

# **Establishment of QoS enabled multimedia collaboration Grid over native IPv6 fabric**

By

Vikas Agrawal

ID No. 2002B2TS894



Birla Institute of Technology and Science

Pilani, Rajasthan, India

(December 2005)

# **Establishment of QoS enabled multimedia collaboration Grid over native IPv6 fabric**

**THESIS**

Submitted in partial fulfilment of the requirements of  
BITS C421T

by

**Vikas Agrawal**

**2002B2TS894**

Under the supervision of  
**Dr. Rahul Banerjee**  
Associate Professor,  
Computer Science Group



**Birla Institute of Technology and Science**

Pilani, Rajasthan, India

(December 2005)

## ACKNOWLEDGMENTS

I am greatly indebted to Dr. Rahul Banerjee, my thesis supervisor. His keen discernment of what is important and what is not has often prevented me from straying in my research. His timely assistance and guidance has been a great source of help. He has been patient with me beyond the call of a supervisor. If I have gained any sense of what research is all about, it is due to him. I have learnt a lot under his supervision and I would like to thank him for the inspirational and resourceful advice and criticism.

I would also thank Dr. R. K. Mittal, Dean ARC division and Dr. Ravi Prakash, Dean Research and Consultancy Division for providing this course.

I also thank my family and friends for their constant and gentle encouragement throughout the course of my Thesis.



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE  
PILANI, RAJASTHAN 333031, INDIA.**

---

**CERTIFICATE**

This is to certify that the thesis entitled “Establishment of QoS enabled multimedia collaboration Grid over native IPv6 fabric” is the bona fide work by Vikas Agrawal (2002B2TS894) done in the first semester of the academic year 2005-2006. What has been actually achieved is a miniature test-bed using which some preliminary research experiments were performed. He has duly completed his thesis and has fulfilled all the requirements of the courses BITS C422T: Thesis and BITS C442T: Seminar, to my satisfaction.

Date:

**Dr. Rahul Banerjee  
(Supervisor)**

## ABSTRACT

This report describes a technique to deliver nearly constant perceptual quality of service when transmitting multimedia sequences over Differentiated Service IP networks. We propose a new approach to managing a rapidly evolving grid fabric and examine the aspect of Quality of Service in the Fabric layer and Application layer. Since Multimedia Applications extensively demand QoS, and this is related to a critical Health Services under “i-Charak”, a project that seeks to install a Grid in rural areas became the motivating factor in seeking out methods of ensuring QoS. The metrics of jitter, packet loss, delay and throughput are the relevant factors for this domain. SNMP may be used for monitoring the network interfaces and protocols for these metrics by the Grid manager. The dynamic nature of Grids resources necessitates that the monitored data be used to provide feedback for dynamic QoS provisioning. Thus the Grid network needs to adapt to the random variations in traffic and their priorities. Traffic control is used to apply the modified traffic policy at intermediate routing devices. Internet was partially simulated in our laboratory setup of Grid by assuming intermediate routers to be DiffServ enabled. Under such a scenario strict reservation of data paths for flows is not possible and without any dynamic traffic control services reduce to Best Effort. Experimental results included in this work demonstrate the impact of our dynamic reconfiguration of traffic control script in the improvement in reliability and throughput of multimedia collaboration between participating nodes in the grid environment.

## ACRONYMS AND ABBREVIATIONS

Diffserv	Differentiated Services Architecture
DSCP	Differentiated Services Code Point
Intserv	Integrated Services Architecture
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
QoS	Quality of Service
CSD	Centre for Software Development
API	Application Program Interface
ASP	Application Service Provider
PDA	Personal Digital Assistant
ICMPv4	Internet Control Message Protocol Version 4
ICMPv6	Internet Control Message Protocol Version 6
PHB	Per Hop Behaviour
IETF	Internet Engineering Task Force
TOS	Type of Service

## List of Figures

Fig. No.	Figure	Page
Figure 2.a	A simple Grid environment showing Grid Middleware	4
Figure 2.b	Grid Architecture and Internet Architecture	5
Figure 2.c	The Globus architecture	7
Figure 2.d	Architecture of Gridbus	9
Figure 4.a	Traffic Control	16
Figure 5.a	Network Characteristics for Grid	20
Figure 5.b	Grid Architecture	22
Figure 6.a	Test bed Setup of our Network in Laboratory	23
Figure 6.b	Network Diagram of Setup showing details of Links	24
Figure 7.a	One Way Delay Measurement of a band of traffic in 7 min. Interval	28
Figure 7.b	Delay Variation (Jitter) of traffic in an interval of 1 min	28
Figure 7.c	One-Way Delay over a period of 3 minutes for multimedia and a normal traffic	29
Figure 7.d	One-Way Delay Distribution for multimedia and a normal traffic	30
Figure 7.e	One Way Delay for two different traffic streams after being shaped	31
Figure 7.f	Traffic of Experiment 3 after being dynamically shaped	33

