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

Robert Boczek Dawid Ciepliński

June 14, 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. Special emphasis is put on the networking-related aspects of virtualization. Thus, the Solaris Crossbow technology is given its own section.

Section 4.1 contains introductory information about the system. A short historical note is presented and general description follows. Main components of the system are introduced and described.

Each of the remaining sections describe in more detail these parts of the operating system that are extensively used by the implemented system. Section 4.2 investigates the Solaris Zones technology. After defining the concept of zones, zone lifecycle model is presented, the achieved level of process isolation is described and the discussion of Zones advantages in comparison to non-virtualized environments follows.

4.1 General information

Oracle Solaris is a multiuser, multitasking, multithreading UNIX-like operating system [7]. Since its release in 1992 (as Sun Solaris 1), the system became one of the most popular environments supporting enterprise-class software. Nowadays, big corporations and companies as well as individual developers use it to do their business and deliver reliable and scalable services.

The Solaris OS provides unique set of tools that support virtualization of practically all types of resources at various levels. There is Logical Domains (LDOMs) technology for full virtualization and lightweight Zones, when all that is needed is the isolation of processes. Logical domains can be connected with complex virtual networks that are created with virtual switches (vsw) and virtual network devices (vnet) [6] and Crossbow can be used to enable lightweight and efficient networking for zones, exploiting capabilities of underlying hardware layer (network interface cards with virtualization level 1, 2 or 3 [2]).

Resource utilization can be managed with integrated administration tools. Resource access policies can be created with high level of granularity (per-process resource control) as well as in more general way (limiting resource access for LDOMs). Resource consumption can be subject of

monitoring and accounting. With extended accounting subsystem enabled, it is possible to capture detailed accounting data even for single processes. Gathered data include CPU usage, number of bytes received or transmitted per DiffServ or Crossbow flow and more.

As far as multiple physical machines are considered, there is also support for VLANs (Virtual Local Area Network). Thanks to VLAN tagging support, it is possible to build systems that guarantee the quality of service from the lowest levels up, even for services belonging to different systems and consolidated within single physical machine.

Figure 4.1: The variety of resources that can be virtualized and that Solaris 10 had integrated support for

As it can be seen, the Oracle Solaris 10 operating system is accompanied by vast variety of virtualization-supporting subsystems. This multiplicity and flexibility makes it a promising platform for service provisioning and building even more abstract architectures on top of it. The following sections describe selected aspects of the system in more detail.

4.2 OS-level virtualization with Solaris Zones

The concept of lightweight (OS-level) virtualization is supported by most (TODO really?) modern operating systems. The solutions are either integrated into the system's kernel and accessible as soon as it is installed (Solaris Containers, AIX Workload partitions, BSD jails [1]) or are provided by third-party manufacturers as kernel patches and utility software (OpenVZ and LXC for Linux OS). Because of the integration with and awareness of other system components, it can be expected that Zones have more potential in them than other virtualization methods.

4.2.1 General information

Zones technology was introduced as of Solaris OS 10. It provides a way of partitioning system resources and allows for isolated and secure application execution environment [5]. Solaris Zones, together with resource management functionality, constitute the Solaris Container 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. A single operating system instance with one global zone can host as many as 8192 non-global zones [5].

Zones can be assigned system resources such as CPU power, the amount of random-access memory or disk space quota (TODO that may not be true). Also, network isolation is supported at two levels: basic, at the IP layer, and network isolation and advanced virtualization with fine grained quality of service control using the Crossbow technology.

Each zone can run a different set of applications, with optional translation of system calls (*Branded Zones Technology*) thus emulating different operating environments [5]. The user is able to create a branded container with translation of Linux system calls and run Linux-specific applications in the container without code recompilation.

