Mapping virtual paths in virtualization environments

Martin G. Metzker, Dieter Kranzlmüller

Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstraße 67
80538 München
{metzker, kranzlmueller}@mnm-team.org

Abstract: Quality of Service (QoS) in networks is an essential ingredient for providing attractive, reliable services to customers. In contrast to model layered protocol stacks, where resources are used exclusively, virtualization introduces a series of novel specific challenges, due to shared resources. This paper presents the model and mapping procedure of an approach for matching network QoS attributes with the needs of virtualization environments (VE). The key of our solution is an extended view on network topologies which adequately includes virtual components and links. We identify discernible types of network components and links, to enabe accurate descriptions of network paths through VEs. Our procedure is used to automatically describe (sub-)topologies yielding enough information to enable network QoS implementations.

1 Introduction

Over the last few years, virtualization with its plethora of applications has been changing the development of IT infrastructures in data centres. Within a virtualization environment (VE), *virtual infrastructures* (VI), consisting of *virtual machines* (VM) and virtual network components, are created, managed and deployed as needed for predefined services or simply as a vanilla topology of servers for arbitrary use through customers.

Specifications and management of VIs are abstracted from the underlying physical infrastructure and intentionally leave out details about the concrete implementation. When activated, VIs are placed within the VE, onto the providing physical components. This includes two important aspects:

1. Placing a VI within a VE usually expands its network links to network paths, including many physical and virtual components.
2. *Quality of Service* (QoS) attributes and requirements are formulated for abstract components and links, requiring refinement to match their placement.

Many aspects of VIs can be changed swiftly and on demand, especially the shape and participants of their respective virtual network topologies. The implied changes on resource consumption and consequently available network capacities, may lead to VMs starving each other for network resources. In extreme cases the services they provide are no longer

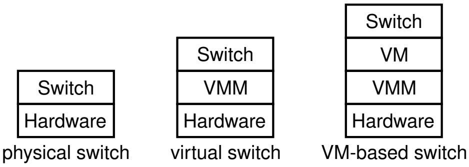


Figure 1: Three methods of implementing a switch

useful. In order to avoid such effects, network QoS management must be performed.

This contribution introduces a model and procedure to map VI network paths onto VEs. This enables automated mapping of network QoS requirements, making them an integral part of a management architecture for network QoS management in VEs. The procedure is intended to be automated, so that it can be applied for all VI paths. We prove our methodology by exemplary introducing the throughput QoS attribute to a Xen based VE.

2 Analysis and Approach

Looking at traditional network QoS with only physical components, network components and endpoints are physical resources with known capacities to provide services. In VIs, components, endpoints and links can be placed on dedicated physical hardware, or as virtual instances, implemented as a service on a shared platform.

Network virtualization yields a variety of network components, that are equivalent regarding their purpose and functionality according to the ISO OSI model, but may mandate different treatment for network QoS and QoS management. Figure 1 exemplary shows three conceptual implementations of a network switch. The physical switch, is implemented on dedicated hardware, whereas the virtual switch is one of possibly many virtual components realised by a *virtual machine monitor* (VMM), installed on a physical computer (*host*). VM-based switches are realised as part of a VM's operating system.

Sharing few physical components among many virtualized entities involves a multiplex which must be managed, in order to correctly manage network QoS. Often, multiplexes are hidden and we attribute this challenge being unsolved to the increasing complexity introduced by virtual components and shared resources.

Performing network QoS management concerns different (sub-)components and therefore requires an adopted management procedure. For each example in Figure 1, each layer represents (at least) a subcomponent of the switch, possibly requiring additional management when management on the switch is performed. This is a challenge for planning and provisioning, as the managed component "switch" becomes one of many possible concrete implementations. Further, in VEs, single components may be moved to another host after initial deployment (*migration*), which may result in a change of the implementation of "switch", and therefore how QoS must be implemented, enforced and monitored.