## List of Tables

Table No.	Table	Page
Table 7.a	Profile of Traffic for Experiment 1	27
Table 7.b	Profile of Traffic for Experiment 2	29
Table 7.c	Profile of Traffic for Experiment 3	31

# TABLE OF CONTENTS

<i>Acknowledgement</i>	<i>iii</i>
<i>Abstract</i>	<i>v</i>
<i>Acronyms and Abbreviations</i>	<i>vi</i>
<i>List of Figures</i>	<i>vii</i>
<i>List of Tables</i>	<i>viii</i>
1. Introduction.....	1
1.1 What is a Grid.....	1
1.2 Quality of Service.....	3
1.3 Motivation.....	3
2. Grid Middleware.....	4
2.1 Grid Architecture.....	4
2.2 Necessity of Grid Middleware.....	6
2.3 Approach towards Resource management.....	7
2.4 Few existing Grid Middleware's.....	7
2.4.1 Globus.....	7
2.4.2 Legion.....	8
2.4.3 Condor.....	8
2.4.4 Gridbus.....	8
2.4.5 Alchemi.....	10
3. Multimedia Application & Protocols.....	11
3.1 Multimedia Collaboration.....	11
3.1.1 Taxonomy of Multimedia collaboration.....	11
3.2 Real Time Streaming Protocol.....	12
3.3 H.323.....	12
4. Quality of Service.....	13
4.1 QoS in Packet Switched Network.....	13
4.1.1 Traffic Parameters.....	14
4.1.2 Resource Reservation.....	14
4.1.3 Admission Control.....	15
4.1.4 Traffic Policing.....	15
4.1.5 Traffic Shaping.....	15
4.1.6 Queuing and Scheduling.....	15
4.1.7 Congestion Control and Buffer Management.....	16
4.2 QoS Problems in IPv4.....	16
4.3 QoS in IPv6.....	16
4.4 Major QoS Framework.....	17
4.4.1 DiffServ .....	17
4.4.2 IntServ.....	17
4.5 Multimedia Application QoS.....	18
4.5.1 Video traffic.....	18
4.5.2 Audio traffic.....	18

4.5.3 Interactive traffic.....	18
5. Design to provide Grid QoS.....	19
5.1 Goals.....	19
5.2 Constraints.....	20
5.3 Various Issues concerning Grid QoS.....	21
5.4 Architecture.....	22
6. Implementation, Measurement and Analysis.....	23
6.1 Network Setup.....	23
6.2 Measurement strategies.....	24
6.2.1 Time Synchronization.....	25
6.2.2 Traffic generation.....	25
7. Results and Analysis.....	27
7.1 Experiment 1 (Set of 5 traffic).....	27
7.2 Experiment 2 (Multimedia and junk traffic).....	29
7.3 Experiment 3 (Shaped Multimedia and Junk traffic).....	30
7.4 Experiment 4 (Dynamically shaped traffic).....	32
8. Conclusions.....	34
<i>Appendix A: Traffic Control scripts</i>	35
<i>Appendix B: Shell script for dynamically generating tc scripts</i>	36
<i>Appendix C: Traffic generation scripts</i>	49
<i>Appendix D: Route Configuration</i>	50
<i>References</i>	53

## Chapter 1

### *Introduction*

Titled as “i-Charak”, this project is an IPv6 QoS aware Grid computing project initiated at CSD, BITS Pilani, India. To provide an initial breakthrough to medical facilities in rural area we are focusing on utilization of urban resources by collecting data from rural areas and after getting it processed by grid these data are passed to specialists in the required area. Grid computing originated from a new computing infrastructure for scientific research and cooperation and is becoming a mainstream technology for large-scale sharing and distributed system integration. This character of the Grid makes it a sort of medium to port with various types of devices and/or sub-grids. In nutshell major concern of this project is to “Map the Grid approach to the specific domain of Health Care Support System” which is the theme specific to our “i-Charak”. In the last couple of years, several projects concerned with health-grid have been funded at national and international level. The crucial part is to integrate or synchronize these technical breakthroughs in the interest of participating players in healthcare, physicians, healthcare centers, and administrations, and of course common mass. “i-Charak” is the result of our sincere effort towards this. First section describes Grid, beginning with the basic definition,