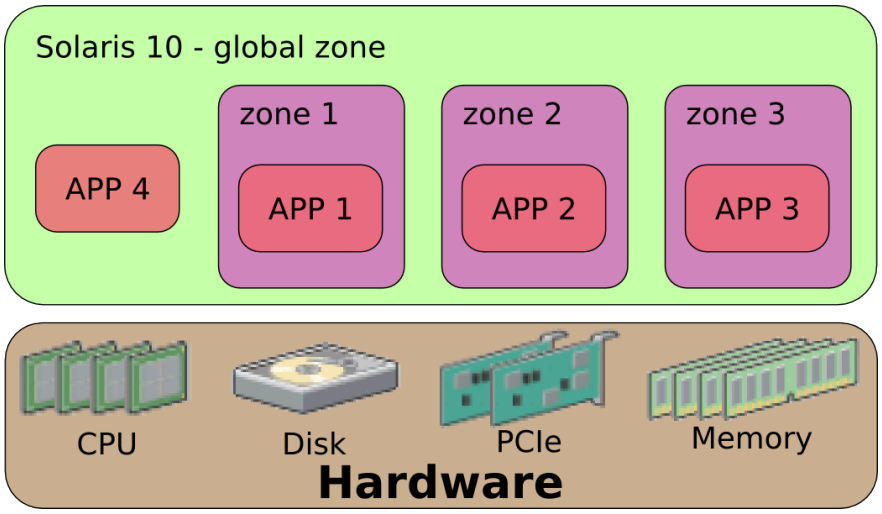

Figure 4.2: Solaris Zones high-level view (source: wiki)

4.2.2 Container lifecycle model

A model was created to describe the states a zone can be in and possible transitions. A non-global zone can be in one of six states: *configured*, *incomplete*, *installed*, *ready*, *running*, *shutting down or down* [5].

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 interprocess communication can take place, except for network-based one, if enabled by the administrator.

Because of the isolation, an application failure possibly affects only the processes in the containing zone. Assuming no interaction between processes in separate zones, the rest of the system remains intact and can operate normally.

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 [3], 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) [3].

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 control. The Solaris system makes it possible to define resource controls (rctls) at various level, also on per-zone basis. CPU shares and maximum number of lightweight processes are the resource control properties that can be set for a zone. This can be further extended by providing fine-grained properties at project, task and process levels [5]. The resource control process is dynamic - the assignments can be changed as the system is running, without interrupting the container normal operation. This of extreme importance if one is building high-availability systems (TODO define?).

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 [4]. Solaris Zones, together with other components of the Solaris OS, constitute a complete framework that implements virtual appliance approach to systems management.

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 on different machines.

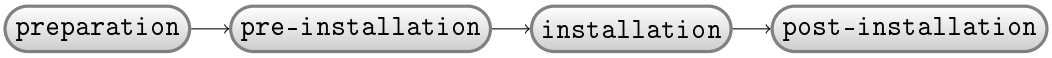

Figure 4.3: Traditional application deployment stages.

Virtual appliance approach makes it possible to reduce deployment time significantly [4]. This is achieved by performing most of the deployment stages once and storing the configured environment in a virtual appliance. The appliance can then be published in publicly-available repository for actual deployment on a host system.

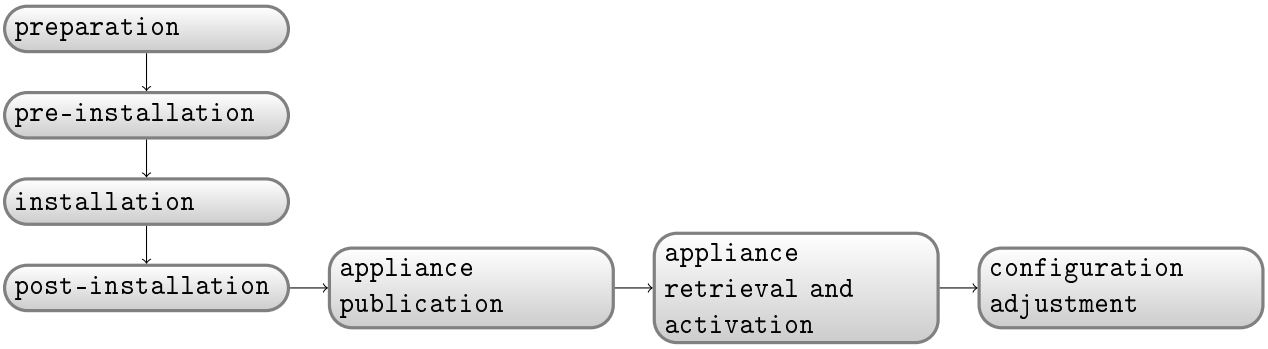

Figure 4.4: Deployment process with virtual appliances. Stage 1 is executed once.

