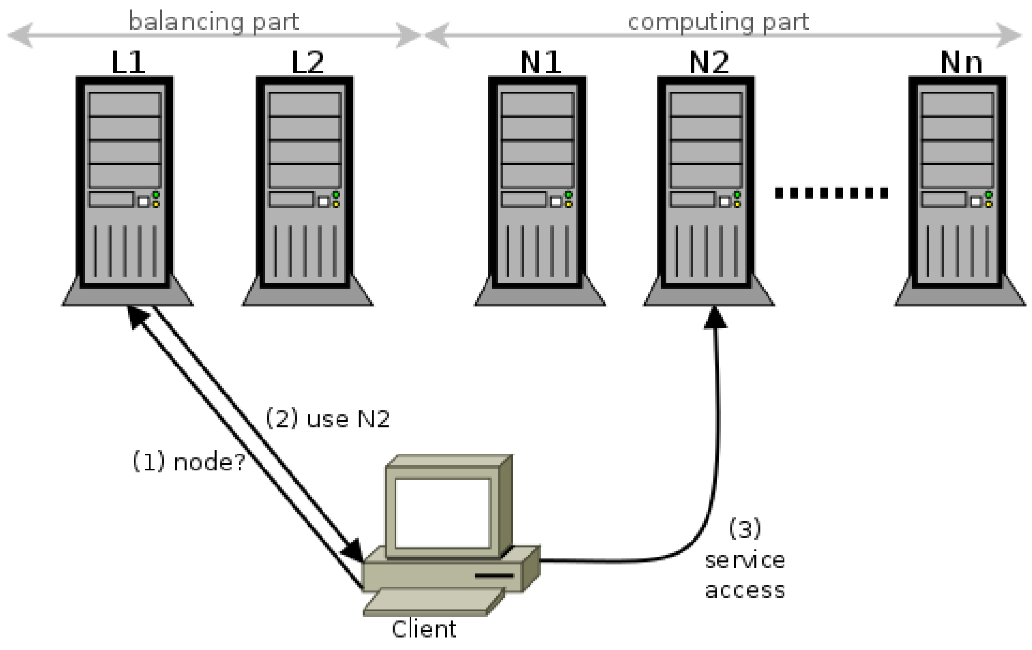


Figure 2.3.6: Load balancing at application level

this problem and propagate different anycast addresses from different autonomous systems.

Global pool of root DNS servers use exactly this load balancing principle so request should always be delivered to the nearest server and thus almost perfecty distributed around word servers. According to data published in [20] up to 80% of DNS queries are routed to the nearest anycast instance.

2.3.4 Load balancers

Load balancing of TCP flows can be done on network layer using box called load balancer. It does not have to be strictly physical box since there are also software solutions. This box modifies headers and basically translate flows from customer's side to internal and back.

The simplest solution is rewriting destination address. Packet received on external interface of load balancer is analyzed, destination address is changes to one of the computation nodes and packet is delivered to computation node. Address rewriting must be performed also on packet received from computing node as well as on every related packet.

Load balancer box must maintain list of available computing nodes and continuously monitor their status because it have to select suitable node for every incoming flow. Monitoring method and node choosing algorithm depends on application. It is also possible to integrate monitoring service into orchestrator and then control load balancer via orchestrator.

It would be probably required to guarantee session persistency so load balancer will keep mapping table between customers and computing nodes. This table provides information about current mappings and thus make it possible to deliver all packets from single flow to same computing node.

There may be additional technologies integrated in load balancer box, for ex-

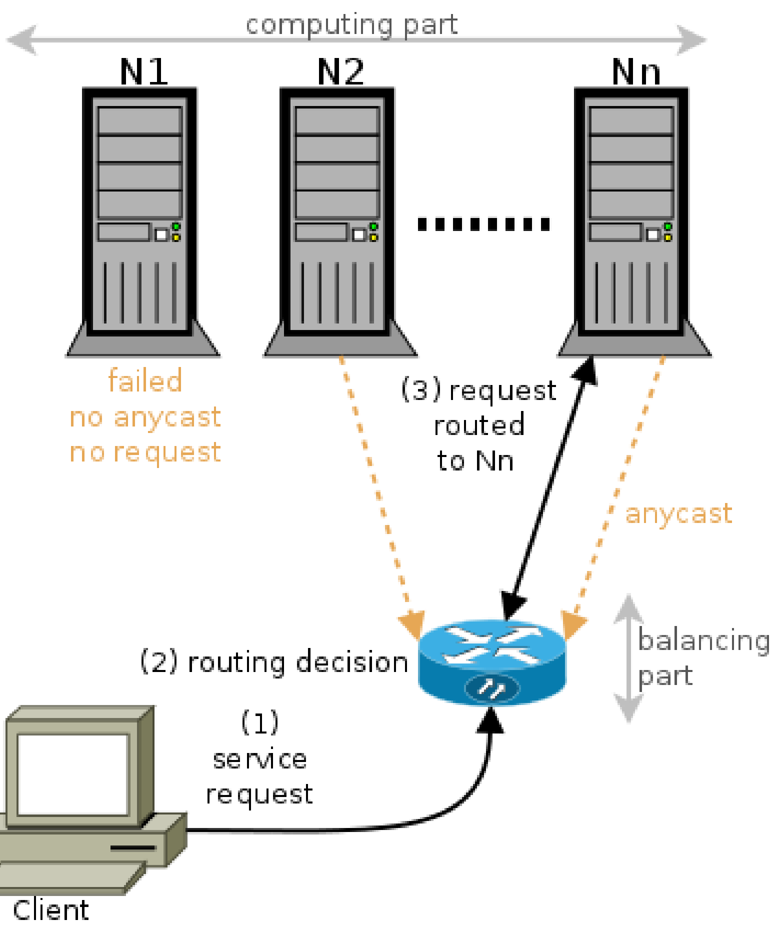


Figure 2.3.7: Anycast load balancing

ample offloading, deep packet inspection or intrusion prevention. SSL offloading is used sometimes to decouple encryption from application running on computing nodes and to enable header rewriting. It is also possible to terminate TCP session on load balancer and establish new session for communication with computing node with maintaining packet's payload. It is even possible to carry out translation between IPv6 and IPv6.

Load balancer box provide advanced function but is introduce bottleneck and single point of failure. Rewriting of packet headers and maintaining mapping table are even more resource expensive than router described in anycast load balancing. This solution is suitable for legacy application without any possibility of adding load balancing capabilities.

2.4 Storage

Storage is an essential part of datacenter since it provides space for saving information. We can distinguish between different types of storages and dozens of storage access methods so it is obvious that building one universal solution for all use cases is impossible.

2.4.1 Physical storage

Storage solution can be divided into layers. First layer is physical storage and it is represented by physical hardware used to save data onto. It is for example physical rotary drive, SSD, tape drive or any other kind of physical storage. Every server can be equipped with smaller drive and use it is uses for services running on this node. This approach is called Direct Attached Storages and it is connected with share-nothing architecture. The advantages of share-nothing is high level of node's independence, but I think that this solution is not flexible enough to be used as main storage scenario for whole data center.

Another way of providing servers with semi-physical storage is using some sort of shared storage. There is a box stuffed with physical storage and other servers are using this storage via some standardized protocol like FCP or iSCSI. Shared storage brings better flexibility compared to direct attached storage, because physical storage is not fixed to a single server. However shared storages is usually centralized too and it can bring bottleneck or single point of failure.

Physical storage can be organized into layers called tiers. Tiers are groups of physical storage devices usually organized by their performance, reliability or price. It is obviously better to uses only high performance and reliable devices, however these are usually the most expensive ones with low capacity/price coefficient. Tiered storage can provide high performance and high capacity together but it is necessary to use different tiers and optimize data placement. Storage with 3 tiers can use following drives:

- tier 1 - FC drives
- tier 2 - SATA drives
- tier 3 - magnetic tapes

Exactly this 3 tier model is used by CESNET's data storages facilities in Pilsen. However tier model can be extended with more technologies such as flashcache[2] capable of increasing drive IOPS by writethrough caching to another faster drive. Another extension layers may be introduced by using different RAID for every tier.

It is obviously not possible to share physical storage in distributed datacenter but semi-physical storage (like iSCSI) is technically possible to share. However latency can be serious problem, because a few millisecond can be significant increase to disk IO operations. For example RTT from server located in Czech republic to Norway (Trondheim) is about 50 ms and 150 ms to USA (Stanford). This time is not acceptable for sharing physical storage, because each disk access time performed in distributed storage will be about 100 ms longer than physical access time. It may be possible to use this kind of storage of longer distances using direct optical connection with minimal RTT, like CESNET Lambda or Photonic.

2.4.2 Virtual storage

To deliver storage for physical server is only the first problem, since it is required to split this raw storage and use for another storage layers. There are three fundamental virtual storage types: block storage, file system and object storage. It does

[2] `http://github.com/facebook/flashcache`

not matter if this virtual storage is used directly by physical server or by virtual server via any virtualization layer since access method is actually the same.

Block storage is used way similar to directly attached. Block device is exposed to operating system and storage is accessed through block device. Storage operations are managed directly by the operating system and it is main characteristic of this storage type. Block device can be formated with filesystem, used as physical volume for LVM or used as encrypted storage.

Filesystem storage is used to save files and it's attributes. Filesystem can be build upon local block device or accessed as NAS. Filesystem on local block device can deliver lower latency and it is easier to match permissions with local user accounts, but these advantages count only for strictly local filesystem. I think that using NAS is almost required in case when any between user or computer sharing is required. Storage accessed via network will probably be easier to maintain and it is also easier to perform backups.

Object storage is something between previously mentioned storage types since it is capable to save objects and access them similar way as accessing files in filesystem. However object storage is more general way of saving data and can be used to store various objects and it is not limited only to files. It provides very high level of storage abstraction because it is capable to operate on almost any data, access it in standardized way and decouples object from their location.

2.4.3 Network access

I think that traditional approach with separate block and file storage is becoming nowadays quite limiting. There are some request which are partially mutually exclusive:

redundancy and distribution to spread load and fail resistance

agility and scalabity to provide flexible and elastic storage

security because it is critical to avoid any unauthorized access or leakage

inteoperabilty between different technologies and vendors

Distributed storages is a solution which can fulfill all of these requirements. This kind of storage can not referred as neither block nor storage, because it is necessary to use different storage element. These elements are stored on storage nodes according to storage map and rules. It is possible to define how many time should be each element saved, on how many node or on which type of physical storage.

Element mentions in paragraph above can be easily referred as an object, so object storage can be build in distributed way. However not every object storage is distributed storage, because object storages can be local too.

Typical example of distributed object storage is Ceph, referred in [10]. It uses Reliable, Automatic, Distributed Object Store (RADOS) mechanism to store object in Ceph cluster. Every type of data is stored as an object in flat namespace so it doesn't matter whether it is text, file or binary image.

Common problem of distributes storages is central gateway, which is used for client connection and coordinates whole cluster, so it introduces single point of failure. However Ceph eliminates this by using CRUSH algorithm to compute object location without querying central lookup table. Each client can compute object placement on it's own and then directly connect to storage node.

Cluster build with Ceph consist of two types of nodes OSD and monitor. OSD is used to store objects and monitor is responsible for maintaining placement map and monitoring of other monitors and OSD. It is necessary to design cluster well because performance can be poor otherwise. It is, for example, not recommended to place OSD and monitor on one disk, because many parallel IO operations will be requested.

Object store is base and it can be extended by other services, there is for example RBD providing block devices and CephFS used to store files. Distributed object storage is well suited for distributed datacenter because it can be designed to provide shared storage. However it introduces additional abstraction layer in storage system, so it will probably provide worse performance compared to strictly physical storage. It is tradeoff for flexibility.

2.5 Orchestration software

There are common routines in cloud data center administration and these routines are repeating very frequently. For example simple workflow for virtual machine creation can involve:

- clone image with prepared operating system
- log into hypervisor console and create VM definition
- deploy virtual machine
- configure firewall and router
- configure vswitch or attach virtual machine into bridge
- set up network interfaces in VM
- set root's password and add authorized keys
- update monitoring definition

There can be dozens of task similar to mentioned above and it can take negligible amount of time. These task are usually very simple and all necessary information can be generated automatically or loaded from an information system. It is very favorable to perform these task automatically because it does not need any assistance of human and automated solution is much more faster and strictly deterministic.

Orchestration is automated management of services and resources performed according to predefined procedure. An inteligence is implemented into orchestrator so it can make desicions and execute actions without an interaction with human. Orchestrator acts autonomously according to configurated parameters in contrast to remote control interface which only perfroms requested actions.

It is necessary to use orchestration for every cloud solution because it is not possible to cope with manual configuration and management of many cooperating services and resources. Rapid provisioning with minimal management effort is required in cloud computing definition mentioned in the beginning of this chapter and it can not be accomplished withou orchestration.

There exists actually one more step between completely manual management and orchestration and it defined by using configration management. Configuration management solution is used for uniform management of configuration and executing repetitive task. It is possible to develop own solution or use any software available. For example Ansible, Puppet or SaltStack are well know open-source softwares. Configuration management software can be instegrated into orchestrator and used as a interlayer between orchestrator and performed actions. Ansible is used for virtual machine configuration used in practical part of this thesis as well as for installation and configuration of OpenNebule IaaS cloud. It creates cloud environemnt with defined parameters and prepare initial configuration so it significantly shorten time required for installation and also eliminates configuration mistakes.

2.5.1 OpenNebula

OpenNebula is open source cloud OS capable of building IaaS solution, so it is technically an orchestrator. [16] However it is not only an orchestrator but complete solution for datacenter orchestration capable to build IaaS. It was initially created as an research project in 2005 and the first public release was in 2008. It is currently developed by the community in cooperation with OpenNebula Systems.

