Component-based system for management of multilevel virtualization of networking resources

Robert Boczek Dawid Ciepliński

June 13, 2011

Contents

Chapter 1

Introduction

Chapter 2

Context

Chapter 3

Requirements analysis

Chapter overview

3.1 Functional requirements

3.1.1 Instantiation

3.1.2 Discovery

3.1.3 Accounting

3.2 Non-functional requirements

3.3 Underlying environment characteristics

3.4 General approach and problems it imposes

3.4.1 Load balancing / Deployment

3.4.2 Infrastructure isolation

3.4.3 Broadcast domain preservation

3.4.4 Constraints

Summary

Chapter 4

Solaris OS as a resource virtualization environment

Chapter overview

The chapter provides an overview of Oracle Solaris 10 and evaluates it as a platform for resource virtualization with respect to networking.

Three main components are presented and discussed: Containers - OS-level virtualization technology, Crossbow - network virtualization environment and the system's reasource access control utilities.

4.1 General information

resource management (allocation/assignment) accounting

4.2 Lightweight OS-level virtualization with Solaris Containers

The concept of lightweight (OS-level) virtualization is commonly known and supported by most (TODO really?) modern operating systems. The solutions are either integrated into the system's kernel (Solaris Containers, AIX Workload partitions, BSD jails TODO cite) or provided by third-party manufacturers as kernel patches and utility software (OpenVZ and LXC for Linux OS).

4.2.1 General information

Containers (or zones) were introduced as of Solaris Operating System 10. They provide a way of partitioning systems' resources and allow for isolated application execution environment.

There are two types of zones: global and non-global. Global zone is the default one and is used to execute applications as well as to administrate the system. Non-global zones can be created from within the global zone only. (TODO cite sag (how zones work)) A single operating system instance with one global zone can host as many as 8192 non-global zones.

Zones can be assigned system resources such as CPU power, the amount of random-access memory or disk space quota. Network isolation is also supported at two levels: basic isolation at the

IP layer and advanced fine-grained network isolation and virtualization with Crossbow technology. (TODO verify the sentence!)

Each non-global zone can run a different set of applications, with optional system call translation (*Branded Zones Technology* TODO cite sag (branded zones technology)) thus emulating different operating environment. The user is able to create a branded container with Linux system call translation and run Linux-specific applications within the container without code recompilation.

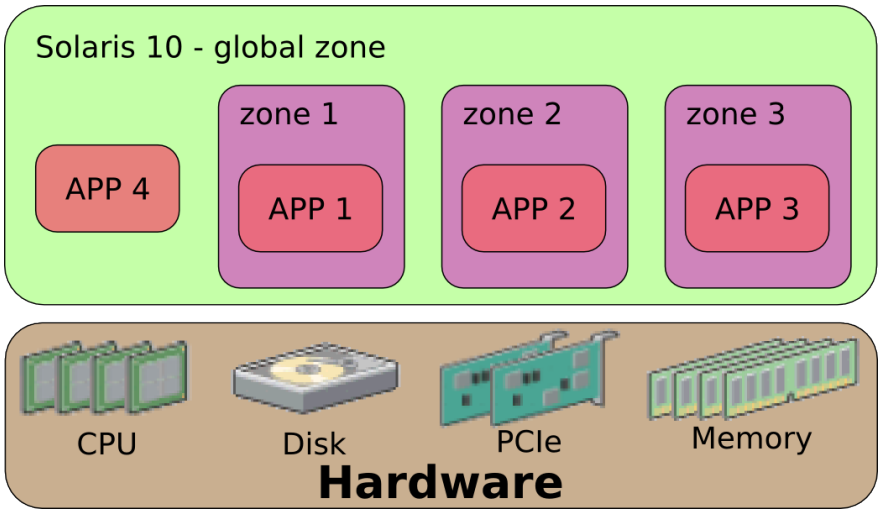

Figure 4.1: Solaris Containers hierarchy (source: wiki)

4.2.2 Container lifecycle

4.2.3 Isolation of processes

The Containers technology offers a high level of application security and isolation. This is accomplished by imposing software bounds on the resource usage and introduction of additional abstraction layer over hardware.