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 crossbows, very efficient in comparison to other existing QoS mechanisms 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.

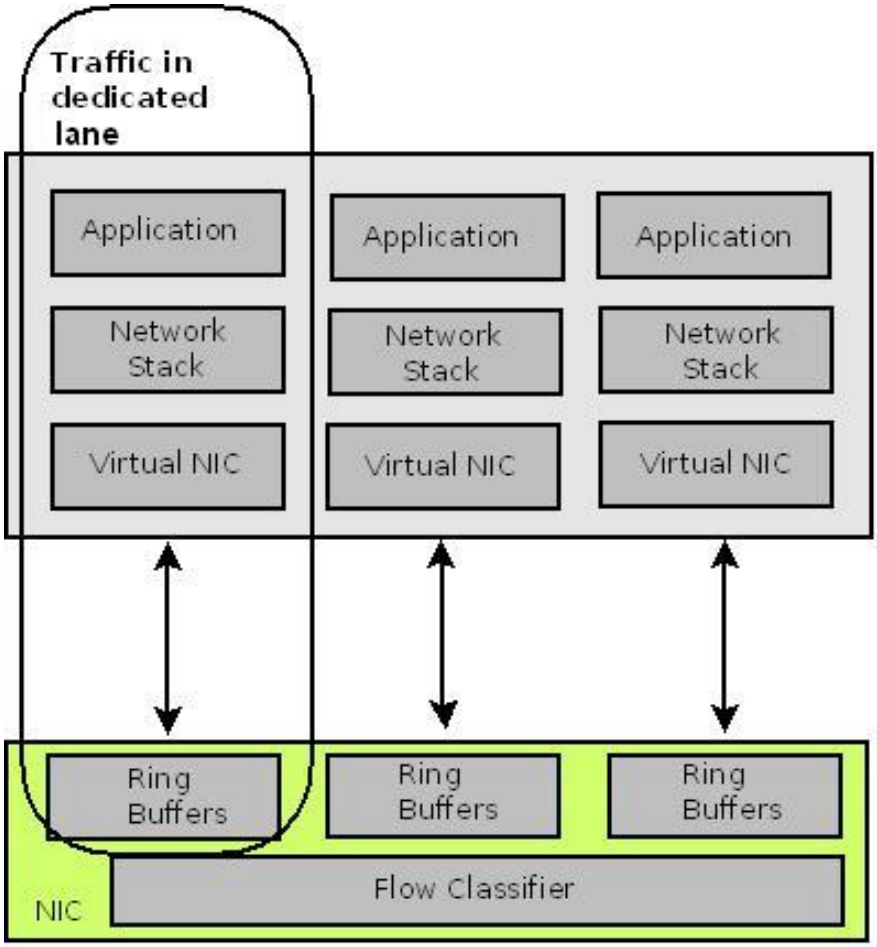

Figure 4.5: Dedicated lanes in the Crossbow architecture

4.3.2 Virtualization lanes

Virtualization lane is the most key component in the Crossbow architecture. Each lane consists some dedicated hardware and software that might be 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).