This gap between VI and component management can be met by requiring static configurations, eliminating a key feature of virtualization, or by a system with abstract description of VIs' QoS paths as *target configuration* that are continuously adopted to the VE's current state. We aim to realise the latter, for which we identify four main steps:

S1 determining involved components,
S2 mapping of high level QoS attributes,
S3 adopt to contestants and shortages,
S4 develop monitoring strategies.

This contribution focuses on S1 to determine all involved components of a VI network path and their types, so that QoS management can be performed correctly, with respect to the VE's current state.

The overall goal is an adoption of the layered approach first introduced in [JN04], that develops "concrete QoS parameters [... without] knowledge of the underlying [...] network conditions" as an intermediary result. In our adoption we aim to map specifications of QoS parameters for VIs onto the VE. Referring to the "layers [representing] very different features of services" in [JN04], our approach corresponds to developing mappings from the *application-layer* to the *resource-layer*.

We follow a bottom up approach and first analyse types of network components and links that differ in QoS management. The result is an enhanced topology resource-layer model, containing information on the degree of virtualization of components. This enables canonical implementation by the concrete technologies and components used to build VEs. The introduced procedure describes the refinement from application- to resource-layer.

2.1 Components

Figure 2 shows a classification of network components by locality of configuration, introduced in [MD10]. *Physical* components are fixed within the VE. While they may be replaced, for instance due to hardware failures, their place and role within the network topology remains unchanged. Local components, are placed on hosts. Their roles and function may be copied to other hosts but their state is relevant only for their current host. *Volatile* components are (equivalent to) VMs in terms of placement within the VE. Their state is invariant to placement within the VE, so they can be integrated at different positions in the (virtual) network topology.

This classification indicates how much immediate control a component has over the underlying physical resources. While physical components can map available resources to network performance accurately, local components contest for shared resources with other components and VMs. Volatile components are in general unaware of physical resources and how they are coordinated. Following through, the introduced classification separates how network QoS must be implemented by components:

- Physical components can directly implement network QoS.
- Local components must additionally handle resource contention. Effectively they implement network QoS with respect to the current load of their host.
- Volatile components implement some QoS themselves, but must delegate the task of ensuring that the required resources are available to the virtualization platform.

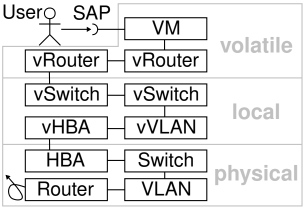


Figure 2: A conceivable path across network component classes and endpoints.

src ↓	dst → **p**	**l**	**v**
p	(a) p. link	(b) p. uplink	(d) pv-bridge
l	(b)	(c) E2E link	(e) v. link
v	(d)	(e)	(f) RDMA

physical **l**ocal **v**olatile

Table 1: Generic discernible direct link types by endpoint types

2.2 Links

As their physical role models, virtual network links are passive and network QoS management is performed at their endpoints. For any functional kind of network component, e.g. switch, classification by locality of configuration yields different types, concerning how resources are committed and network QoS management is performed. Linking components of different types, yields a set of discernible link types, summarised in Table 1.

To characterise these six link types, we look at how the endpoints and therefore the link are managed. The types *physical link*, *physical uplink*, *E2E link* correspond to links also found in strictly physical infrastructures, while the types *pv-bridge*, *virtual link*, *RDMA* bear virtualization specific challenges. We characterize the link types as follows:

(a) **physical link** There aren't any virtual components involved, "traditional" QoS [ea98].

(b) **physical uplink** Virtual components are connected to the physical network. The VMM can assign a virtual component resources like an OS scheduler to a process. For QoS management the virtual component can regarded as an endpoint in strictly physical topologies.

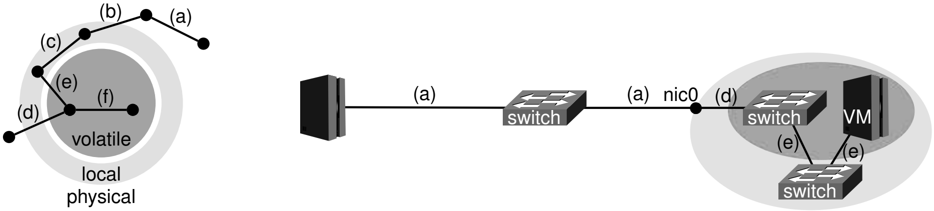
(c) **E2E link** A trivial end-to-end (E2E) link in the context of physical networks, as both endpoints are located on the same host.