It is completely platform agnostics so mayor virtualization techniques can be used. KVM, XEN and VMware is supported at current time but it is possible to develop modules for other virtualization platforms. There is for example driver OneLXC developed by China Mobile and this driver brings support for LXC hosts and containers.

Project architecture is module and can be modified according to system requirements. There is one node called frontend which is responsible for orchestration and other nodes are used as computing nodes, i. e. hypervisors. It is not required to dedicated separate hardware node for frontend because it can be deployed on physical server together with computing node. However it is recommended to deployed frontend as a virtual machine in HA since it is more flexible and robust. Sample physical infrastructure is depicted in figure 2.5.8.

Frontend acts orchestrator and uses additional modules to operate cloud infrastructure. There modules are universal to work with various underlying systems and each module must provide standardized interface for orchestrator. Modules and functions as defined in [16] are:

Infrastructure and cloud drivers enable access to infrastructure and cloud providers

Virtual machine manager is used for managing VMs and executing action on them

Network manager provide network configuration and management

Storage manager supply storage for services and customers

Image manager maintain library of VM images

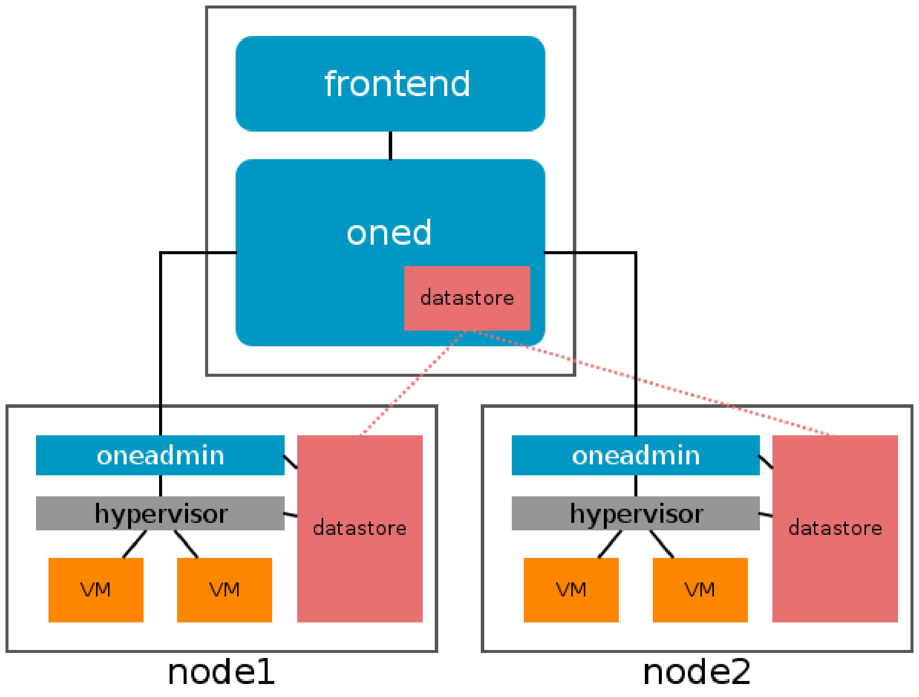


Figure 2.5.8: OpenNebula architecture

Information manager is collecting runtime information about physical infrastructure, VMs and other devices

Authentication and authorization is used to authenticate users, store information about them, their permission and quotas

Accounting and auditing gather information about resource usage and can be used to generate billing data

Federation manager provide mechanism to access remote cloud providers

Scheduler manages initial placement of new VMs according to scheduling policy

Administrative tools provide interface for users and administrator to perform task on cloud system

Service manager can work with group of interconnected VMs as with one service with defined requirements and deployment rules

Orchestration is performed by frontend and remote tasks are executed at nodes using SSH. There is a single point of failure because frontend may go down so it is recommended to use HA solution and minimize possibility of frontend unavailability. However frontend failure does not affect running virtual machines since they stay online but monitoring will stop and it will not be possible to execute any action on virtual machines.o

Datastores

Storage part of OpenNebula system is called datastore. It is abstraction of physical storage and is used to store persistent and non-persistent data. Persistent data are preserved during whole VM life cycle and non-persistent objects are restored to default state after virtual machine recreation. There are three types of datastore according to type and format of stored data:

image datastore is used to store images of non-running virtual machines

system datastore hold images used of running VMs

files datastore is used to save single files like kernels, contextualization data and files which are stored alone, meaning not as part of image

The image of virtual machine is cloned from image datatastore to system datastore during deployment phase and then copied back after shutdown if image is persistent. Non-persistent images are not save back to system datastore so they can be directly destroyed. It is necessary to select technology for transfer to system datastore at nodes. There are options listed below, but it is possible create script for any other method original scripts are located at /var/lib/one/remotes/{datastore,tm}/.[3]

shared is filesystem directory and OpenNebula does not care about sharing technology, it just expects directory to be available on every node

ssh can be used to transfer the images, it is always available but also vastly slow

vmfs copies images using vmkfstools (VMware)

qcow driver uses qemu-qcow to handle images

ceph use ceph cluster to store images as RBDs

lvm images are shared using clustered LVM

Networking

OpenNebula can assign virtual network to every running VM so networking driver is run during virtual machine deployment and virtual machine is connected to virtual network according to virtual network definition. Networking driver can provide virtual machine isolation and basic network configuration. Network manager takes care about leased IP addresses[4] and generates contextualization.

The simplest network driver is called dummy and VM's interface is only added info bridge using bridge-utils. Destination bridge must be configured in advance. This drive does not provide any additional functionality but it can be used as an example for writing customized network drives. Every network driver can be extended with hooks too.

Little more advanced driver is fw and it does the same job as dummy driver but it can configure firewall too. Firewall rules are applied at physical host so it is not necessary to install any software into virtual machine. Iptables package must be install on node to use this driver. Firewall rules described in figure 5.3.6 are created after VM deployment and removed after shutdown. TCP and UDP ports can be whitelisted or blacklisted and it is also possible to drop incoming ICMP packets. Driver's capabilities can be easily extended by editing scripts located at /var/lib/one/remotes/vnm/fw/{pre,post,clean}.

[3]It is necessary to run `onehost sync` after changing any remote script at fronend.

[4]IPv6 is supported as well as legacy IPv4

Figure 2.5.9: Iptables rules created by fw network driver

```
# Create a new chain for each network interface
-A FORWARD -m physdev --physdev-out <tap_device> -j one-<vm_id>-<net_id>
# Accept already established connections
-A one-<vm_id>-<net_id> -p <protocol> -m state --state ESTABLISHED \
-j ACCEPT
# Accept the specified <iprange>
-A one-<vm_id>-<net_id> -p <protocol> -m multiport --dports <iprange> \
-j ACCEPT
# Drop everything else
-A one-<vm_id>-<net_id> -p <protocol> -j DROP

# Create a new chain for each network interface
-A FORWARD -m physdev --physdev-out <tap_device> -j one-<vm_id>-<net_id>
# Drop traffic directed to the iprange ports
-A one-<vm_id>-<net_id> -p <protocol> -m multiport --dports <iprange> \
-j DROP

# Create a new chain for each network interface
-A FORWARD -m physdev --physdev-out <tap_device> -j one-<vm_id>-<net_id>
# Accept already established ICMP connections
-A one-<vm_id>-<net_id> -p icmp -m state --state ESTABLISHED -j ACCEPT
# Drop new ICMP connections
-A one-<vm_id>-<net_id> -p icmp -j DROP
```

802.1Q driver uses VLANs to isolate virtual machines. It creates bridge for every virtual network, assigns VLAN id to this bridge and attaches physical interface defined in PHYDEV variable. Physical interface is in trunk mode because it transfers already tagged Ethernet frames. This approach is beneficial because VLAN aware network switch can be used to forward tagged traffic. VLAN support is required on nodes so it is necessary to load kernel module called *8021q* or compile support directly into the kernel. VLAN id is calculated as a sum of $CONF[:start\_vlan]$ from /var/lib/one/remotes/vnm/OpenNebulaNetwork.rb and virtual network id, however both can be edited or course.

Driver called ebtables is simple but can be useful is many cases. It uses ebtables package and create ebtables rules described in figure 2.5.10. It effectively prevent virtual machine from changing it's assigned MAC address and eliminates possibility of mac spoofing.

Figure 2.5.10: Ebtables rules uses by ebtables network driver

```
-s ! <mac_address>/ff:ff:ff:ff:ff:0 -o <tap_device> -j DROP
-s ! <mac_address> -i <tap_device> -j DROP
```

The most advanced driver is is Open vSwitch (OVS). This driver provides same network isolation functionality as 802.1Q driver but also enables to use special functions provided by Open vSwitch, for example OpenFlow rules or using logically centralized network controller.

There are two variants of this Open vSwitch driver:

- ovswitch can be used only with KVM nodes
- ovswitch_brcomat can be used with KVM and Xen, however this driver requires compatibility layer for bridging

I think that this driver is the best choice because it provides all functionality of Open vSwitch. It means that is it possible to use advanced filtering, NetFlow, traffic shaping and the most important thing is OpenFlow. OpenFlow is control plane protocol for forwarding plane configuration so it is possible to decouple control plane from switch and let network controller to manage switches remotely. It is possible to manage physical and virtual switches together and create one converged network. I think that Open vSwitch driver is the best choice if advanced configuration is needed apart from use cases when simple bridging is sufficient

However this drive is the most difficult to configure because Open vSwitch must be installed on nodes. It is necessary to have OVS support in kernel and install userspace tools. Last version of OVS is 2.3 and it support linux kernel version 2.6.32 to 3.14 so new kernel version can not be used to run nodes with Open vSwitch. However mayor distribution use compatible kernels, at least version with long term support. For example latest Ubuntu server version 14.04 is using kernel 3.13.0 so there is not any incompatibility problem.

Templates

Virtual machine deployment in cloud OS is different from method used in bare virtualization because it is not possible to create virtual machine directly. It is typical for bare virtualization that it is necessary to manually create virtual machine, generate or import disk image, configure parameters and then boot it. However it is not longer possible because VM deployment is managed by virtual machine manager module and user is not able to directly interact with hypervisors.

OpenNebula is using concept of templates for all virtual and physical entities. Template is plain definition of parameters and is used by modules. Template file for virtual machine used for measurements in practical part is in figure 2.5.11. For example virtual machine manager read template and creates virtual machine using infrastructure driver and VM is then deployed by scheduler. It is of course possible to manually edit parameters of virtual machine however initial creation must be always performed by manager.

Contextualization

It is common that single virtual disk image is running in many instances and it can be used for horizontal scaling or failover. The group of virtual machines deployed for same purpose is called pool. It is necessary to clone disk image from image repository to system repository for every machine and second even more important task is to adjust configuration parameters. It is not applicable to run

Figure 2.5.11: Template for virtual machine

```
CONTEXT=[
        CONTEXTUALIZED="1",
        NETWORK="YES",
        SET_HOSTNAME="themis-VM",
        SSH_PUBLIC_KEY="ssh-rsa AAtb-shortened-geNmcJO8QbyG/xLOP",
        THEMIS_TYPE="VM",
        THEMIS_USER="root"
        ]
CPU="1"
DESCRIPTION="VM ready to be uses by Themis project"
DISK=[
        IMAGE="themis - VM - Ubuntu server 14.04. base",
        IMAGE_UNAME="tom"
        ]
GRAPHICS=[
        LISTEN="0.0.0.0",
        TYPE="VNC"
        ]
MEMORY="512"
NIC=[
        IP="10.104.33.8",
        NETWORK="club Buben - Themis",
        NETWORK_UNAME="tom"
        ]
OS=[
        ARCH="x86_64"
        ]
```

each machine from single pool with same configuration since at least MAC address, IP address and hostname need to be changed.

Changing parameters before first boot is one of available solutions. Hooks can be used to mount disk image, perform required changes, unmount it and boot virtual machine. However this solution is slow and computation expensive.

Another approach is performing imperative configuration after initial boot. Every machine can use same IP address during first boot and it will be changed by a configuration management system, e.g. Ansible or Puppet. There are some problems which are not easy to solve and the most serious is simultaneous booting of multiple virtual machines because it is not possible to duplicate IP within single virtal network. Change of IP address will cause interruption of any ongoing communication including management channel (SSH for example). This approach will not scale well because there is a central authority responsible for initial configuration and it is not possible to boot many virtual machines simultaneously due to IP duplication problem.

Solution used by OpenNebula is called contextualization and it solve all of problems mentioned above. First problem to be solved is how to deliver contextualization information to virtual machine. There are two contextualization mechanisms with totally different approach.

Auto IP assignment is capable to configure only IP address. Hypervisor can assign MAC address of VM's network interface and this address is used for IP address autogeneration. Virtual machine's disk image need to be updated with file /etc/init.d/vmcontext.sh which is executed during boot in runlevel 2. This Bash script parses 3.-6. group of hexadecimal numbers from MAC address, converts each group to decimal base and assigns this as an IP address. It usually just generates configuration and restarts networking script, but it depends on distribution used and can be changed really easily. However there is always used 255.255.255.0 network mask thus this contextualization approach can not be used for networks with different mask. Example of IP generation is in equation 2.1. Autogeneration algorithm can be upgraded to work with different network mask by modifying vmcontext.sh script to read mask from first or second group, but first three groups of MAC address are OUI so generated MAC addresses may collide with range already assigned to an existing organization.

$$02 : 00 : \underline{0a} : \underline{68} : \underline{21} : \underline{08} \rightarrow 10.10.33.8 \tag{2.1}$$

Second contextualization approach is called general contextualization and it is more mighty than previous one. File named context.sh is used to save all information and configuration. This file is generated before deployment of virtual machine and it can be easily extended with user defined variables. Contextualization file is packed into binary image with ISO 9660 filesystem and configured as disk in virtual machine. It is still necessary to provide additional script to read and apply contextualization but it is much more powerful than previous approach. Main script is called vmcontext and it is responsible for mounting image with contextualization, loading contextualization variables and executing scripts /etc/one-context.d/*.