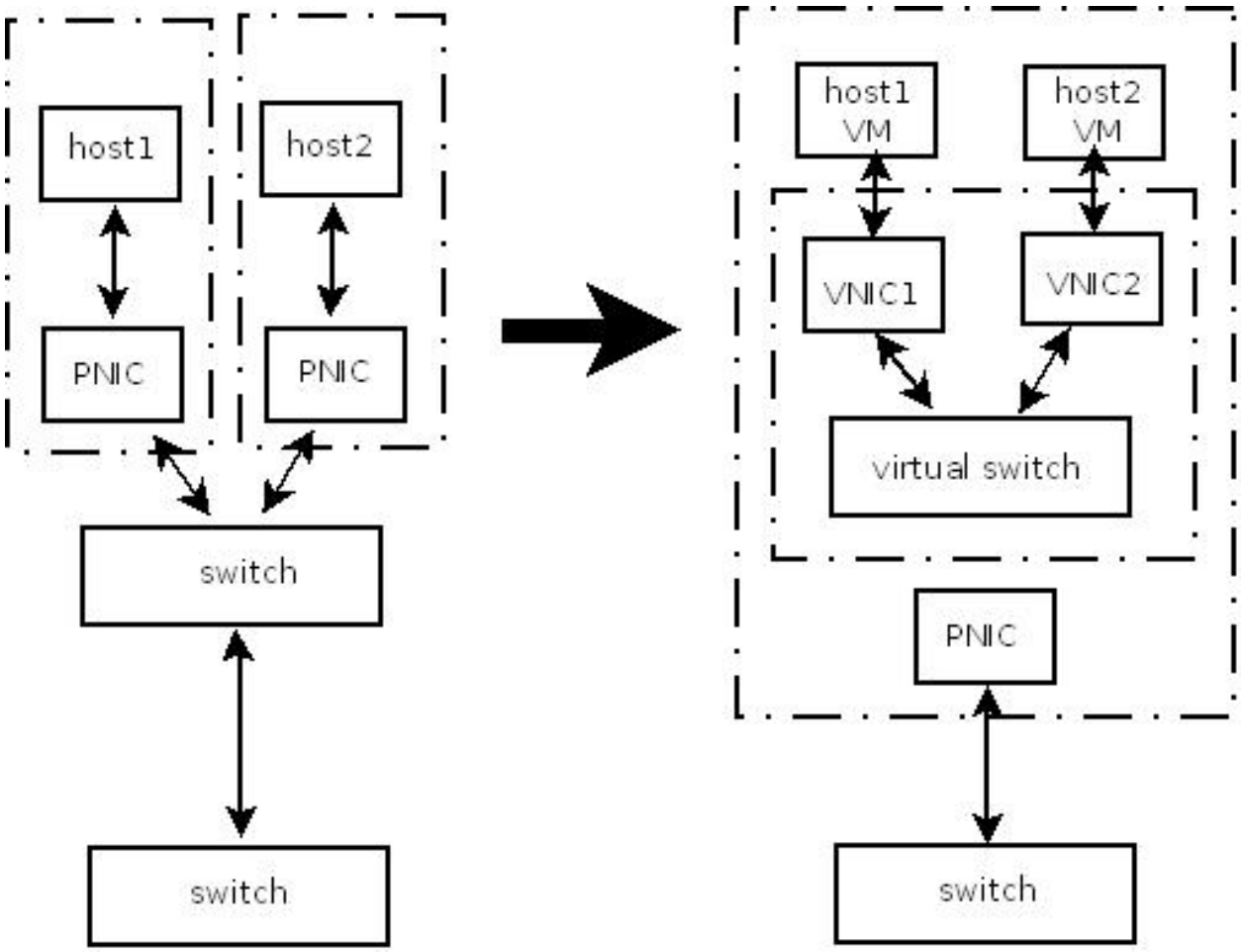

Figure 4.6: Mapping between physical and virtual network building elements

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.

4.4 Crossbow components

The Crossbow specification describes three major components: VNics, Etherstubs and Flows. This chapter will give an insight into theirs application and usage.

4.4.1 VNics

Virtual NICs (VNICs) each containing their own lane are the key element in crossbow architecture. There is no difference between NIC and VNIC in administration, as 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.

4.4.2 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.

4.4.3 Flows

Flows are additional instruments created to allow easier network traffic administration. They might be used in order to provide administer bandwidth resource control and priority for protocols, services, containers. 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 before adding new one its correctness is checked. Input and output packets are matched to flows in very efficient manner with minimal performance impact.

flowadm is the console command used to create, modify, remove or display network bandwidth and priority limits assigned to a particular link.