(d) **pv-bridge** The uplink of a volatile component to the physical topology. The volatile component is generally not aware of resource contestants, but has been given an uncontested direct uplink, through the VMM. Even though the uplink is uncontested, the VMM must ensure the volatile component has enough other resources available.

(e) **virtual link** The usual way of connecting virtual components, mostly a volatile component and a local component. The VMM manages both endpoints and can allocate the resources to meet QoS demands.

(f) **RDMA** A theoretical direct connection between volatile components, without data ever crossing the VMMs domain. It is called remote direct memory access (RDMA) as this is most likely how it would be implemented, as both endpoints are placed on the same host and traffic between them is never handed to the VMM.

This constitutes a complete list of link types regarding the virtualization of endpoints, with which network paths through VEs can be described topologically. For (QoS) management



(a) Links from Table 1

(b) Enhanced topological view with virtualization host introspection

Figure 3: Resource layer views

links also differ in the OSI layers the endpoints implement and the concrete technology employed for virtualization. The above list is a generic template and specialized versions for the specific technologies employed in the endpoints must be derived. Technology and OSI layer characteristics for QoS handling yield a VE specific finite set of discernible link types, for which management functions must be available.

2.3 Resource-Layer topology view

At resource-layer all relevant information is described non-ambiguously [JN04]. We propose a topological view which groups network components by host and distinguishes between local and volatile components. This is consistent with describing links by their endpoints and components by their degree of abstraction from the physical hardware.

Figure 3(a) shows an abstract example featuring each link named in Table 1. By making the endpoints' domains (physical, local, volatile) explicit, the various link types are discernible without the labels. Figure 3(b) keeps the labelling for clarity purposes. The Figure itself shows an application example, connecting two servers over three switches, where each switch is one of the implementations introduced in Section 2. The physical switch, next to the physical host is correctly displayed as independent and the VM-based switch is displayed in a different domain as the local switch, yet within the same host.

Coloured/shaded areas group local and volatile components by host. All virtual components realised by the same physical host belong to the same area. With this illustration, volatile components are always contained within the local area. They are set apart optically, by using a different shade. The link between the two virtualized switches crosses domains, which means it must be a type-e link. The intersection of a link with the host boundary in Figure 3(b) is marked and explicitly names the used host's NIC, *nic0*. Which NIC is used determines which resources are used for the specific link, thus must be represented on the resource layer.

This enhanced topology view may include all identified component and link types. The increase in represented information meets the increased complexity and qualifies as resource layer for QoS layer partitioning following [JN04] as described in Section 2.

3 Refinement procedure

With the resource-layer established in Section 2.1, we can outline a generic procedure for mapping application-layer management information or tasks to the resource-layer. As an example we use the mapping of management information in the form of QoS attributes. This is achieved by iteratively refining application-layer links to eventually match the resource-layer topology. QoS attribute refinement is then performed based on reasoning on intermediary results. With each step the QoS attribute and its requirements are adopted to suit the sub-path which ultimately results in QoS requirements for each link at resource layer, which can then be implemented in a technology specific manner.