There are already prepared scripts for configuration of network and dns, generating autorized_keys file, mounting swap, setting hostname and executing addition script supplied from files datastore. Contextualization file used in practical part is in figure 2.5.12. There are two custom variable THEMIS_TYPE and THEMIS_USER defined in template and used by middleware. Purpose of other variables is obvious.

Figure 2.5.12: Contextualization file

```
# Context variables generated by OpenNebula
CONTEXTUALIZED='1'
DISK_ID='1'
ETH0_DNS='10.104.1.2 8.8.8.8'
ETH0_GATEWAY='10.104.1.1'
ETH0_IP='10.104.33.8'
ETH0_MAC='02:00:0a:68:21:08'
ETH0_MASK='255.254.0.0'
ETH0_NETWORK='10.104.0.0'
NETWORK='YES'
SET_HOSTNAME='themis-VM'
SSH_PUBLIC_KEY='ssh-rsa AAtb-shortened-geNmcJO8QbyG/xLOP'
TARGET='hda'
THEMIS_TYPE='VM'
THEMIS_USER='root'
```

Migration of virtual machines

Virtualization is enabling technology for cloud computing and it provides separation of running operating system (VM) from physical machine. Software can be fully decoupled from hardware and located anywhere in datacenter or even migrated between datacenters.

Virtual machines (VM) are running on physical servers called nodes or hypervisors. Migration is process of shifting virtual machine from source hypervisor to destination hypervisor. Migration should be undetectable for virtual machine since software running inside must stay running and remain intact. It can be theoretically possible to detect ongoing migration but detection will be based performance changes so it should not be possible to detect hypervisor VM is running on.

There are two different types of migration - cold and live. Both types migrate virtual machine from source hypervisor to destination, but difference is in migration parameters and method for solving boundary value problem.

3.1 Migration of resources

Virtual machine can be abstracted as a group of mutually cooperating resources. There resource are required for virtual machine operation and thus all of the resource need to be transfered to destination hypervisor during migration. Resources can be

- VM parameters (e.g. number of CPU, allocated memory, virtual NIC)
- image of system disk (used to boot operating system)
- additional images (e.g. CD-ROM images, encrypted block devices)
- other interfaces (e.g. physical USB devices)
- virtual network
- memory

Migrating all of these resource may not be trivial because different migration approach must be used. Migration of VM parameters is easy because it is just very small plain text, XML or JSON file. There are technologies to migrate other resources, except physical devices attached to hypervisor.

3.1.1 Storage

Image of system disk, as well as additional images, need to be available for destination hypervisor. Easiest solution is to transfer images during migration but it can take long time, consume all bandwidth available and thus significantly affects service performance as well as other tenant's traffic. Workaround based on dynamic rate-limiting is proposed in [5].

More advanced solution is using shared storage. This storage is shared between all of hypervisors and thus all images are immediately available. It can be used for distributed datacenter where hypervisor may be distant in geographical and network manner. Critical parameter for storage in distributed datacenter is round-trip time because synchronous write operations on storage need to be acknowledged. It is possible to use storage in asynchronous mode but it is dangerous because data corruption may occur and single control node is needed. Single control node is single point of failure as well.

I think that migration between datacenter (i.e. in distributed datacenter) should combine different technologies. It does not make sense to store all images in inter-datacenter shared storage because there is significant performance penalty caused by IO operations transfered over network. Migration can be realized in two steps. First image can be migrated from inter-datacenter to intra-datacenter storage and then whole virtual machine migration can be carried-out.

3.1.2 Network

Some of tenants may require to maintain Layer 2 connectivity after migration, but this practically mean that Layer 2 connectivity between hypervisors is needed.

It relative easy to build Layer 2 connectivity between hypervisors in single datacenter, but it gets much more complicated for distributed datacenter. Overlay networks described in 2.3.1 are capable to spread Layer 2 between datacenters, so this technologies can be used if required by tenant.

However I think that it is better to build application without L2 connectivity between computing nodes, for example by using load-balancing approach on higher layers, because L2 connectivity is quite limiting for moving to another datacenter or cloud-bursting. It is better, at least in my opinion, to use other ways to provide communication between virtual machines than using overlays for geographically large installations.

3.1.3 Memory

Memory migration is necessary to preserve VM state during migration, i.e. perform live migration. It is not necessary to migrate memory for cold migration because virtual machine is powered-off and thus memory is actually empty during migration and can be easily recreated on destination hypervisor.

Migration procedure must be able to read VM's memory at source node and create identical copy on destination node. However virtual machine is still running on source node and memory is constantly changing at source. Transfer mechanism proposed in [5] introduce three phases of memory migration:

push Memory pages from source to destination and labels pages changed during transfer as "dirty". Dirty pages are transfered in next round. However is is not possible to transfer all pages during this phase, because some pages get dirtied faster than they can be transfered.

stop-and-copy Virtual machine is paused and all remaining dirty pages are transfered. This phase is used to transfer remaining quickly dirtied pages as page dirtying is paused.

pull Virtual machine is already running on destination hypervisor but there can be pages which are not copied yet so they are transfered on-demand.

Serious complication is rapid page dirtying described in [5], caused by rapidly modified pages which are dirtied promptly after their transfer. This is caused by disproportion between memory write speed and network bandwidth because it is possible to write into memory much more faster than transfer dirty pages over network. Only one available solution is to use stop-and-copy phase and stop memory writing during transfer. However memory transmission can take long time and thus it can significantly increase service downtime, but total migration time will be reduced.

3.2 Cold migration

Procedure of cold migration is simpler than live migration. Virtual machine must be in power-off state before migration, so disadvantage of this method is obvious because all running processes need to be terminated and complete operating system shutdown is needed.

Service downtime for cold migration is much longer compared to live migration, because it is required to shutdown VM and virtual machine is not running during migration. However complete virtual machine shutdown can be beneficial for virtual machines with intensive memory writes because it significantly decreases total migration time.

Cold migration is suitable for virtual machines which are part of cluster with working failover and shutdown of single virtual machine is not going to cause service outage. Another appropriate case is migration without shared storage and thus image must be transfered during migration, however disk transfer may be beneficial if it is necessary to change datastore or even virtualization technology.

This type of migration is easier to perform in distributed datacenter than live migration because resources, like disk image, can be converted during transfer and shared storage is not required.

3.3 Live migration

Migration can be performed "live" almost without service disruption. According to measurements provided in [5] can downtime be as low as 60 ms but it depends on application and infrastructure parameters.

Live migration provide administrator with tool for shifting virtual machines between hypervisors without any significant outage. It is beneficial for cloud administration and maintenance because it is possible to move VM as required. It allows to make hardware upgrades since all virtual machines can be migrated to another hypervisor, hypervisor can be upgraded and then VM migrated back. It is extensively used for IaaS because infrastructure administrator can migrate machines without the need of root access into VM.

It is necessary to make migration in secure way so virtual machine must not stay unusable on both hypervisors. There are 4 basic step which need to be performed:

1. VM is created on destination node, but it is paused.
2. Migration of resources is started. This include disk image migration (or sharing), memory migration and also ensuring that all other resource are available on destination node.
3. VM is paused on source node.
4. VM is resumed on destination node and deleted on source node.

Cloud orchestrator usually require shared datastore for live migration of disk images, NFS or any distributed filesystem can be used. Live migration of memory is performed by combination of push and stop-and-copy approach. Push phase is responsible to transfer most of the memory and stop-and-copy is used preferably to quickly move rest of memory. Migration mechanism should monitor duration of migration and memory writes because it may be necessary to switch to stop-and-copy phase even if significant amount of memory pages is still waiting to be transfered. This is cause by rapid page dirtying and necessary to stop virtual machine otherwise it would never finish migration.

There are extra tasks which can be executed after successful migration. It is for example necessary to update FDB and ARP table on all intermediate network boxes because virtual machine changed it's location. Obvious solution is to sent gratuitous ARP, but some router may block this kind of ARP message. Virtual machine can send directed ARP messages to all addresses in it's cache as suggested in [5]. It effectively bypass blockage and deliver the notification.

Part II

Practical

The goal of practical part is to develop system capable to measure virtual machine availability during migration.

Virtualization is enabling technology for VM migration since it decouples virtual machine from physical server. However there are some dependencies which must be transfered together with virtual machine such as virtual network and disk image. Transfer of disk image can be solved by shared storage or using cold migration, but it is not very clear how will migration affect networking communication.

Network connected to virtual machine is called virtual network but there need to be physical infrastructure capable to transfer signal between virtual server and other side of communication (usually customer). It is possible to migrate virtual machine to any server almost without any limitations but there may be serious problem with virtual network.

Virtual machine availability can be measured on many layers. Application layer can be tested by simulating customer requests. There is a program called ApacheBench for example and it is capable to send many parallel request to HTTP servers and measure statistics like time for request, transfered bytes etc. There are other similar benchmarks as dkftpbench or SPECweb99 but they are always limited to one type of service or protocol. I think that this limitation is critical for virtual machine migrations testing and it is necessary to use more general approach.

General measurement procedure should be service and platform independent so only possible solution is to perform measurement directly on network layer. Statistics acquired by network measurement may be approximately converted into higher layers and estimation of service availability can be done. Network measurement can give some information about network environment during migration as well. Packet loss is the most relevant sign for virtual machine availability during migration but there are other parameters which should be taken into account. For example packet delay can significantly affect service quality, especially for storage.

Testing of virtual machine migration is extensive topic because there are additional parameters specific for migration of virtual machines. Total migration time may be used to decide whether it is convenient to use live or cold migration. Total migration time is crucial indicator used in emergency migration cases which may include unplanned outage or a natural disaster. Another special parameter is migration success rate since migration request can be unsuccessful in some cases.

I think that it is important to introduce application capable to evaluate virtual machine migrations so I have developed application called Themis[1]. It is modular framework which can be instructed to provide migrations in defined way, collect performance data and export them to user.

[1]Themis is Greek Titaness usually depicted holding scales and it is a reason why application is called Themis.

Methodology overview

First thing necessary to be defined is methodology used. Testing framework should be universal as much as possible so I have decided to measure performance network layer since this it the most universal technology used. Same is also applied to used technologies so it should be possible to run an application on many platform.

Primary task of an application is to measure availability of virtual machine during it's live migrations. There are two virtual machines used: VM under test and supervisor. Virtual machine is migrated between hypervisor while measure session between VM and supervisor is established. Supervisor is deployed as a virtual machine because it can be moved to any location and actually change testing parameters, but supervisor is fixed and is not migrating during measurement session. It can deployed as physical machine as well.

It is expected that VM retains it's IP address because there is measurement session established and it would break in case of address change. It is possible, however, to measure migration with IP change, but it is necessary to use VPN or advanced routing to provide VM-supervisor connectivity.

Framework is ready to measure cold and live migration. Live migrations can be used without any special configuration, but VM must be prepared to perform cold migration. Machine is powered-off during migration and then booted at destination hypervisor so it is required to start migration agent right after booting. This may be achieved by init script or process monitoring framework, e.g. God.

4.1 Measurement session

Session between VM need to be established to obtain data for analysis. Packet generator and receiver need to be running on VM and supervisor. I have developed agents capable to run session and export results back to the backend.

Traffic generator are investigated and compared in [14] and [21]. I have decided to use iperf because this tool is widely available, runs on many platforms and gives similar results as others without any significant deviation. It really does not matter which tool is used because it is very easy to adjust management module and agent code to use different tool.

4.2 Management

Managements access is used to orchestrate virtual machines as well as for orchestrating an orchestrator. These steps need to be performed for one session:

1. load measure session parameters
2. launch measurement agents on VM and supervisor

3. request migration (VM is migrated, supervisor is fixed)
4. wait for migration to finish (e.i VM is in running state)
5. end measurement session
6. check whether migration was correct

Various protocols can be used to run commands on virtual machines but SSH is most common and provide all required features as well as sufficient security level.

Management module must be able to control OpenNebula orchestrator and acquire information about hosts and virtual machines. There are various methods how to control orchestrator. OpenNebula provides low-level API via XML-RPC with wrappers available in Java and Ruby. There is also an OCCI interface implemented, but XML-RPC interface seems to be better choice because it is tailored for OpenNebula.

Security aspect must be taken into account because it is unacceptable to allow unauthorized access to OpenNebula cloud interface and SSH console of virtual machines. It is also not acceptable to save passwords into source code repository so another methods need to be used.