#### *1.1 What is Grid?*

Grid refers to an aggregated system comprising of geographically sparsely distributed “autonomous” resources depending on their availability, capability, cost, and user QoS requirements [1]. Here resources can be

- Computers - Personal computers, workstations, clusters, supercomputers, laptops/notebooks, mobile devices, Personal Digital Assistants etc.
- Software – e.g., ASPs renting expensive special purpose applications on demand;
- Catalogued data and databases – e.g. transparent access to elements and alloys chemical properties;
- Special devices/instruments – e.g., radio telescope – SETI@Home
- People/collaborators.

Distinguished from conventional distributed computational technologies, Grid computing originated from a new computing infrastructure for scientific research and cooperation and is

becoming a mainstream technology for large-scale resource sharing and distributed system integration. It provides integrated approach to the coordinated resources at multiple sites for computation. Normally, we hope that the Grid can gather computing power of many supercomputing facilities together [2]. But because of the limitation of network bandwidth and latency over the Internet, the true computation over the Grid is still a task to be accomplished. To make the Grid more accurate, and widely useful, one must be able to design applications that are flexible enough to exploit various ensembles of true supercomputers, clusters, workstations, servers etc. along with matching application requirements and characteristics with Grid resources, in order that there is very little of these kinds of control messages.

1.1.1 Grids can be further classified as :

- Scavenging Grid: These grids utilize idle cycles of participating nodes to support applications that can be broken into independent tasks. Examples being SETI@home and World Community Grid
- Collaboration Grid: Grids that support the applications that enhance human – to – human interactions. Example being Access Grid
- Data Grid: Grids that synthesize new information from data that is maintained in geographically distributed repositories, digital libraries, and databases. Examples being TerraServer and Sloan Digital Sky Server
- Enterprise / Campus / Intra-Grid: Grid that brings together resources within the same organization.

1.1.2 Classes of Services that can be powered by Grids are:

- Computational Services – CPU cycles  
e.g. NASA IPG, WWG, TeraGrid, SETI@Home
- Data Services
  - Data replication, management, secure Application Services
  - Access to remote software/libraries and license management—NetSolve
- Information Services
  - Extraction and presentation of data with meaning
- Knowledge Services
  - The way knowledge is acquired and managed—data mining.

### 1.1.3 Characterization of Grid Entities / Issues

The basic characteristic of geographically distributedness forms the integral component of Grid, with Users, Resources, and Owners spread through a large area. Further Resources, Users, Applications, QoS requirements can be heterogeneous in nature. Since participating stations can be dedicated or otherwise, Resource availability / capability may vary from time to time. Policies and strategies are also heterogeneous and decentralized since Grid is composed of various domains under different owners. Depending on resources, availability and time cost varies in Grid.

## 1.2 Quality of Service

Our project focusing on such a sensitive issue of health issues that it demands Quality of Service (QoS). Quality of Service is the collective effect of service performance which determines the degree of satisfaction of a user of the service. Particular challenges in providing QoS include concurrent flows with different flow specifications, application level monitoring and control, and end-to-end QoS across networks and other devices [13]. The QoS term has been used primarily in the networking community to define a set of network performance characteristics such as delay, jitter, bit error rate, packet loss, and more. This is covered in detail in coming chapter number 4.

## 1.3 Motivation

Research and development while being the primary goals of the technical community and academia are not an end in themselves but are the means to address and provide solutions to real world concerns. Project “i-Charak”, under the umbrella project “Grid-One” purpose is to apply the generated knowledge and technology in setting up a grid enabled public health service system. The immense spread of a country like India makes it very difficult to provide public services that can claim to uniformity in their standards. The aim is to provide a ubiquitous, low cost solution keeping in mind the various aspects of the problem.

## Chapter 2

### *Grid Middleware*

Grid middleware provides users with seamless computing ability and uniform access to resources in the heterogeneous Grid environment. They are software stacks designed to present disparate compute and data resources in a uniform manner, such that these resources can be accessed remotely by client software without needing to know *a priori* the systems' configurations. In Fig 2.a the middleware module is shown along with the hardware resources. It can be conceptualized as a sort of layer between the Operating system and the applications. Several software toolkits and systems have been developed, most of which are results of academic research projects, all over the world like Globus and Gridbus.