Network components are often characterized by the highest layer of the ISO OSI reference model they implement, e.g. routers implement layer 3. Combined with our taxonomy in Section 2.1 every component within a virtualization infrastructure can be characterized by the highest OSI layer it implements and its locality of configuration. The refinement procedure must therefore trigger two tasks: adoption of QoS attributes to the identified link segments and advancing the refinement process by detecting and resolving compound links. Our procedure develops the resource-layer topology (final topology) from the application-layer topology (initial topology), based on available management information. Algorithm 1 shows the core procedure refinePath, which uses two noteworthy subroutines: link (lines 8 and 22) and adoptQoS (lines 8, 12 and 22).

```
input : $T_{res}$ as the list of all links in the final topology, $P_{input}$ as endpoints of a path and the path's QoS attribute
output: $P_{refined}$ as a list of links in the final topology associated with an associated QoS attribute
1  $P_{refined} \leftarrow \epsilon$
2  $e_0, e_1 \leftarrow P_{input}.endpoints$
3  $P_{res-layer} \leftarrow$ pathAtOsiLayer($e_0$,$e_1$,$e_0.layer$)
4  while $P_{res-layer} \notin T_{res}$ do
5      $n \leftarrow$ neighborOf($e_0$,$P_{res-layer}$)
6      if $n \neq e_1$ then
7          if path($e_0$, $n$) $\in T_{res}$ then
8              $P_{refined}$.append( link($e_0$,$n$,adoptQoS($P_{input}$,$n$)))
9          end
10         else
11             $P_{sub} \leftarrow$ path($e_0$,$n$)
12             $P_{sub}.qos \leftarrow$ adoptQoS($P_{input}$,$n$)
13             $P_{refined}$.append( elementsOf(recursion into subpath))
14         end
15         $e_0 \leftarrow n$
16     end
17     else if $n = e_1$ then
18         $e_0.layer \leftarrow e_0.layer$- 1
19     end
20     $P_{res-layer} \leftarrow$ pathAtOsiLayer($e_0$,$e_1$,$e_0.layer$)
21 end
22 $P_{refined}$.append( link($e_0$,$e_1$,adoptQoS($P_{res-layer}$,$e_0.layer$)))
23 return $P_{refined}$
```

Algorithm 1: The refinePath procedure

The refinePath procedure uses the property of the OSI model, that every layer n uses the links and services provided by layer n-1, to break paths into sub-paths, starting at the layer of an input-endpoint, proceeding to lower layers. If a sub-path cannot be split up, it corresponds to a single resource-layer link, for which adequate management functions are available. When a path with a directly corresponding link in the resource-layer is found, the function link is called. Calling link is a request to apply the input-path's

QoS attribute to a resource-layer link. To actually apply the attribute, link uses topology information to identify the technologies used for the endpoints and the link type. With this information link can determine the correct management for the path segment.

The task of adoptQoS is provide an adopted QoS attribute to link, by tracking the performed path refinement. The concrete adoption procedure is specific to the QoS attribute. For instance, for a sensible implementation for an attribute “data rate”, adoptQoS would alter the attribute values to account for protocol header fields, when regarding data rate at varying OSI layers, or special implementations of virtualized links.

Besides accounting for implementation characteristics, adoptQoS tracks paths and subpaths, to consolidate requirements of multiple high-level links sharing the same resource-layer link. This happens when link is called multiple times for the same resource-layer link. Also, the information collected by adoptQoS is used to set up monitoring.

The result of the procedure are the application-layer links as resource-layer paths, where each path segment is a resource-layer link with a refined QoS attribute associated. The actual application of the attribute is resource allocation and deploying monitoring.

The implementation of links, resource allocation and QoS is technology specific. For example, in previous work [DgFKM11], we analysed that VMMs often use different metrics to model similar aspects of virtual and physical components. As QoS management heavily depends on the monitoring data obtained from the managed systems, QoS management must become ever more specific for the managed object. This can be seen as one of the main causes for the existing gap in QoS management, when introducing virtual components to IT infrastructures. The advantage of using the model and procedure introduced herein, lies in the generic structure and methodology for describing links and components. The model’s application results in a finite and sensible set of managed object types and corresponding management functions or strategies that allow for automated adoption, propagation and application of management information and functions from the high level view of virtual infrastructures to the concrete provisioning components.

4 Experiments

We apply our procedure to an exemplary VI with a single QoS attribute *data rate*. To illustrate the effect, we include a greedy “attacker” VM, which consumes resources to cause network performance degradation. With our procedure we identify the relevant components to perform QoS management to mitigate the attacker’s negative effect.