Every process and its children are bound to concrete zone and the assignment cannot be changed. Moreover, it is impossible for processes in distinct zones to monitor each other operation. They are not visible to each other and no standard interprocess communication can take place (standard==all except networking, TODO verify).

An application failure possibly affects only the processes in containing zone. (TODO easier recovery, independent container management)

4.2.4 Advantages of Containers technology when compared to non-virtualized environments

The architecture of Solaris Containers makes it a competitive solution as far as systems administration and operation efficiency is concerned. The technology, imposing negligible overhead [2], allows to perform tasks that would be impossible or very hard to perform if traditional setup is used. Examples of such tasks include dynamic resource assignment, instantaneous cloning and migration of systems between physical nodes.

The technology allows for running a number of isolated instances of operating system sharing CPU time, physical network bandwidth, filesystem contents and binary code. Sharing of these resources can greatly improve overall system efficiency and reduce the amount of occupied memory. The speed of network communication between different zones can also be improved thanks to „short-circuited" traffic (i.e. omitting of layers below IP in the OSI/ISO stack). The instances are able to execute applications with minimum overhead introduced mainly due to accessing commands and libraries through the lofs filesystem (TODO more about that) [2].

When using file system that supports snapshots (as, for example, Solaris 10's default ZFS, TODO citation), containers can be serialized (a snapshot of the file system can be taken) and sent over the network connection (or other means of data transfer) to another machine. There the zone can be restored and operate as a part of the host system.

Another important aspect of building the infrastructure with containers is resource access control.

flexibility and dynamic fine-grained resource allocation

The zones can be created and destroyed on demand, which allows for 1:1 zone:service mapping.

Resources (such as CPU, memory, network bandwidth) can be assigned dynamically without interrupting the container operation.

service consolidation -> one host with multiple containers

4.2.5 Virtual appliances

Virtual appliance is a pre-built, pre-configured, ready-to-run enterprise application packaged along with an optimized operating system inside a virtual machine (definition from [3]). Solaris Containers fulfil all the requirements (TODO what requirements?) to be used as virtual appliance building block.

The main problem virtual appliances can solve is the complexity and duration of application deployment process. In general, a service deployment can be described as comprising the following stages: preparation (learning the dependencies), pre-installation, installation and post-installation. With traditional (non-virtualized) approach, these stages have to be repeated every time a service is deployed.

preparation $\rightarrow$ pre-installation $\rightarrow$ installation $\rightarrow$ post-installation

Figure 4.2: Traditional application deployment stages.

Virtual appliance approach makes it possible to reduce deployment time significantly [3]. This is achieved by performing most of the deployment stages once and storing the configured environment in a virtual appliance. The appliance can now be published in publicly-available repository for actual deployment on a host system.

1. preparation, pre-installation, installation, post-installation
2. (virtual) appliance preparation
3. appliance publication
4. appliance retrieval and activation
5. configuration adjustment

Figure 4.3: Deployment process with virtual appliances. Stage 1 is executed once.

It is possible to prepare sets of virtual appliances containing traditional services (such as application servers, database servers or media servers) as well as highly specialized networking-focused appliances that can act as routers, firewalls or load balancers. These Virtual (Network) Appliances, together with other components provided by Solaris 10, can be leveraged to build fully virtual network topologies.

4.3 Crossbow - network virtualization technology

It is generally acknowledged that Crossbow was invented in China in 341 B.C but it was in middle ages when it earned its recognition. Very easy in use and simultaneously very effective. The Solaris Crossbow mechanism for QoS are just like real crossbow's very efficient in comparison to other existing QoS mechanism and this similarity indicates the project name origin.