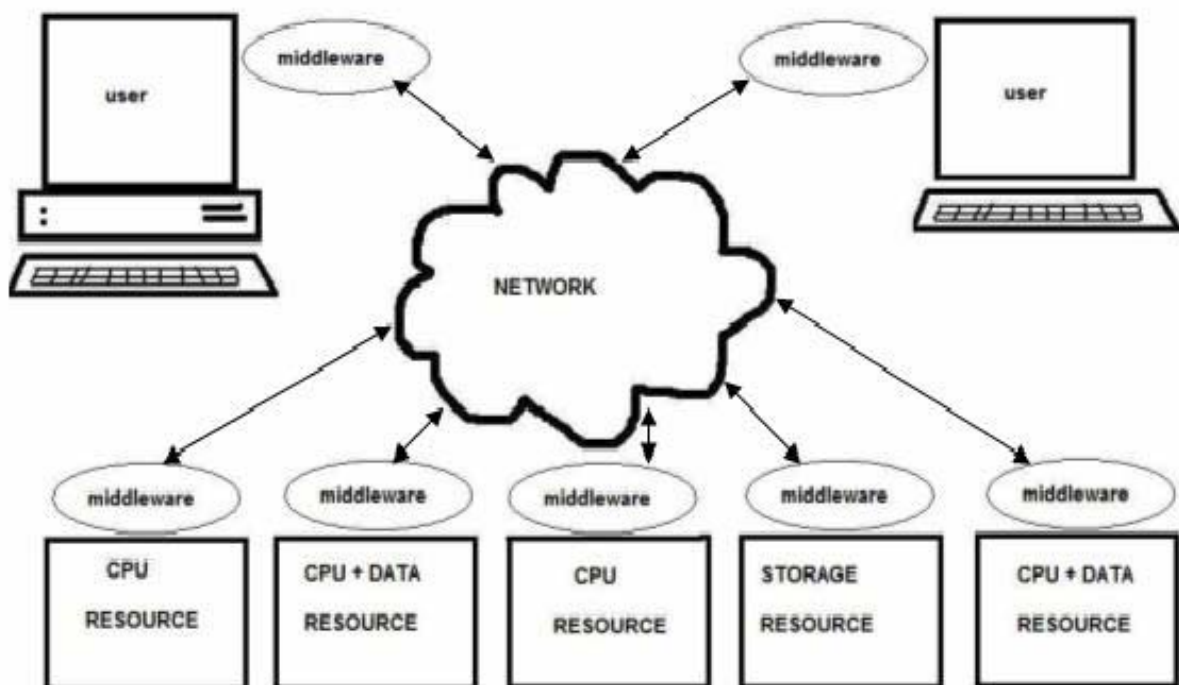


Fig 2.a A simple Grid environment showing Grid Middleware

### 2.1 *Grid Architecture*

Grid may be seen as made up of five layers:

- Application Layer
- Collective Layer
- Resource Layer
- Connectivity Layer
- Distributed computing / fabric Layer