Figure 4 shows our test bed: a VM connected via a router to another VM, placed on our hosts xensrv02, xensrv02 and xensrv03, respectively. While xensrv01 and xensrv02 are connected via a physical switch, xensrv02 and xensrv03 are directly linked. All VI components are volatile and only local switches are employed. The resource-layer topology is augmented by the classifications of its eleven path segments.

In this example, the QoS attribute is implemented correctly, when all path segments deliver at least the required data rate of 200 MBit/s ($\approx$ 25 MByte/s). The requirement is valid at

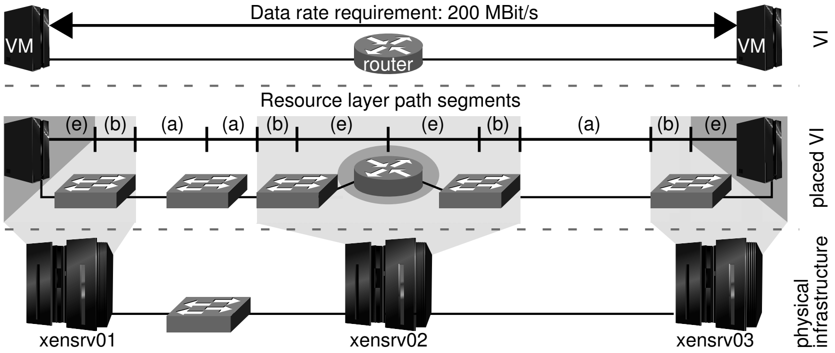


Figure 4: An exemplary VI mapped onto our test bed

application layer, and the data rate must be is adopted to correctly include the IP and Ethernet PCI. With standard Ethernet frames (up to 1500 bytes data), our use case specific `adoptQoS` evaluates ≈ 200.3 MBit/s for IP and ≈ 200.6 MBit/s Ethernet path segments.

When provided the two VMs as endpoints at layer 7, the procedure performs as follows:

1. Four iterations: OSI l7 → l3.
2. `adoptQoS` → 200.3 MBit/s
3. Twice: recursion into subpath

(a) Five iterations: OSI l7 → 2

(b) `adoptQoS` → 200.6 MBit/s

(c) `link` for all found segments.

The application of our customized `adoptQoS` and `link` functions yields the following configuration steps:

- traffic shaping inside the VMs, for all traffic addressed to the other server,
- a new data path on each local switch for each direction and
- a new CPU-pool on each host for the VI's component.

We test our resulting configuration by comparing data flows between the servers with different kinds of load on `xensrv02`. To create flows, we use the program *mbuffer* to eliminate potential side effects due to disc I/O.

Our measurements are illustrated in Figure 5. Without the "attacker" VM, we achieve a fairly constant data rate of 39 MByte/s. Under CPU or network load, a drop below the required 25 MByte/s is observed. Having applied the developed configuration steps, the QoS goal is met, and the performance degradation evaded.

5 Related Work

The management challenge addressed in this paper pertains to network QoS in VEs. The special characteristics are that network components and network path segments are provisioned alongside endpoints one the one hand, and on-demand placement of virtual components which can be moved between hosts after initial provisioning on the other hand.

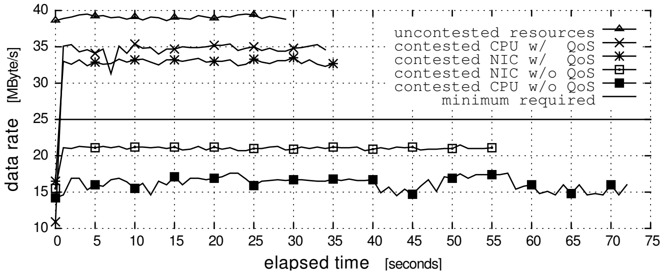


Figure 5: Achieved data rates

The gap between VI and component management has been addressed before. In [ea09c] and [ea09b] the authors suggest adaptive infrastructure management based on feedback loops. While this targets resources allocated to VMs, the network aspect is not handled specifically. The herein proposed approach can be used in such feedback loops and bridge the gap as described in this related work.