4.3.1 Crossbow architecture

One of the most important condition in terms of network virtualization is that network traffic should be insulated between virtual machines. This kind of isolation can be achieved by having a dedicated physical NIC, network cable and port from the switch to the virtual machine itself. Moreover, switch must also ensure sustainability on every port. In every other case virtual machines will definitely interfere between each other. In a particular case when we have to share physical NIC between virtual machines the most promising solution is to virtualize NIC hardware and the second layer of the OSI/ISO stack where sharing is fair and interferences will be avoided. These approach was adapted in the Crossbow architecture in OpenSolaris OS. Traffic separation is achieved by fundamental blocks of new architecture which are Virtual NICs (VNICs) created by dividing NIC into many VNICs. A VNIC can be created over NIC or Etherstub (more about them later) and be dynamically controlled by the bandwidth and CPU resources assigned to it. The crossbow architecture has introduced fully paralized network stack structure. Each stack could be seen as fully independent lane (without any shared locks, queues, and CPUs) therefore network isolation is guaranteed. Key concept is hardware classification performed by the NIC over which VNIC was created. Each lane has a dedicated buffer for Transmit (Tx) and Receive (Rx) ring. In case when load exceeds assigned limit packets must be dropped as it is wiser to dop them then to expend OS CPU resources.

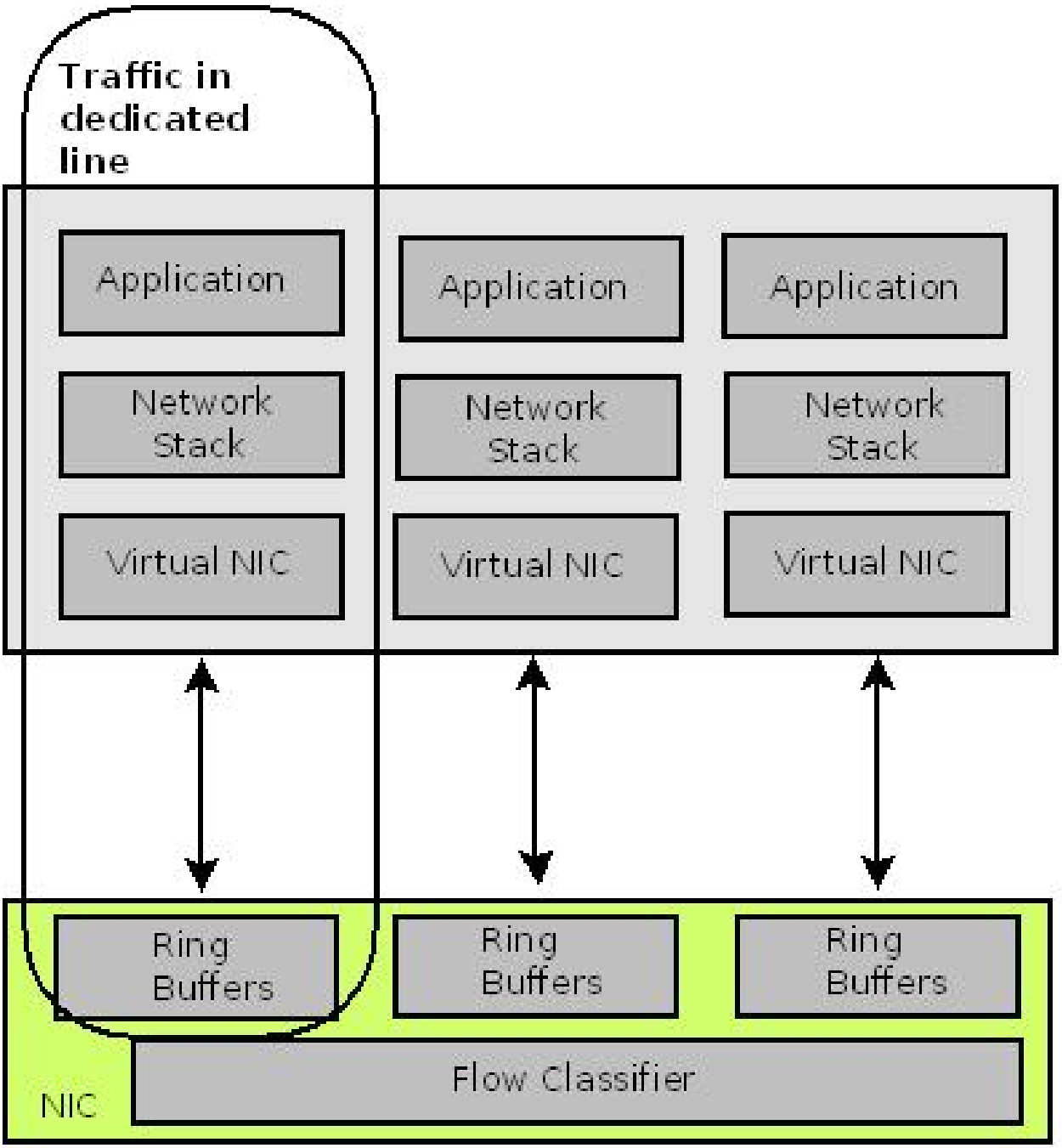

Figure 4.4: Dedicated lines in the Crossbow architecture

4.3.2 Virtualization lines

Virtualization is the most key component in the Crossbow architecture. Each lane consists some used to some concrete type of traffic. It usually would be composed of:

1. NIC resources(receive and transmit rings, interrupts, MAC address slots)

2. Driver resources(DMA bindings)

3. MAC layer resources (data structures, execution threads, locks)

A virtualization lane can be one of two types, hardware-based or software-based.

Hardware-based virtualization lanes

This type requires ability to partitioning resources from NIC. The minimum requirement is that a hardware-based lane should must have a dedicated receive ring. Other resources such as transmit lane can be exclusive or shared between lanes. Each virtual machine could have one or more lanes assigned and the incoming packets would be distributed among them based on even scheduling unless some administrative polices where created, such as priority or bandwidth limit.

Software-based virtualization lanes

In case when NIC runs out of hardware-based virtualization lane, receive and transmit rings may be shared by multiple VNICs. The number of software-based virtualization lanes also often called softrings is unlimited. The main disadvantage of software-based lanes is the lack of fairness and isolation which in fact is provided in hardware-based lanes. The received and sent rings may work also in mix mode, whereas some of the rings may be assigned to software and some may be assigned to hardware based lanes.

4.3.3 Dynamic polling

The Crossbow architecture proposed two types of working mode. Currently used mode is determined by traffic and load. Under low load, where the rate of arraving packets is lower than time of packet processing, lane works in the interrupt mode which means that receive ring generates an interrupt when new packet arrives. However, when the backlog grows, the line switches to dynamic polling mode in which a kernel thread goes down to the receive ring in the NIC hardware to extract all outstanding packets in a single chain. Key aspect is that every virtualization lane works independently and transparently from each other. Usually only three threads are used per lane:

1. Poll thread which goes to the NIC hardware to get all packet chain

2. Worker thread which is responsible for protocol processing (IP and above) or delivers packets to virtual machine. Thread performs also any additional transmit work which is a natural requirement some concrete protocol, such as processing TCP packets that require sending ACK packets.

3. Transmit thread that is activated when if packets are being sent after transmit side flow control relief discharge, or after retreiving transmit descriptor. Application or virtual machine can transmit any packets without performing queuing because of flow control or context switching.

4.3.4 Virtual switching

Virtual switches are always created implicitly when the first VNIC is defined under existing NIC and could never be accessed directly nor be visible by any user (even administrator).