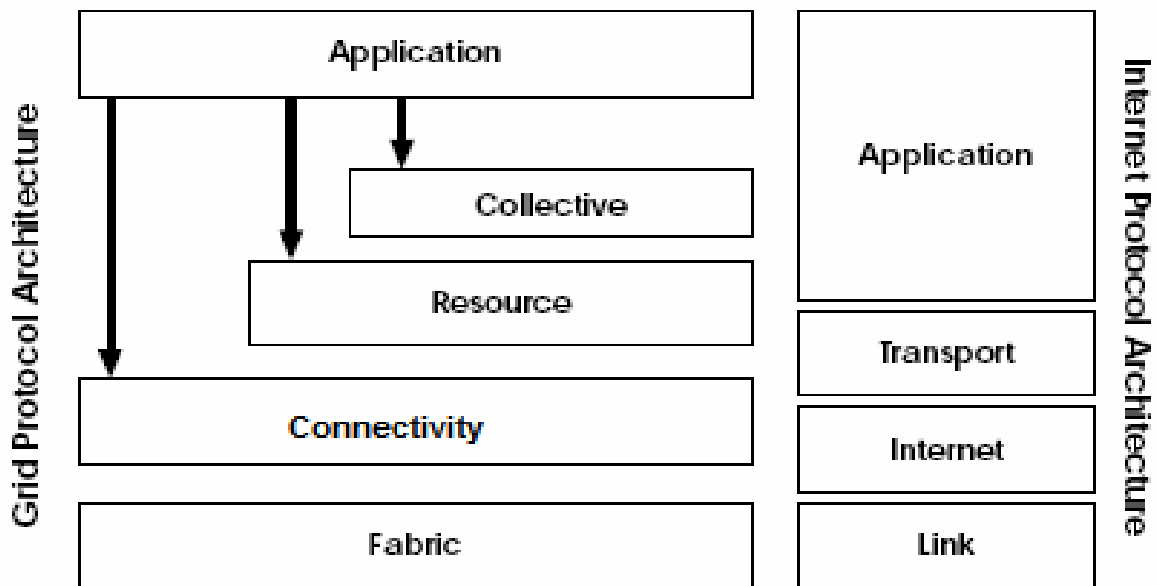


Fig 2.b Grid Architecture and Internet Architecture

This new architecture called grid architecture, identifies the basic components of a grid system, defines the purpose and functions of such components and indicates how each of these components interact with one another (Foster, Kesselmann & Tuecke). The main attention of the architecture is on the interoperability among resource providers and users to establish the sharing relationship. This interoperability means common protocols at each layer of the architecture model. Fig.2.b displays this protocol architecture which defines common mechanisms, interfaces, schema, and protocols at each layer, by which users and resources can negotiate, establish, manage and share resources. It also shows the corresponding Internet protocol layers.

Each layer shares the behavior of the underlying component layers. The following describes the core features of each of these component layers, starting from the bottom of the stack and moving upward:

- Fabric layer: The fabric layer defines the interface to local resources, which may be shared. This includes computational resources, data storage, networks, catalogs, software modules, and other system resources.
- Connectivity layer: The connectivity layer defines the basic communication and authentication protocols required for grid-specific networking-service transactions.
- Resource layer: This layer uses the communication and security protocols (defined by the connectivity layer) to control secure negotiation, initiation, monitoring, accounting, and payment for the sharing of functions of individual resources. The resource layer calls the fabric

layer functions to access and control local resources. This layer only handles individual resources, ignoring global states and atomic actions across the resource collection pool, which are the responsibility of the collective layer.

- **Collective layer:** While the resource layer manages an individual resource, the collective layer is responsible for all global resource management and interaction with collections of resources. This protocol layer implements a wide variety of sharing behaviors using a small number of resource-layer and connectivity-layer protocols.
- **Application layer:** The application layer enables the use of resources in a grid environment through various collaboration and resource access protocols.

## 2.2 Necessity of Grid Middleware

Grid middleware is an integral component of grid architecture, since this layer hides underlying complexity for the users. All challenges including Security, Resource Discovery, Resource Allocation and Scheduling, Uniform Access, System Management, Computational Economy, Data Locality, Network Management is managed by Grid middleware [14]. Hence, user need not worry about issues like “Node of executing my job”, “How many nodes should be employed for the job”. But yes, as point of interaction (Service access points), user can specify various parameters to middleware which takes care of those issues later. There are several sources of complexities that Grid middleware handles for us like

- Size (large number of nodes, providers, consumers)
- Heterogeneity of resources (PCs, Workstations, clusters, and supercomputers)
- Heterogeneity of fabric management systems (single system image OS, queuing systems, etc.)
- Heterogeneity of fabric management policies
- Heterogeneity of applications (scientific, engineering, and commerce)
- Heterogeneity of application requirements (CPU, I/O, memory, and/or network intensive)
- Heterogeneity in demand patterns
- Geographic distribution and different time zones
- Differing goals (producers and consumers have different objectives and strategies)
- Insecure and Unreliable environment

### 2.3 Approach towards Resource Management

Traditional Approach towards Resource Management uses centralized policy, and hence need complete state information, common fabric management policy or de-centralized consensus based policy, which will not be applicable in the scenario of Grid environment. Due to too many heterogenous parameters in the Grid it is impossible to define system-wide performance matrix and common fabric management policy that is acceptable to all. Therefore, a decentralised management technique such as “distributed computational economy-” seems feasible. It proved successful in managing decentralization and heterogeneity that is present in human economies! We can easy leverage proven Economic principles and techniques, easy to regulate demand and supply, user-centric, scalable, adaptable, value-driven costing, etc. by offering incentive for being part of the grid.

### 2.4 Few existing Grid Middleware:

#### 2.4.1 Globus

The Globus project is a multi-institutional research to create a basic infrastructure and high-level services for a computational grid [8]. They have now evolved into an infrastructure for resource sharing among heterogeneous virtual organizations.