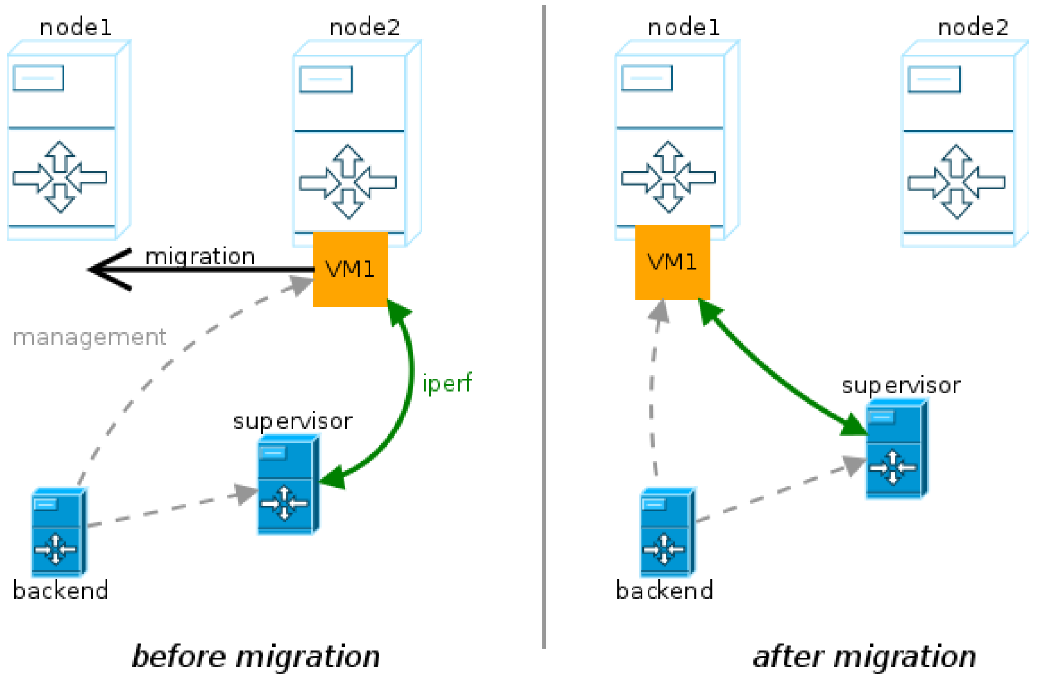


Figure 4.2.1: Methodology overview

Themis application

Themis is an application for evaluation of virtual machine migrations. It is prepared to be used for availability measurements and can be easily adjusted to perform other task during migration.

Application is modular and can be adapted for different orchestrator or to perform different task during migration. Architecture is depicted in figure 5.0.1. Backend is responsible for measurement management and results processing. Frontend provides web interface for users. Result can be displayed directly as table or graph in browser or exported into CSV.

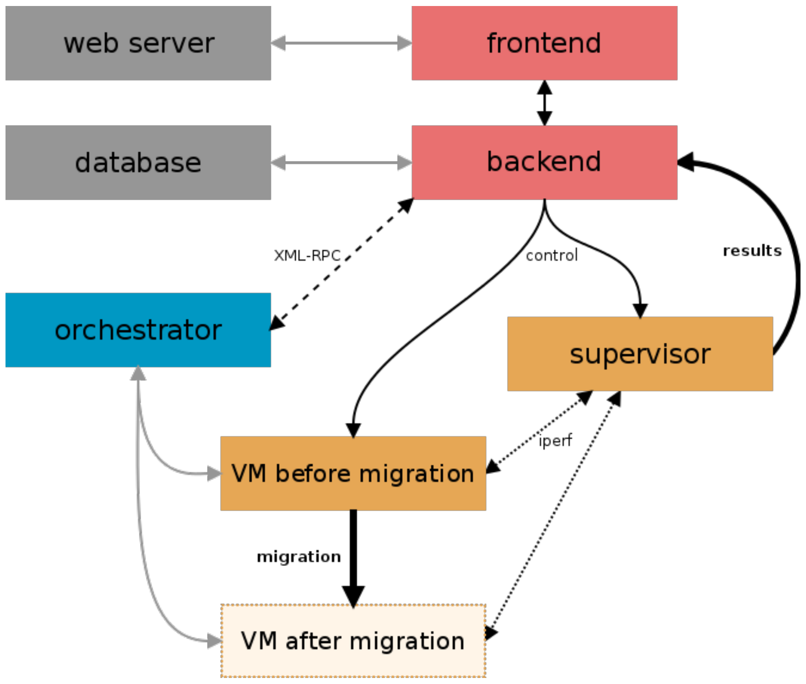


Figure 5.0.1: Model of Themis application

Application is written in Ruby using Ruby on Rails framework. Ruby is platform independent and can run on every currently used operating system. Ruby on Rails (RoR) is framework providing database abstraction and strictly based on model-view-controller (MVC) architecture. Application is based on object model, outputs are generatied using views and controller is responsible for sending commands to models and forward results to views.

I have decided to use this framework because it provides better interaction with system services, e.g. SSH and SCP, than other web frameworks. There are public available classes for interaction with OpenNebula and OpenStack cloud API so it is not necessary to create XML-RPC parsers from scratch.

5.1 Measure models

There are three models defining measurement tasks and results. Definition, session and transfer. These models are used to describe migration tasks instructions, migration progress and results. Relation between models is depicted in figure 5.1.2.

All measurement models mentioned bellow are descendants of *ActiveRecord::Base* and mapping between objects and tables is handled by this build-in class. It also describes inter-model associations and performs validation.

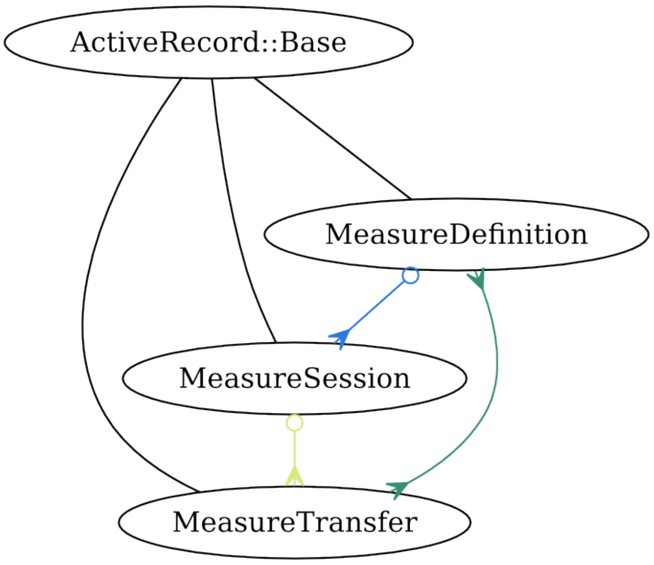


Figure 5.1.2: Relation between measurement models

5.1.1 Definition

Definition class, formally MeasureDefinition, is used to save prescription for measurement task and track time taken. Parameters are listed in table 5.1.1 and their meaning is described bellow.

vm is virtual machine which is going to be migrated. List of VMs available for migration is loaded on-demand from orchestrator using OneOrchestrator class. Virtual machine must be in running state and variable `THEMIS_TYPE = 'VM'` need to be present in contextualization settings.

source is source host for virtual machine. VM need to be migrated to this host before starting measurement session. List of hosts is loaded using OneOrchestrator class and hosts status is checked for each host.

destination is destination host. VM is migrated to this host during measurement session.

bandwidth is determines packet generation rate passed to agent.

cycles set number of migration repetitions.

supervisor is an IP address of supervisor services, i.e. packet receiver and result exporter

description has an obvious meaning.

started_at is timestamp taken at the beginning of measurement

finished_at is timestamp taken after finishing all migrations.

finished_at is timestamp taken after finishing all migrations.

Definition class has one-to-many relation with session class and also one-to-many indirect relation with transfer class connected through sessions. It means that is is possible to load every information from subordinate classes (models) and it is useful for generating exports and web views.

Definition class is the highest class in the hierarchy so it is connected with orchestrator. There is a class variable @@shared_orchestrator which is providing link to orchestrator interface. This variable is shared by all class instances so it it not necessary to initialize more connections at once.

Methods for manipulation with orchestrator resources are declared in definition class. *Host_source* and *host_destination* returns host object for source and destination host. Method called *virtual_machine* returns virtual machine object which is going be used for migration evaluation.

Most important method of definition class is *start* because it executes migration evaluation. It is responsible for generation of sessions and starting all of them. One session object will be prepared for each migration cycle, so number of session objects is same as number stored in cycles parameter. Sessions need to be started one by one so there is an loop which starts new session right after finishing of previous one. Session start is performed by calling *start* of session class. This method is different from previously mentioned method with same name, because this one is defined in session class. Finished_at parameter is set to current timestamp right after finishing of last session, i.e. at the end of migration evaluation.

Migration evaluation is not very computation expensive, but it can take really long time to perform many migration cycles, in particular for virtual machines under load. It is not possible to run these long running tasks on request from web interface because request will timeout shortly and task would be terminated. I have decided to use Delayed::Job (DJ) to run these task asynchronously. DJ can run task in detached process and it can run for a long time without any timeout problems. It is also possible to run DJ worker on separate machine so migration evaluation is executed in completely separated environment. This approach eliminates interference with other processes or network traffic. Security is improved too, since orchestrator interface does not need to be accessible from machine with web interface.

There is one special function called flush and it clears measurement definition and all of it's subordinate objects. It can be used for debugging, because it is sometimes necessary to repeat migration with same parameters. However it is not possible to run migration which was running in the past and all of sessions have already finished.

Table 5.1.1: MeasureDefinition parameters

Parameter	Required	Type	Editable by user	Notes
vm	yes	string	yes	
source	yes	integer	yes	
destination	yes	integer	yes	
bandwidth	yes	integer	yes	
cycles	yes	integer	yes	required bigger than 0
supervisor	yes	string	yes	IP address
description	no	text	yes	
started_at	no	timestamp	no	
finished_at	no	timestamp	no	

5.1.2 Session

Session class, precisely MeasureSession, is a link between definition and transfers. This class is responsible for migration and measurement coordination. Remote management of virtual machines and orchestrator is performed inside this class.

There are no parameters editable by user because all necessary information about migration are inherited from definition. Table 5.1.2 describes all parameters. There are two timestamp fields with evident purpose, reference to MeasureDefinition and two fields special for this class:

seq is sequence number in scope of superior MeasureDefinition. Session with seq = 0 is going to be executed first and seq = measure_definition.cycles is the last one.

status determines status of measurement session:

- **0** - pending - session is waiting for execution, this is default state
- **1** - running - session is running right now
- **2** - done - migration was finished successfully
- **3** - failed - migration was executed and failed to finish

Table 5.1.2: MeasureSession parameters

Parameter	Required	Type	Edit.	Notes
measure_definition_id	yes	reference	no	reference to MeasureDefinition
status	yes	integer	no	
seq	yes	integer	no	
started_at	no	timestamp	no	
finished_at	no	timestamp	no	

Most important part of MeasureSession class is *start* method. This method is executed by superior definition, so it is not necessary set asynchronous execution with Delayed::Job, because higher object is already running asynchronously.

It is necessary to load migration parameters, so only an information about supervisor IP is loaded as a string and the rest is loaded as and objects. It is beneficial to work with objects instead of identifiers because it allows to execute commands directly without any additional parsing.

Net::SSH client library is used for connection to virtual machine under test and supervisor. I have implemented authentication with keys because it is necessary to provide password-less login for backend service. It is possible to implement password authentication just by editing client library configuration, but I wanted to avoid storing any password in source code.

Migration process can be divided into 3 stages. First stage is preparation for migration, second stage is migration and third stage is migration verification and reporting. Tasks must be carried out sequentially because next task always depends on previous one.

First task after loading migration information is clearing previous measurement session. Established session between packet generator and packer receiver can become stale if it was not terminated successfully during previous migration. Packet generator and receiver should be terminated after virtual machine migration, but it may stay running in case of unexpected backend error or unclean shutdown. Command 5.1.3 is executed on VM and supervisor just to be sure there are no stale sessions. This command is optimized for agent.rb and need to be adapted to work with another measurement tools.

Figure 5.1.3: Clear stale sessions command

```
KILLPID=$(ps aux | grep -v grep | grep agent\.rb | grep ruby \
| xargs | cut -d' ' -f2); if [ -n "$KILLPID" ]; then \
kill -SIGINT "$KILLPID"; fi
```

Next action performed during first stage is migration to source host. Source and destination hosts are loaded from measure definition and migration must be performed exactly from source to destination, so VM need to be running on source host. Unmeasured migration to source host is requested during this stage if VM is not already running on right host.

Last task in preparation stage is to run agents. It is necessary to start agents on VM and supervisor and both agents must stay running after SSH disconnection. It is a bit tricky to run Ruby script in remote machine in subshell, because normal behavior it is terminate process after disconnection. I am using `screen` software which is able to run detached processes. However situation is even more complicated due to different Ruby installation methods on VM and supervisor. Virtual machine uses standard Ruby version installed via package manager by command `apt-get install ruby`, so it easier to run agent.rb because it do not require interactive shell. Rbenv[1] is used to install Ruby on supervisor because agent.rb in receiver mode requires newer version than provided by package manager. Rbenv is initialized in file ~/.bashrc so it is necessary to run all Ruby script in interactive shell. Parameters for agent.rb can be found below in table 5.3.5.

[1] https://github.com/sstephenson/rbenv

Figure 5.1.4: Run remote agents command

```
###  generator - VM
screen -d -m /bin/bash -c '~/themis/agent.rb "generator" \
#{supervisor} #{5000 + (id % 1000)} #{measure_definition.bandwidth}M'

### receiver - supervisor
screen -d -m /bin/bash -li -c '~/themis/agent.rb "receiver" \
#{supervisor} #{5000 + (id % 1000)} \
#{measure_transfers_upload_url(:measure_session => id, :format => 'json')}'
```

Known problems

Sessions are not atomic because asynchronous API is used and many sequential actions need to be performed. Orchestrator works in "best effort" manner, so migration request is refused sometimes. It is usually caused by temporary unknown VM state. This behavior can not be solved in application so this kind of session is just marked as failed and next session is started.

Another problem is virtual machine stuck in migration state. It is caused by hypervisor error (usually deadlock). Orchestrator keeps asking about virtual machine state but never gets an answer, so virtual machine is stuck in actual state. It is necessary to fix this error manually in orchestrator, because application will get stuck in waiting for migration to finish. It is necessary to fix faulty hypervisor, delete VM and recreate it. Measurement task will continue after VM under test is back in running state.