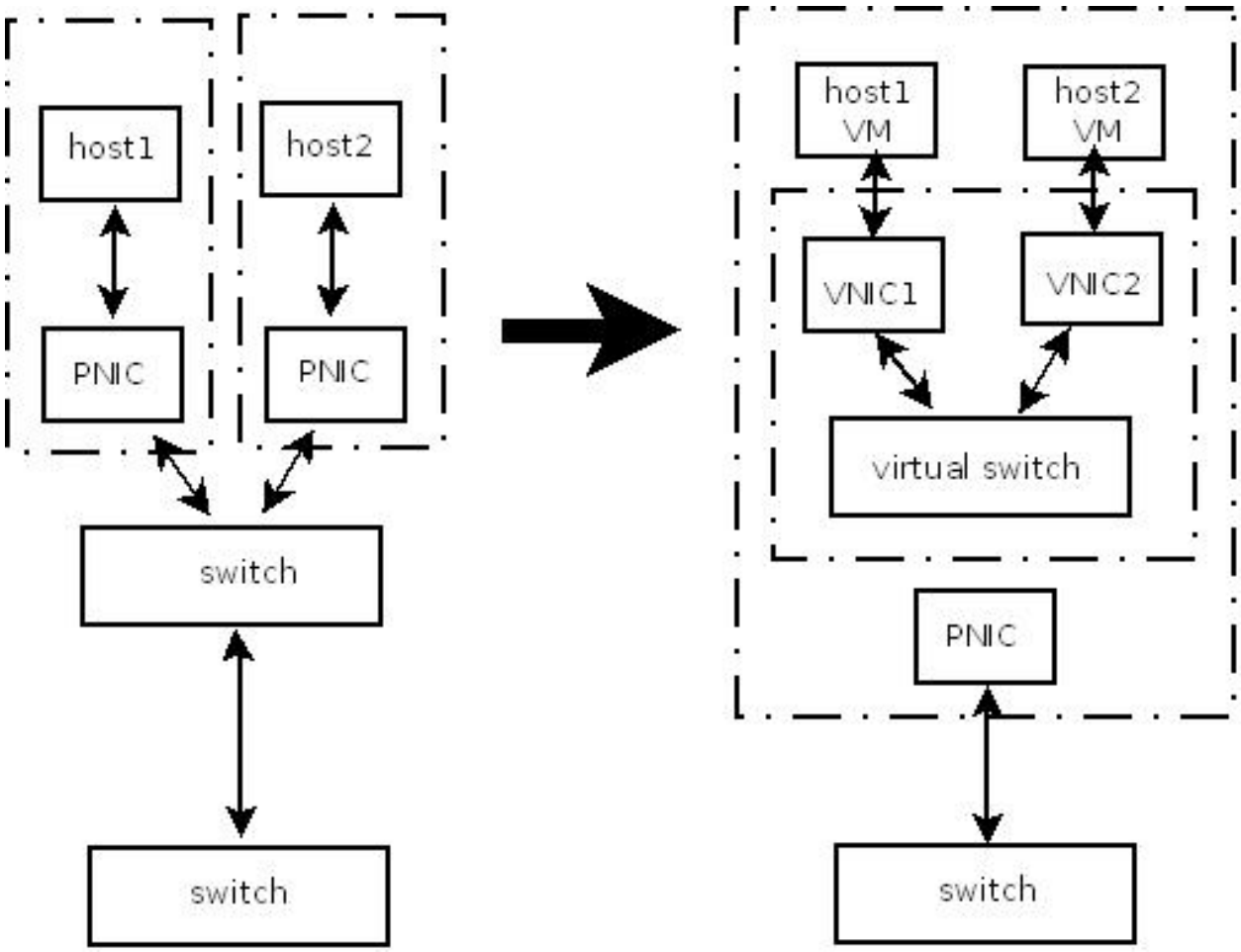

Figure 4.5: Mapping between physical and virtual network building elements

Semantics assured by virtual switches are the same as provided by physical switches:

1. VNICs created on top of the same NIC can send ehterstub packets to each other

2. Broadcast packets received by the underlying NIC are distributed to every single VNIC that was defined on the top of this NIC

3. Broadcast packets sent by one of the VNICs is distributed to all VNICs defined on the top of the same NIC and to the NIC for further transmission as well

4. In terms of multicast network traffic multicast group membership is monitored and used for distributing packets to appropriate VNIC

Connectivity between VNICs is available only when they were defined on the top of the same NIC.

4.3.5 Crossbow elements

VNics

Virtual NICs (VNICs) that are related to their own lane are the key element in crossbow architecture. There is no difference between NIC and VNIC in administration, they are all treated as data links. Every VNIC has an assigned lane and flow classifier which classifies received packets by VNIC's MAC address and sometimes by the VLAN tag. If created with a VLAN tag, protocols like GVRP or MVRP may be used to register the VLAN tag with the physical switches too.

In terms of sharing bandwidth, Crossbow enables administrative control of bandwidth for every single VNIC. The bandwidth of the link is implemented by regulating the periodic intake of incoming packets per dedicated lane. The network stack allows only as many packets as it was assigned to specific VNIC. The lane picks more packets when the next period begins. In case of regulating the speed of transmited bandwidth it is much easier as the network stack can either control the application that is generating the stream of packets or just drop the excessive amount of packets. These mechanisms are also used in flows QoS described and discussed later in this paper.