[ea09a] also identifies the need to adopt QoS attributes to technologies and (possibly hidden) network paths. This can be considered part of a solution, as they target networks only and the concept of volatile configurations does not fit their model.

Methods for refining resource allocations are still a topic for research, e.g. in [LN09]. This work provides the foundation for providing QoS links and paths in VEs. The recent study [ea13] analyses many such approaches and shows, that current VMMs don't "provide sufficient performance isolation [note: see [ea03]], to guarantee the effectiveness of resource sharing", which imposes a limit on efficiency and how accurate QoS can be enforced. The problem of performance of shared resources has been been recognised before [Rix08] and may motivate an automated implementation of our procedure to address such problems and trigger dynamic adoptions of QoS attributes if combined with some of the available other approaches.

6 Conclusions

The proposed refinement procedure requires management information about the VI, i.e. application-layer topology, and its placement within the VE, i.e. resource-layer topology. It then reconstructs the placement of network links and in doing to filters the available management information for all required link segments and DTEs. The thereby generated information also contains all logical abstractions and compositions that contribute to the final "visible" link in the VI. This is the important information, required to adopt management information to sensible rules that can be applied.

The suggested method of generating views and analysing paths enables the detection of hidden paths and logical links and the triggering of special treatment, if required. With this model as information base, generic rules for mapping VI QoS requirements to the resource-layer and the current state of a VE can be devised. This is a next step, out of the scope of this paper.

As part of an integrated management system for VEs, our approach is designed to allow substantial automation. It can be used to perform management on an existing VI, as demonstrated, or as part of the placement procedure. The specific `link` and `adoptQoS` functions can be built to provide feedback to acknowledge when VMs can be placed according to the specifications, or to force a different placement of VI components. Another aspect could be the actual planning in infrastructures where redundant network links allow for multiple network paths between hosts. To facilitate this heuristics can be added to the recursion stage of the procedure.

Acknowledgments The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of this paper.

References

[DgFKM11] V. Danciu, N. gentschen Felde, M. Kasch, and M. Metzker. Bottom–up harmonisation of management attributes describing hypervisors and virtual machines. In *Proceedings of the 5th International DMTF Workshop on Systems and Virtualization Management: Standards and the Cloud (SVM 2011)*, volume 2011, Paris, France, October 2011. Distributed Management Task Force (DMTF), IEEE Xplore.

[ea98] Paul Ferguson et al. *Quality of service: delivering QoS on the Internet and in corporate networks*. Wiley New York, NY, 1998.

[ea03] Paul Barham et al. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[ea09a] Karsten Oberle et al. Network virtualization: The missing piece. In *Intelligence in Next Generation Networks. ICIN 2009. 13th Int. Conf. on*, pages 1–6. IEEE, 2009.

[ea09b] Pradeep Padala et al. Automated control of multiple virtualized resources. In *Proc. of the 4th ACM Europ. conf. on Computer systems*, pages 13–26. ACM, 2009.

[ea09c] Qiang Li et al. Adaptive management of virtualized resources in cloud computing using feedback control. In *Information Science and Engineering (ICISE) 1st International Conference on*, pages 99–102. IEEE, 2009.

[ea13] Yiduo Mei et al. Performance analysis of network I/O workloads in virtualized data centers. 2013.

[JN04] Jingwen Jin and Klara Nahrstedt. QoS specification languages for distributed multimedia applications: A survey and taxonomy. *Multimedia, IEEE*, 11(3):74–87, 2004.

[LN09] J Lakshmi and SK Nandy. I/O Device Virtualization in the multi-core era, a QoS perspective. In *Grid and Pervasive Computing Conference, 2009. GPC'09. Workshops at the*, pages 128–135. IEEE, 2009.

[MD10] Martin G Metzker and Vitalian A Danciu. Towards end-to-end management of network QoS in virtualized infrastructures. In *Systems and Virtualization Management (SVM), 2010 4th Int. DMTF Academic Alliance Wksp. on*, pages 21–24. IEEE, 2010.

[Rix08] Scot Rixner. Network virtualization: Breaking the performance barrier. *Queue*, 6(1):37, 2008.