Most serious problem is caused by zombie VMs. Zombie is virtual machine running on hypervisor, although it should not. Even worse is that OpenNebula orchestrator does not display zombies in list of virtual machines and it is possible to deploy virtual machine with same id on different host. This situation is depicted in figure 5.1.5. It is obviously not possible to migrate *one-247* between hosts, because it already exists on both of them. I am working on modification of OpenNebula scheduler to stop deployment of virtual machine in case of there is zombie with same id.

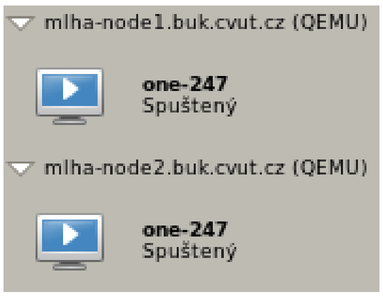


Figure 5.1.5: Zombie VM

All of these errors are cause by incorrect orchestrator behavior, so it is necessary to solve them in orchestrator. Application will wait for problem to be fixed or labels session as failed.

5.1.3 Transfer

Class MeasureTransfer is used to store information about transfers and it is actually parsed output from packet receiver. There are no user editable fields because objects are uploaded via API. Model parameters are described in table 5.1.3.

Themis application use iperf to measure packet flow, so MeasureTransfer class is tailored for output of iperf. Adapting to another measurement software is fairly easy because just one database migration script and small model changes will do the job.

Table 5.1.3: MeasureTransfer parameters

Parameter	**Required**	**Type**	**Edit.**	**Notes**
measure_session_id	yes	reference	no	reference to MeasureSession
timestamp	no	timestamp	no	
time_relative	yes	float	no	
jitter	no	float	no	
datagrams_transfered	no	integer	no	
datagrams_lost	no	integer	no	
bits_transfered	no	integer	no	
bits_lost	no	integer	no	
bandwidth	no	integer	no	bits per second

I it necessary to provide interface for automatic uploading MeasureTransfer objects. URL for uploads is /measure_transfers/:measure_session(.:format) and it is routed to measure_transfers#upload. This URL need to be passed to packet generator.

First task is to load superior MeasureSession object and check whether this object actually exists. Transfers without session is not valid and can no be saved.

Upload action of MeasureTranfers controller parses input objects received in JSON format and saves them into database. However it is necessary to make a few changes to every object. Iperf uses different format of timestamp than database and it does not support time zone, so each timestamp received from iperf need to be parsed, converted to native DateTime object and merged with timezone data.

Parsed DateTime object is used to calculate time_relative which is time between current time and start of migration session. Relative time is used to align migrations for graphing and exports.

Rails framework implements CSRF prevention mechanism, so it is necessary to create an exception for upload action. I have added exception to ApplicationController for all request in JSON format.

5.2 Virtual machines

Virtual machine, used for migration testing, needs to be prepared first. It is possible to migrate every virtual machine, but agent.rb wrapper need to be present and working to be able to establish measurement session. Agent is also used to export results from supervisor to the backend.

Agent.rb is expected to exist in path ~/themis/agent.rb and it is already saved here in prepared images. It is also necessary to install Ruby interpreter. I have used package available in distribution for VM under test and rbenv for supervisor.

Virtual machines are save on attached disc. Both are already contextualized for OpenNebula, and agent.rb with Ruby is installed. These images can be directly imported into OpenNebula image datastore and used in templates.

Table 5.2.4: Virtual machines parameters

Parameter	**VM for migration**	**Supervisor**
Operating system	Ubuntu server 14.04.1 LTS	
Image size	5.9G	
Image type	raw	
Device prefix	vd	
Username	root	
Password	none, only SSH keys using contextualization	
Agent.rb	/root/themis/agent.rb	
Ruby version	1.9.3p484	2.1.3p242
Ruby install method	package manager	rbenv

5.3 Agent

It was necessary to develop a script wrapper capable to parse iperf output and upload results to backend module. This script is called agent.rb, it is attached on on the CD and will be published in project repository. It is written in Ruby to be compatible with rest of the project. Agent script is executed from backend in migration_session model in start method and SSH is used for remote execution.

Agent script needs to be available in virtual machine under test as well as in supervisor. However distribution is fairly simple because it is just a single file (agent.rb) with a few dependencies. This file can be distributed manually, which is not very usable for large or frequent deployment. Automatic deployment using OpenNebula's contextualization is much more efficient since orchestrator takes care of saving file into VMs. It is also possible to use configuration management system to upload this file and prepare environment to run migration. I have used Ansible to deploy agent.rb because it was necessary to update Ruby version.

There are two modes of running agent. Modes differ in packet generator parameters and required dependencies. Mode is determined by `ARGV[0]` parameter, which is the first parameter after filename. It is necessary to select correct mode and properly configure all parameters from table 5.3.5 because measure session can not be established otherwise.

Generator mode only creates packets and sends them to receiver so it requires nothing more than Ruby and iperf available. Receiver mode is more advanced and it uploads results to backend besides receiving packets. Receiver mode requires these dependencies:

- **net/http** used to upload results to backend using HTTP POST method

- **json** necessary to export data into JSON before sending
- **date** used to parse iperf timestamp and convert it into backend compatible format

Both modes use IO class to read pipeline output from iperf program. This class is part of Ruby core so it is not necessary to install it separately. Receiver dependencies can be installed using package manager or with gem utility using command `gem install net/http json date`.

Table 5.3.5: Agent.rb parameters

	Generator mode		**Receiver mode**	
`ARGV[i]`	Parameter	Example	Parameter	Example
0	Mode	generator	Mode	receiver
1	Destination IP	192.0.2.1	Listen IP	192.0.2.1
2	Destination port	5004	Listen port	5004
3	Bandwidth	1M	Upload URL	`http://backend/measure_transfers/23.json`

Agent is using legacy iperf version developed by NLANR/DAST, but it introduces several problems which must be resolved in agent.rb and measure session routine. I am going to adapt agent.rb for iperf3[2] which is new implementation developed by ESnet/Lawrence Berkeley National Laboratory. Iperf3 provides JSON output and probably will not suffer from problems presented below.

First problem is automatic session reestablishment. This occurs when running session is interrupted by a client and new session is initialized with the same receiver in short interval (less than few seconds). Iperf server joins new session with previous one which is not desired solution. This it the reason why there is 5 second interval inserted before generator restart.

Second problem is handling INT signal by legacy iperf. SIGINT is reserved for external interrupt and this signal is for example sent to process when Ctrl + C is pressed. Iperf catches this signal preventing user to accidentally stop measure session. I understand reason why this function was implemented but I think that is total nonsense to require two consecutive INT signals to quit program. I have solved this by trapping SIGINT and sending KILL signal to iperf before agent.rb exits. This is only one possible way to reliably stop running agent together with iperf.

5.4 Frontend

I have developed web interface for managing Themis application, because it is easier for users to interact with web interface then configure application using console. Although web interface was not main goal of this thesis, I have decided

[2]Available on `https://github.com/esnet/iperf`

Figure 5.3.6: Example of agent.rb and iperf commands

```
# agent in generator mode
./agent.rb "generator" 192.0.2.1 5004 10M
# expanded iperf command in generator mode
iperf --udp --interval 1 --time 3600 --client 192.0.2.1 --port 5004 \
--bandwidth 10M --format b

# agent in receiver mode
./agent.rb "receiver" 192.0.2.1 5004 http://backend/measure_transfers/23.json
# expanded iperf command in receiver mode
iperf --server --bind 192.0.2.1 --port 5004 --udp --interval \
--reportstyle c --format
```

to implement it to provide better information about running migration and simple interface.

Web frontend is created in respect with model-view-architecture of Ruby on Rails. Twitter Bootstrap[3] is used for user interface components.

There are three tabs in the main page:

- Themis
- Waiting jobs
- Measure definitions

First tab is just welcome page with elementary information. The most important information on this page is current version. Application is prepared to be deployed using Capistrano[4] so current running version is loaded from GIT repository.

Waiting jobs tab displays running and pending tasks. All long-running tasks need to be executed asynchronously with Delayed::Job, so task is first saved into the database and then executed by a worker after some time defined in the configuration. Pending and running jobs can be reviewed in this tab. However it is not allowed to perform any changes on running or pending task, because it could break relation between Delayed::Job and running processes.

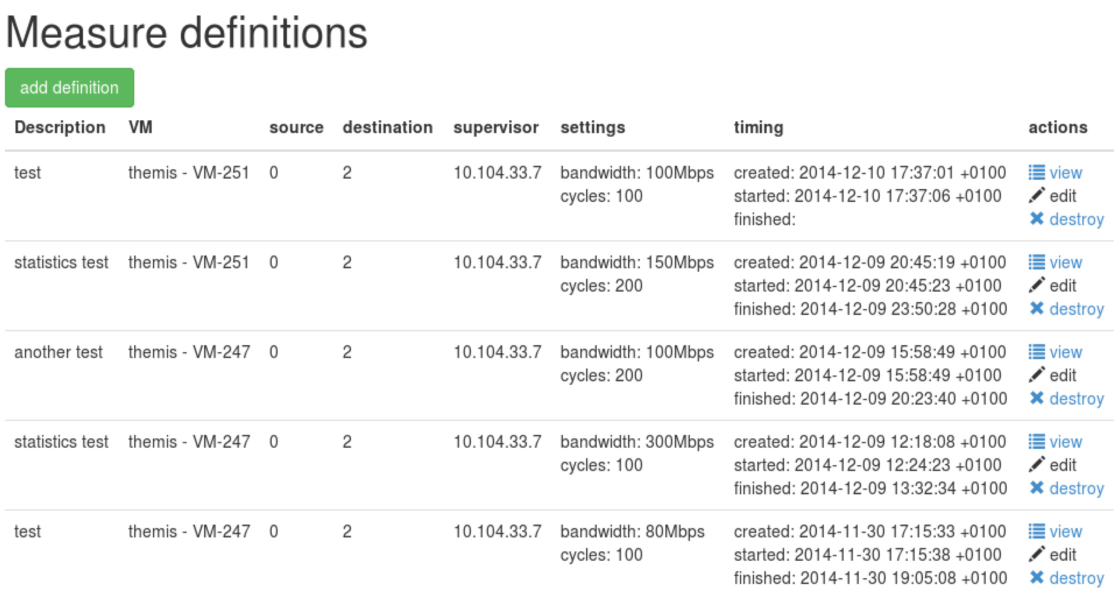
5.4.1 Definitions

Most important tab is Measure definitions because actions can be performed here. List of all definitions is displayed after clicking the tab, screenshot is in the figure 5.4.7. User accessible parameters from table 5.1.1 are displayed here. There are also basic actions as view, edit and destroy. Already started definition can not be edited.

I have decided to print source and destination host only as an id, because each name lookup takes one request send to the orchestrator. I think that listing id is sufficient, because user should already be familiar with OpenNebula hosts.

[3]https://github.com/twbs/bootstrap

[4]https://github.com/capistrano/capistrano

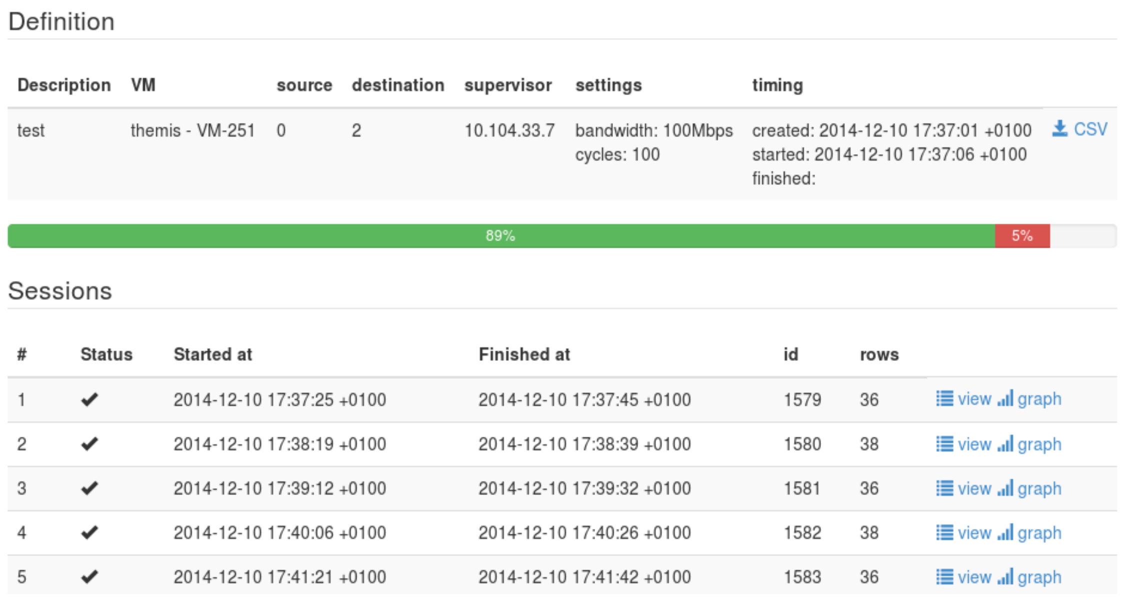

Measure definitions

add definition

Description	VM	source	destination	supervisor	settings	timing	actions
test	themis - VM-251	0	2	10.104.33.7	bandwidth: 100Mbps cycles: 100	created: 2014-12-10 17:37:01 +0100 started: 2014-12-10 17:37:06 +0100 finished:	view edit destroy
statistics test	themis - VM-251	0	2	10.104.33.7	bandwidth: 150Mbps cycles: 200	created: 2014-12-09 20:45:19 +0100 started: 2014-12-09 20:45:23 +0100 finished: 2014-12-09 23:50:28 +0100	view edit destroy
another test	themis - VM-247	0	2	10.104.33.7	bandwidth: 100Mbps cycles: 200	created: 2014-12-09 15:58:49 +0100 started: 2014-12-09 15:58:49 +0100 finished: 2014-12-09 20:23:40 +0100	view edit destroy
statistics test	themis - VM-247	0	2	10.104.33.7	bandwidth: 300Mbps cycles: 100	created: 2014-12-09 12:18:08 +0100 started: 2014-12-09 12:24:23 +0100 finished: 2014-12-09 13:32:34 +0100	view edit destroy
test	themis - VM-247	0	2	10.104.33.7	bandwidth: 80Mbps cycles: 100	created: 2014-11-30 17:15:33 +0100 started: 2014-11-30 17:15:38 +0100 finished: 2014-11-30 19:05:08 +0100	view edit destroy

Figure 5.4.7: List of all definitions

More information about definition can be displayed by following "view" action link on the right side. This page displays all information about selected definition, it's parameters, sessions and transfers. Screenshot is in the figure 5.4.8.