4.4.4 Running examples

dladm and **flowadm** are two basic administrative commands for dealing with the Crossbow's components. Below a few general examples of their usage are presented.

dladm is the admin command for crossbow datalinks elements management. Below a few examples of VNICs, Etherstubs management commands are presented and how bandwidth and priority values might be assigned to these 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

These were just basics, for more examples see **man dladm**

flowadm is the admin command for flow management. It migth be used as follow:

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

To see more see **man flowadm**

4.5 Resource access control

Nowadays existing operating systems must provide mechanisms for response to the varying resource demands per workload which is an aggregation of processes of an application. By default resource management features are not used and system gives equal access to resources. When it is necassary it is possible to modify these default behaviour with respect to different workloads. These management allows you to:

1. Restrict access to specific resource
2. Offer resources to workloads on a preferential basis
3. Isolate workloads from each another

Resource is any part of computing system that may be modified in order to change application behaviour. Resource management enables more effective resource utilization and avoid wasting available ones due to load variability. Reserving additional capability during off-peak periods and effective sharing resources definately inceases application performance.

Solaris Operating System introduced three types of resource management control mechanisms:

1. constraints - allows defining set of bounds on used resources for a workload
2. partitioning - enables binding subset of system's available resources to specific workload
3. scheduling - involves predictable algorithm making sequence of allocation decisions at specific intervals

Summary

Solaris 10 OS seems to be ideal cross-platform choice for customers dealing with managment of high level services, complex system administration and high costs. It is the only open operating system which has proven results running from every critical enterprise databases to high performance Web farms that is why Solaris OS is becoming strategic platform for today's constantly growing demands towards operating systems.

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

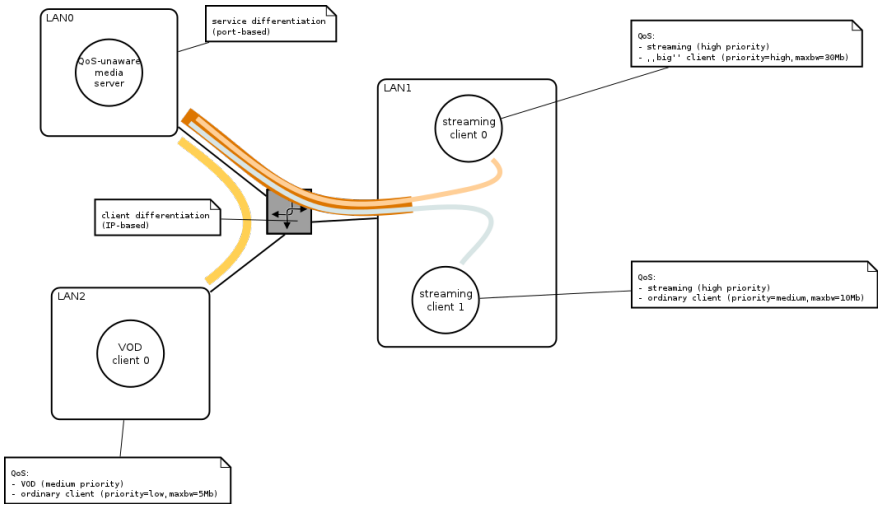

Figure 7.1: VOD + streaming clients test case example

7.1.2 Resource access requirements

7.1.3 Providing tunable and scalable virtual infrastructure

Summary

Chapter 8

Summary

Chapter overview

Bibliography [?] 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] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In *In Proc. 2nd Intl. SANE Conference*, 2000.

[2] Jose Renato Santos Yoshio Turner Jayaram Mudigonda. Taming heterogeneous nic capabilities for i/o virtualization. 2008.

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

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

[5] Inc. Sun Microsystems. *System Administration Guide: Solaris Containers—Resource Management and Solaris Zones*. 2005.

[6] Inc. Sun Microsystems. *Logical Domains 1.3 Administration Guide*. 2010.

[7] Paul A. Watters. *Solaris 10: The Complete Reference*. 1st edition, 2005.