Etherstubs

As it was mentioned before, the MAC layer provides the virtual switching capabilities which allow VNICs to be created over existing physical NICs. In some cases, creating virtual networks without the use of a physical NIC is more welcomed than creating over physical NICs. In that case VNICs would be defined on the top of pseudo NICs. The Crossbow provides these kind of elements which are called Etherstubs. These components could be used instead of NICs during creation of VNICs.

Flows

Flows are additional instruments to allow easier network traffic administration. They might be used in order to provide administer bandwidth resource control and priority for protocols, services, containers. **flowadm** is the console command used to create, modify, remove or display network bandwidth and priority limits assigned to a particular link. Defined flow is a set of attributes based on Layer 3 and Layer 4 headers of the OSI/ISO model which are then used to identify protocol, service or virtual machine. Flows assigned to link must be independent therefore new one must is checked before adding new one newly created correctness is checked. Input and output packets are matched to flows in very efficient manner with minimal performance impact.

Examples

dladm is the admin command for managing of NICs, VNICs and Etherstubs. Below we present a few examples of creating VNICs, Etherstbus and how to assigned bandwidth and priority to theses elements.

1. # dladm create-vnic vnic1 -l e1000g0 - creates new VNIC **vnic1** over existing NIC **e1000g0**
2. # dladm create-etherstub ether00 - creates new Etherstub **ether00**
3. # dladm show-linkprop vnic11 - lists all properties assigned to **vnic11** link
4. # dladm set-linkprop -pmaxbw=1000 vnic11 - assignes 1Mbps bandwith limit to **vnic11** link

5. # dladm set-linkprop -ppriority=low vnic11 - assignes low priority to **vnic11** link

Here we have just presented some basic commands. For more examples see **man dladm**

flowadm is the admin command for managing of flows. Below we present a few examples of this command usage.

1. # flowadm show-flow -l e1000g0 - displays all flows assigned to link **e1000g0**
2. # flowadm add-flow -l e1000g0 -a transport=udp udpflow - creates new flow assigned to link **e1000g0** for all udp packets
3. # flowadm add-flow -l e1000g0 -a transport=udp udpflow - creates new flow assigned to link **e1000g0** for all udp packets

To see more check **man flowadm**

4.4 Resource access control

Summary

Chapter 5

The system architecture

Chapter overview

The Domain model and data flows section describes the transformations performed by the system's components in order to instantiate/deploy an object model. These include simple one-node instantiation as well as more complex multi-node instantiations.

5.1 High-level design

5.2 System components and their responsibilities

5.2.1 Assigner

5.2.2 Supervisor

5.2.3 Worker

5.3 Crossbow resources instrumentation

5.4 Domain model and data flows

Summary

Chapter 6

Implementation

Chapter overview

6.1 Implementation environment

6.2 Domain model transformation details

6.3 Low-level functions access

6.4 Building and running the platform

Summary

Chapter 7

Case Study

Chapter overview

7.1 Multimedia server

7.1.1 Scenario description

- similar to DiffServ (traffic classes, selectors, filters, priority, queuing)
- DiffServ doesn't specify anything virtual
- DSS and adaptive codecs
- 2 classes: VOD + streaming
- _ unicast _ vs multicast streaming
- access rules for resources of different quality
- enabling QoS for defined classes
- priorities + limiting the bandwidth (per user)!
- 3 users: 2 streaming, 1 VOD

7.1.2 Resource access requirements

7.1.3 Providing tunable and scalable virtual infrastructure

Summary

Chapter 8

Summary

Chapter overview

Bibliography [1] test.

8.1 Conclusions

8.2 Achieved goals

8.3 Further work

In terms of the future work there are many many improvments that might be implemented. Probably the largest component we'd plannned to implement was automatic resource assigner, which would run and perform automatic assigning resources to nodes that run under least load. This assigner with attached rule based system should gather data about the load on each node and based on that should decide what and where instantiate. What we have managed to complete is manual assigner, where you have to select on which node you would like to have your virtual resources created.

Bibliography

[1] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LaTeX Companion*. Second edition, 2004.

[2] Daniel Price and Andrew Tucker. Solaris zones: Operating system support for consolidating commercial workloads. 2004.

[3] Changhua Sun, Le He, Qingbo Wang, and Ruth Willenborg. Simplifying service deployment with virtual appliances. 2008.