Definition

Description	VM	source	destination	supervisor	settings	timing	
test	themis - VM-251	0	2	10.104.33.7	bandwidth: 100Mbps cycles: 100	created: 2014-12-10 17:37:01 +0100 started: 2014-12-10 17:37:06 +0100 finished:	CSV

Sessions

#	Status	Started at	Finished at	id	rows	
1	✓	2014-12-10 17:37:25 +0100	2014-12-10 17:37:45 +0100	1579	36	view graph
2	✓	2014-12-10 17:38:19 +0100	2014-12-10 17:38:39 +0100	1580	38	view graph
3	✓	2014-12-10 17:39:12 +0100	2014-12-10 17:39:32 +0100	1581	36	view graph
4	✓	2014-12-10 17:40:06 +0100	2014-12-10 17:40:26 +0100	1582	38	view graph
5	✓	2014-12-10 17:41:21 +0100	2014-12-10 17:41:42 +0100	1583	36	view graph

Figure 5.4.8: Definition overview

Progress bar represents ratio between finished, failed and pending sessions. First green part is ratio of successfully finished sessions to all migrations, red part is ratio of failed sessions and the rest is percentage of pending.

There is table of all sessions under the progress bar. Session means one migration from source host to destination host together with network measurement. Status, time of start end finish, id and number of rows is displayed for each session.

Information about transfers during session can be displayed as a table or a graph. Graphing library is Chart.js[5] so it is necessary to use browser with JavaScript and HTML5 support. It is much slower to generate graph in browser than using MATLAB, because it is not optimized to work with huge datasets. Graphing in browser is intended to be used for quick overview and more complex visualization can be generated from exported CSV file using MATLAB or matplotlib.

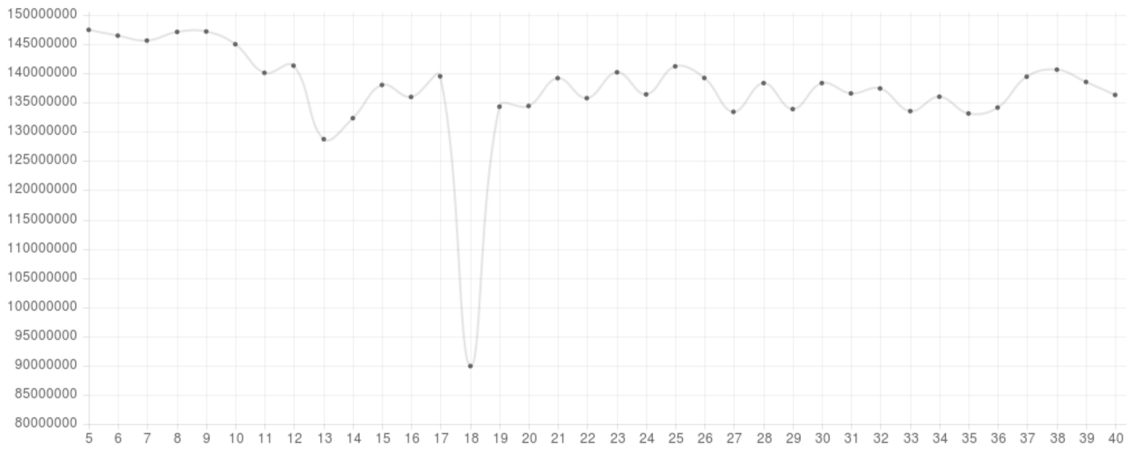


Figure 5.4.9: Bandwidth visualization

Frontend does not implement any kind of authorization and authentication because it is supposed to run on isolated network or as a part of existing system with authorization. Every client gets unlimited access to all migration data and can perform any action. It is necessary to negotiate rules in case of multiuser usage. It is not difficult to implement user access control, but it is beyond the scope of this thesis. Authentication library Authlogic[6] can be uses with authorization provided by for example CanCanCan[7].

[5]https://github.com/nnnick/Chart.js
[6]https://github.com/binarylogic/authlogic
[7]https://github.com/CanCanCommunity/cancancan

Conclusion

I have analyzed and compared networking and storage technologies used in distributed (cloud) datacenters. Virtualization is mentioned in the beginning because it is an enabling technology for the cloud computing. Cloud deployment and service models are compared and appropriate use cases are mentioned.

The biggest attention is dedicated to cloud computing, especially to networking, storage and orchestration technologies. Comparison is made by defining use cases and commenting advantages and disadvantages. It is not possible to rigorously decide on best solution because there are many use cases and each require individual approach.

Networking is essential part of datacenter design and legacy technologies are not able to fulfill current demand. Overlays techniques are supposed to provide required flexibility and network virtualization. VXLAN, STT and NVGRE is discussed. Hop-by-hop network virtualization is mentioned too, but it is tightly tied to SDN and SDN is still kind of sci-fi technology, because it is not widely supported. I have found out that load balancing is important topic connected with overlay networks and VM migrations so I have included load balancing into networking section too.

Orchestration is described in theoretical part and special attention is given to OpenNebula because it is used in practical part. I have explained principles of orchestration with special focus on virtual networks. Contextualization packages prepared to be used in practical part are presented.

I think that virtualization brings so many improvements to datacenters that legacy networking technologies are not able to keep pace. Most of technologies currently used in network layer were designed before virtualization era and it is the reason why they are to rigid to meet nowadays requirements. It is common to live migrate virtual machine nowadays but it was impossible in a few years before.

We are trying to build highly agile technology, like virtualized and distributed datacenter, on top of the legacy techniques. This approach is obviously not able to work well and there are two solutions possible.

First radical solution is to totally redesign current network stack and take current requirements into account. However it is very hard to design solution for all use cases with with respect to future usage. It also does not make sense from economical point of view because all network equipment need to be upgraded or replaced. SDN is, in my opinion, typical example of this technology since it brings amazing new features and it is usually can not be used on legacy devices, because hardware changes are necessary.

Second approach is to build new overlay network on top of existing network. This overlay network can provide additional functionality, but also brings some limitations caused by underlaying physical network. It is, for example, not possible to handle priority packets from overlay networks with special care in underlay network, because underlay knows nothing about overlay network. Another problem is BUM traffic because it is usually problematic to handle and multicast need to be implemented in underlay network. Overlay networks can immediately bring some significant improvements to datacenter networking, but there is a trade-off. I thing that overlays are appropriate temporary solution, but it would recommend to use hop-by-hop with SDN when available.

It is possible to migrate virtual machine with really minimal outage so VMs can be moved between hypervisors to optimize performance or minimize energy consumption. It is necessary to think about networking aspect of migration, because transfer degradation will definitely appear during migration. I have developed an application called Themis, which is capable to evaluate virtual machine availability during live migration. It combines network measurements, orchestration and data analysis.

Typical use case is migration of virtual machine with SLA. It is necessary to measure service disruption before migration because there are limits, for example for packet loss, defined in SLA. Themis can be used to migrate testing virtual machine and check whether service degradation during migration is acceptable.

Migration schema can be defined using console or web interface. Virtual machine availability is then evaluated according to schema. Repetitive migrations are supported as well as configurable bandwidth for packet generator. Application configures virtual machine via SSH, starts measurement session, request migration via orchestrator API and collects results. Whole process is fully automated, so no manual configuration is need.

Result can be viewed in browser or exported into CSV file. Sample measurement outputs are presented in appendix A.1. Source data were exported from Themis application and processed in MATLAB.

Fork of the application is used in Department of Electromagnetic Field, CTU FEE, for automatic measurement of wireless links and monitoring. Migration routines are not used and remote management is replaced by agent responsible for continuous packet generation. I am going to continue in development of this fork and new features will be merged into main branch.

List of Abbreviations

API	Application Programming Interface.
ARP	Address Resolution Protocol.
Bash	Bourne-again sh.
BUM	Broadcast, Unknown unicast and Multicast.
CAPEX	Capital Expenditures.
CPU	Central Processing Unit.
CRUSH	Controlled, Scalable, Decentralized Placement of Replicated Data.
CSRF	Cross-size Request Forgery.
CSV	Comma-separated values.
DJ	Delayed::Job.
DNS	Domain Name Service.
ECMP	Equal-cost multi-path routing.
FC	Fibre Channel.
FCP	Fibre Channel Protocol.
FDB	Forwarding Database.
GRE	Generic Routing Encapsulation.
HA	High Availability.
HTML	Hypertext Markup Language.
HTTP	HyperText Transfer Protocol.
IaaS	Infrastructure as a Service.
ICMP	Internet Control Message Protocol.
IO	Input/Output.
IOPS	Input/Output Operations Per Second.
IP	Internet Protocol.
IPMI	Intelligent Platform Management Interface.
IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6.
ISATAP	Intra-Site Automatic Tunnel Addressing Protocol.
iSCSI	Internet Small Computer System Interface.
JSON	JavaScript Object Notation.
KVM	Kernel-based Virtual Machine.
LVM	Logical Volume Management.
LXC	LinuX Containers.
MAC	Media Access Control.
MVC	Model-view-controller.
NAS	Network Access Storage.
NFS	Network File System.
NIC	Network Interface Card.

NIST	National Institure of Standards and Technology.
NVE	Network Virtualization Edge.
NVGRE	Network Virtualization using Generic Routing Encapsulation.
OCCI	Open Cloud Computing Inteface.
OPEX	Operating Expenditures.
OS	Operating System.
OSD	Object Storage Device.
OSPF	Open Shortest Path First.
OUI	Organization Unique Identifier.
OVS	Open vSwitch.
PaaS	Platform as a Service.
RADOS	Reliable Automatic Distributed Object Store.
RAID	Redundant Array of Independent Disks.
RBD	RADOS Block Device.
RIR	Regional Internet Registry.
RoR	Ruby on Rails.
RPC	Remote Procedure Call.
RTT	Round-trip Time.
SaaS	Software as a Service.
SAN	Storage Area Network.
SATA	Serial Advanced Technology Attachment.
SCP	Secure Copy.
SDN	Software Defined Networking.
SLA	Service Layer Agreement.
SOA	Start Of Authority.
SSD	Solid State Drive.
SSH	Secure Shell.
SSL	Secure Sockets Layer.
STP	Spanning Tree Protocol.
STT	Stateless Transport Tunelling.
TCP	Transmission Control Protocol.
ToR	Top of Rack.
TTL	Time To Live.
UDP	User Datagram Protocol.
UPS	Uninterruptible Power Supply.
URL	Uniform Resource Locator.
US	United States.
USB	Universal Serial Bus.
VLAN	Virtual Local Area Network.
VM	Virtual Machine.
VNI	VXLAN Network Identifier.
VPN	Virtual Private Network.
VPS	Virtual Private Server.
VSID	Virtual Subnet Identifier.
VTEP	VXLAN Tunnel Endpoint.
VXLAN	Virtual Extensible Local Area Network.
XML	Extensible Markup Language.

List of Figures

List of Tables

Bibliography

[1] Ondřej Celetka. IPv4 jako služba aneb jak síť zbavit dual-stacku. http://www.root.cz/clanky/ipv4-jako-sluzba-aneb-jak-sit-zbavit-dual-stacku/. [Online; retrieved 2014-09-30].

[2] IBM Corporation. Virtualization in education. http://www-07.ibm.com/solutions/in/education/download/Virtualization%20in%20Education.pdf, 2007. [Online; retrieved 2014-09-17].

[3] Davie and Gross. A stateless transport tunneling protocol for network virtualization (stt). http://tools.ietf.org/html/draft-davie-stt-06. [Online; retrieved 2014-11-12].

[4] G. Dommety. Key and sequence number extensions to gre. http://tools.ietf.org/html/rfc2890. [Online; retrieved 2014-08-10].

[5] Clark et al. Live migration of virtual machines. https://www.usenix.org/legacy/events/nsdi05/tech/full_papers/clark/clark.pdf, 2005. [Online; retrieved 2014-08-17].

[6] Farinacci et al. Generic routing encapsulation (gre). http://tools.ietf.org/html/rfc2748. [Online; retrieved 2014-08-10].

[7] Lasserre et al. Framework for data center (dc) network virtualization. http://tools.ietf.org/html/rfc7365. [Online; retrieved 2014-11-11].

[8] Mahalingam et al. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. http://tools.ietf.org/html/rfc7348. [Online; retrieved 2014-11-10].

[9] Sridharan et al. Nvgre: Network virtualization using generic routing encapsulation. http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-06. [Online; retrieved 2014-11-11].

[10] Weil et al. Ceph: A scalable, high-performance distributed file system. In *7th Conference on Operating System Design and Implementation*, 2006.

[11] A. Hammadi and L Mhamdi. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40, 2014.

[12] Chris Horne. Understanding full virtualization, paravirtualization, and hardware assist. http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf. [Online; retrieved 2014-08-20].

[13] Dustin Kirkland. Docker in ubuntu, ubuntu in docker. http://blog.docker.com/2014/04/docker-in-ubuntu-ubuntu-in-docker/. [Online; retrieved 2014-09-20].

[14] S.S. Kolahi, S. Narayan, D.D.T. Nguyen, and Y. Sunarto. Performance monitoring of various network traffic generators. In *Computer Modelling and Simulation (UKSim), 2011 UkSim 13th International Conference on*, pages 501–506, March 2011.

[15] T. Mell, P. Grance. The NIST definition of cloud computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. [Online; retrieved 2014-08-17].

[16] R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, 45(12):65–72, Dec 2012.

[17] Ellen Nakashima. Judge orders microsoft to turn over data held overseas. http://www.washingtonpost.com/world/national-security/judge-orders-microsoft-to-turn-over-data-held-overseas/2014/07/31/b07c4952-18d4-11e4-9e3b-7f2f110c6265_story.html. [Online; retrieved 2014-09-12].

[18] B. Radha and S. Selvakumar. Deepav2: A dns monitor tool for prevention of public ip dns rebinding attack. In *Advances in Recent Technologies in Communication and Computing (ARTCom 2011), 3rd International Conference on*, pages 72–77, Nov 2011.

[19] M. Townsley S. Tsuchiya, Ed. and S. Ohkubo. IPv6 rapid deployment (6rd) in a large data center. http://tools.ietf.org/html/draft-sakura-6rd-datacenter-04. [Online; retrieved 2014-05-30].

[20] S. Sarat, Vasileios Pappas, and A. Terzis. On the use of anycast in dns. In *Computer Communications and Networks, 2006. ICCCN 2006. Proceedings.15th International Conference on*, pages 71–78, Oct 2006.

[21] S. Srivastava, S. Anmulwar, A.M. Sapkal, T. Batra, A.K. Gupta, and V. Kumar. Comparative study of various traffic generator tools. In *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in*, pages 1–6, March 2014.

Appendix

A.1 Measurement samples

A.1.1 Measurement #23

100 Mb/s, 100 cycles, 6% failed

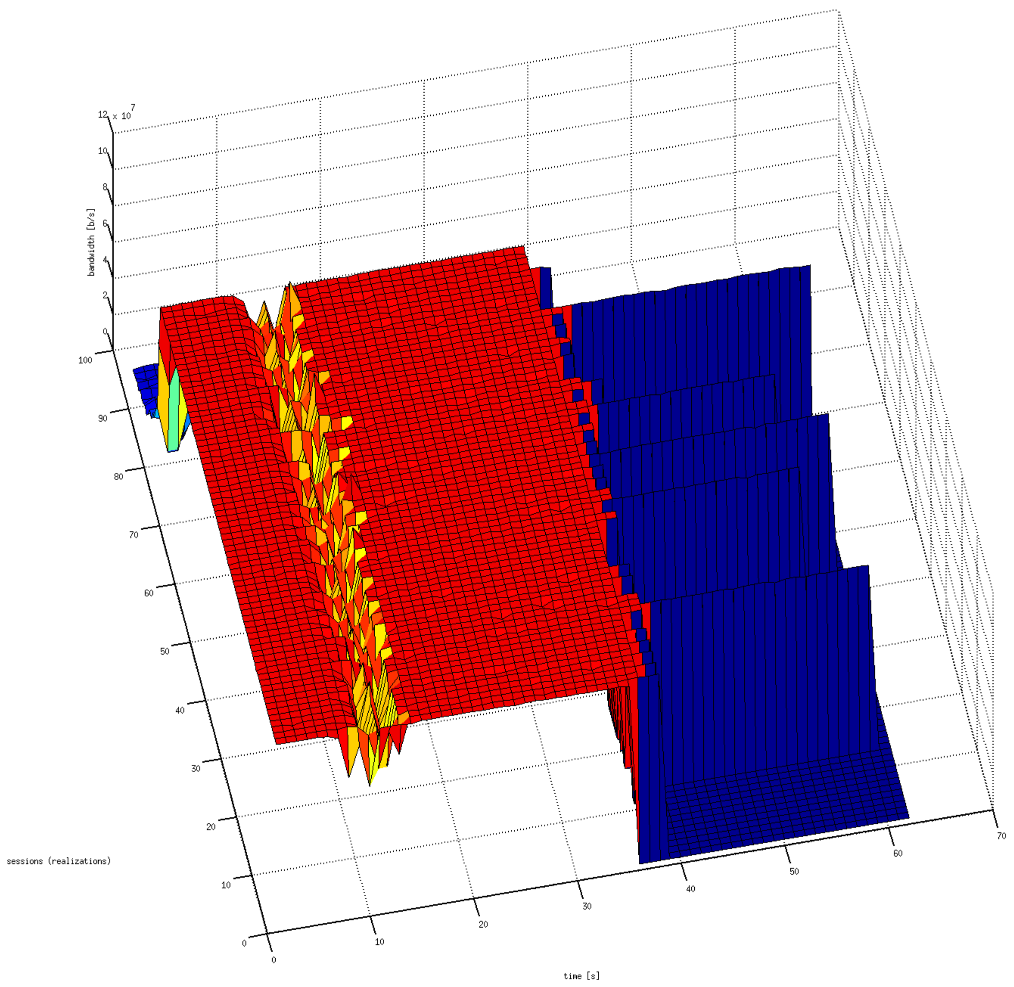


Figure A.1.1: #23, bandwidth [b/s]

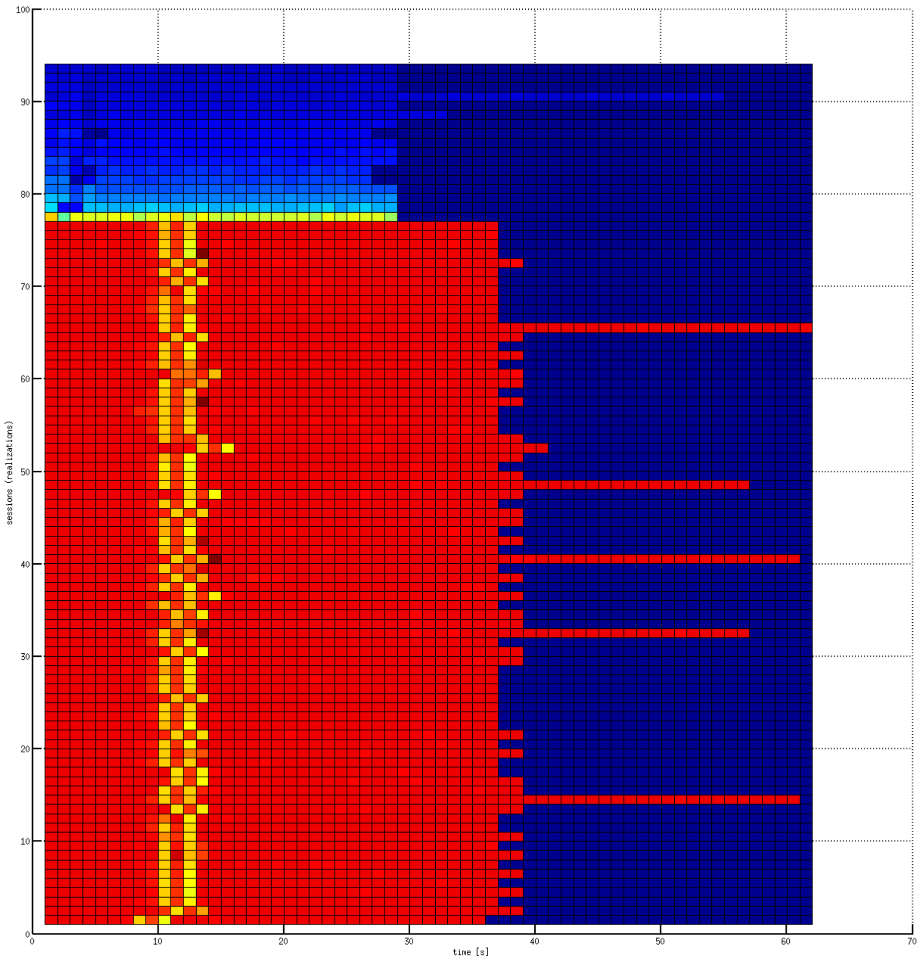
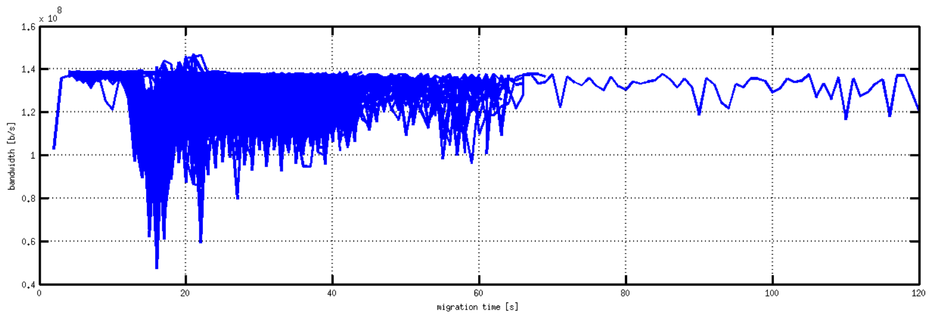


Figure A.1.2: #23, bandwidth [b/s], flat view

A.1.2 Measurement #26

140 Mb/s, 300 cycles, 6% failed



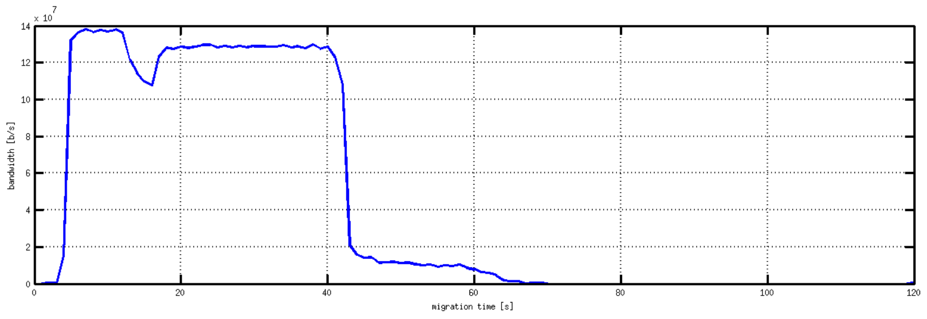


Figure A.1.3: #26, bandwidth [b/s]

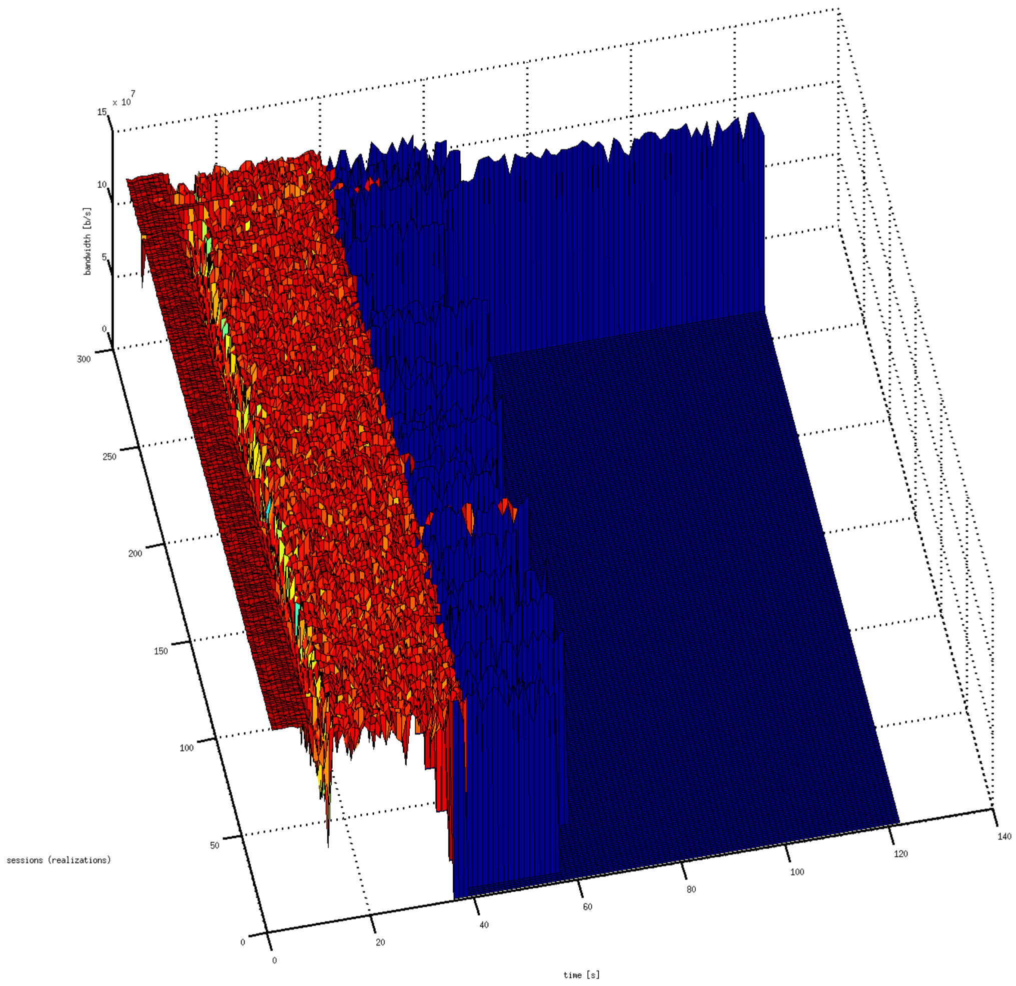


Figure A.1.4: #26, bandwidth [b/s]

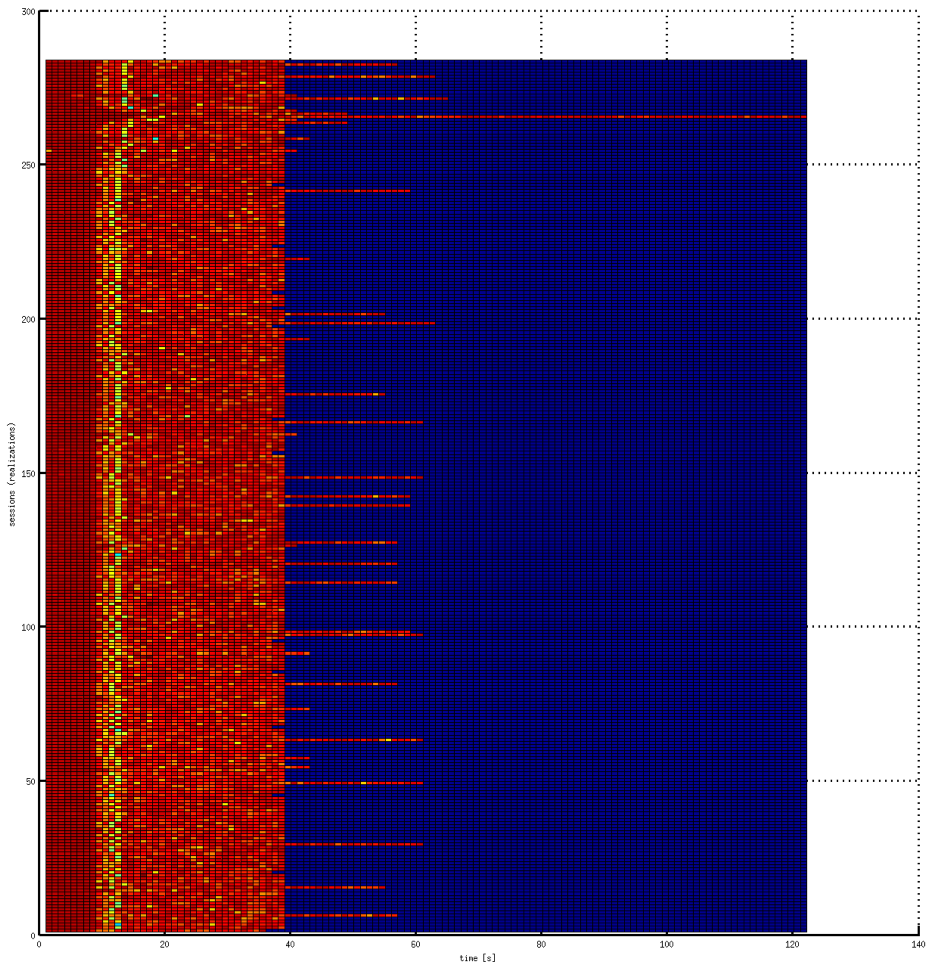


Figure A.1.5: #26, bandwidth [b/s], flat view

A.1.3 Measurement #27

80 Mb/s, 200 cycles, 10% failed

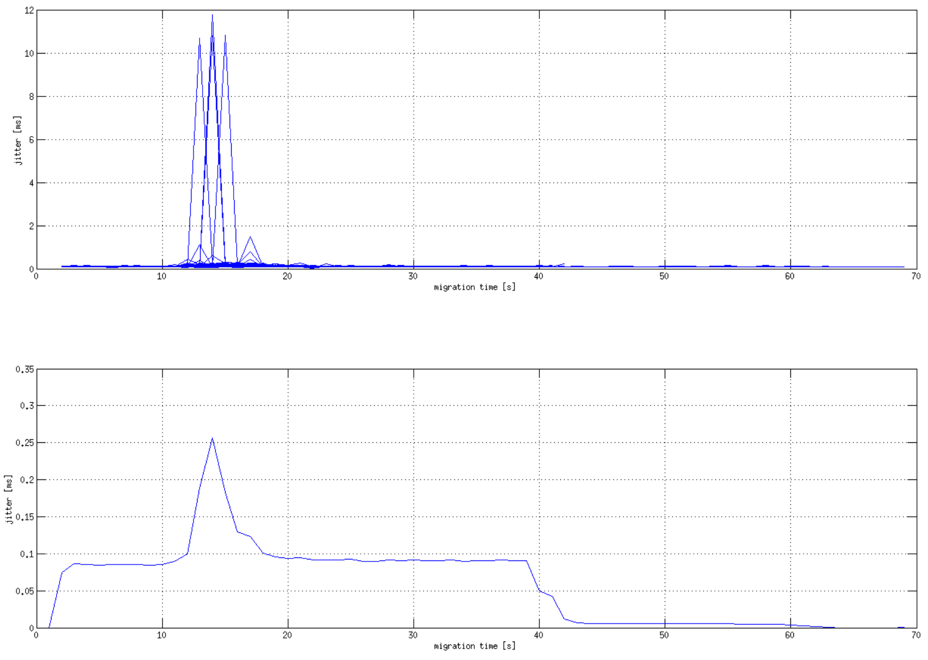
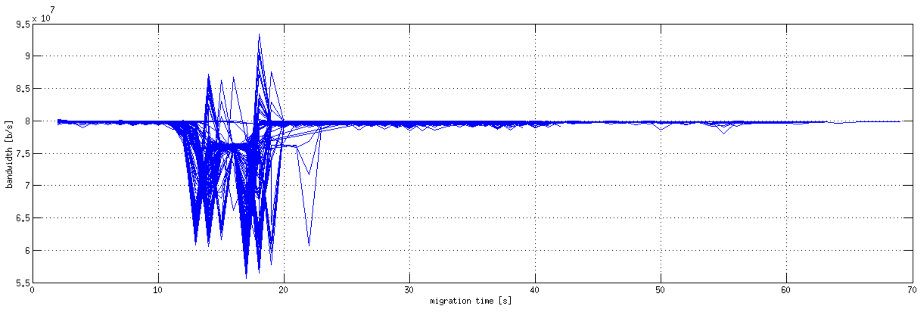


Figure A.1.6: #27, packet jitter [ms]



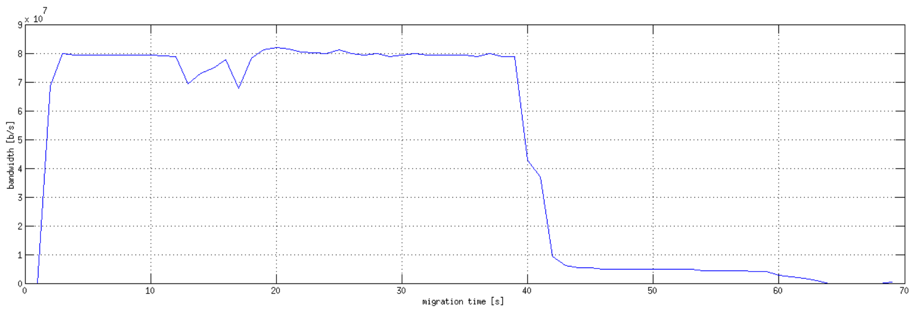


Figure A.1.7: #27, bandwidth [b/s]

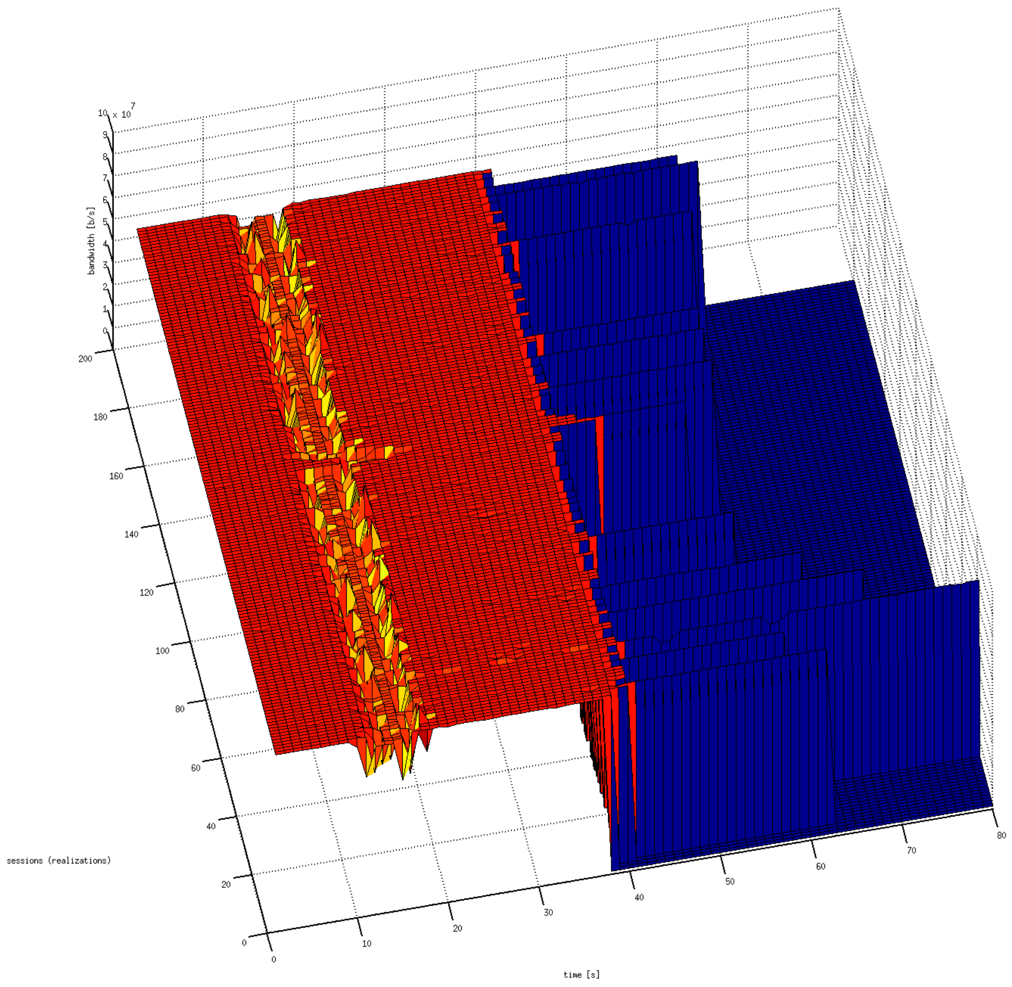


Figure A.1.8: #27, bandwidth [b/s]

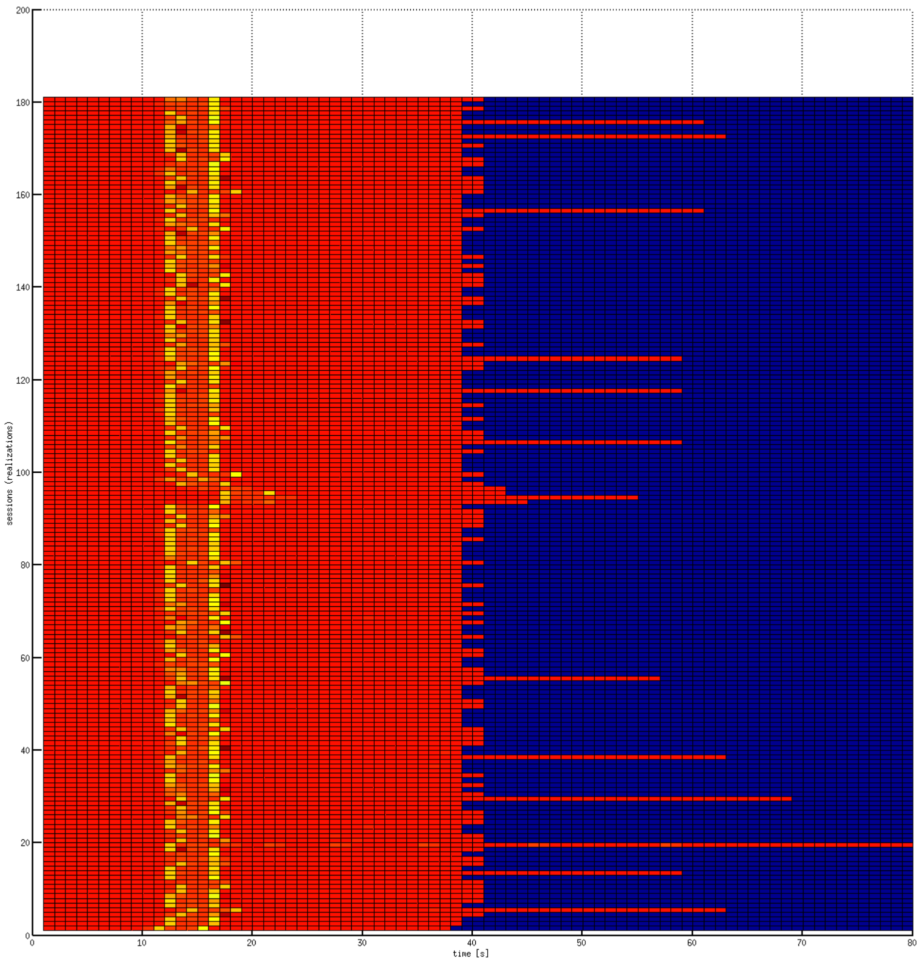


Figure A.1.9: #27, bandwidth [b/s], flat view