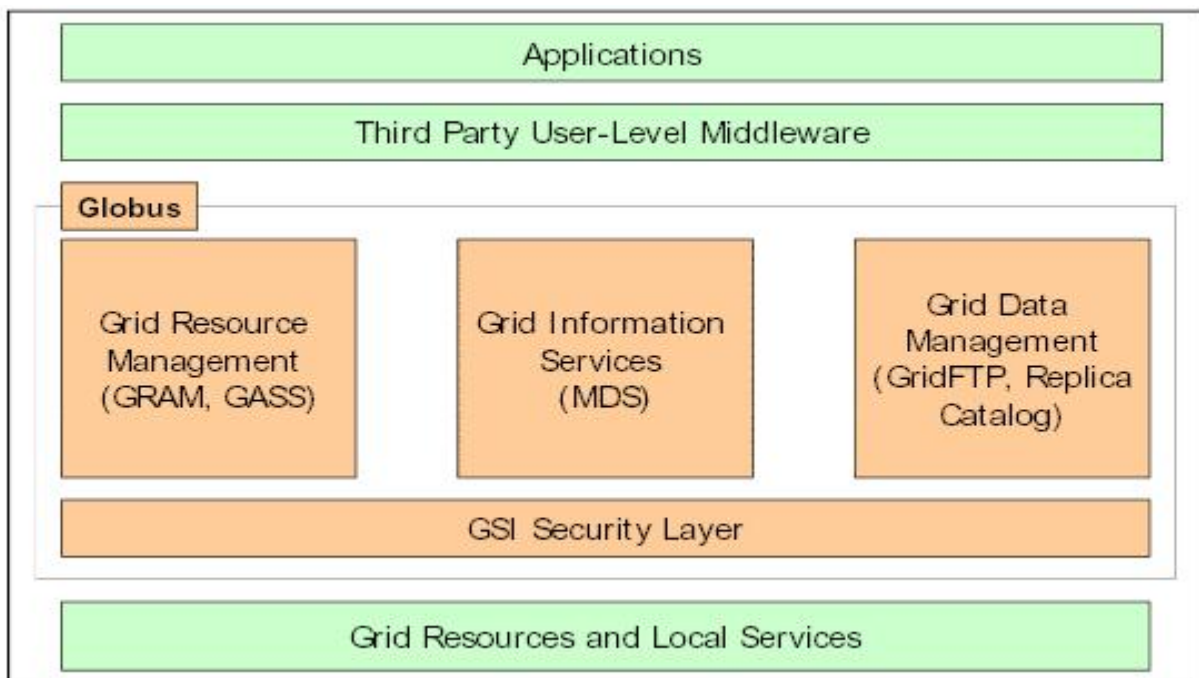


Fig 2.c The Globus architecture

Globus Toolkit also provides a set of tools for application programming (APIs) and system development kits (SDKs) which are basically implemented in the C language. Globus provides a rapid development kit known as CoG (Commodity Grid), which supports the following technologies: Java, Python, Web services, CORBA, Java Server Pages, Perl, Matlab.

#### 2.4.2 *Legion*

It is a middleware project initiated by the University of Virginia, is object based metasystems software for grid applications. Goal being to promote the principle design of distributed system software by providing standard object representations for processors, data systems, file systems, and so on.

#### 2.4.3 *Condor*

This focuses on utilization of capacity of idle workstations for computational tasks. It is well suited for jobs which generally do not need to communicate with each other. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management

#### 2.4.4 *Gridbus*

The Gridbus Project is an open-source, multi-institutional project led by the GRIDS Lab at the University of Melbourne [8]. It is engaged in the design and development of service-oriented cluster and grid middleware technologies to support e-Science and e-Business applications. It extensively leverages related software technologies and provides an abstraction layer to hide idiosyncrasies of heterogeneous resources and low-level middleware technologies from application developers. In addition, it extensively focuses on realization of utility computing model scaling from clusters to grids and to peer-to-peer computing systems. It uses economic models in efficient management of shared resources and promotes commoditization of their services. Thus, it enhances the tradability of grid services and manages efficiently the supply and demand for resources. Fig. 2.d shows architecture of Gridbus.

Gridbus supports commoditization of grid services at various levels:

- Raw resource level (e.g., selling CPU cycles and storage resources)
- Application level (e.g., molecular docking operations for drug design application)
- Aggregated services (e.g., brokering and reselling of services across multiple domains)

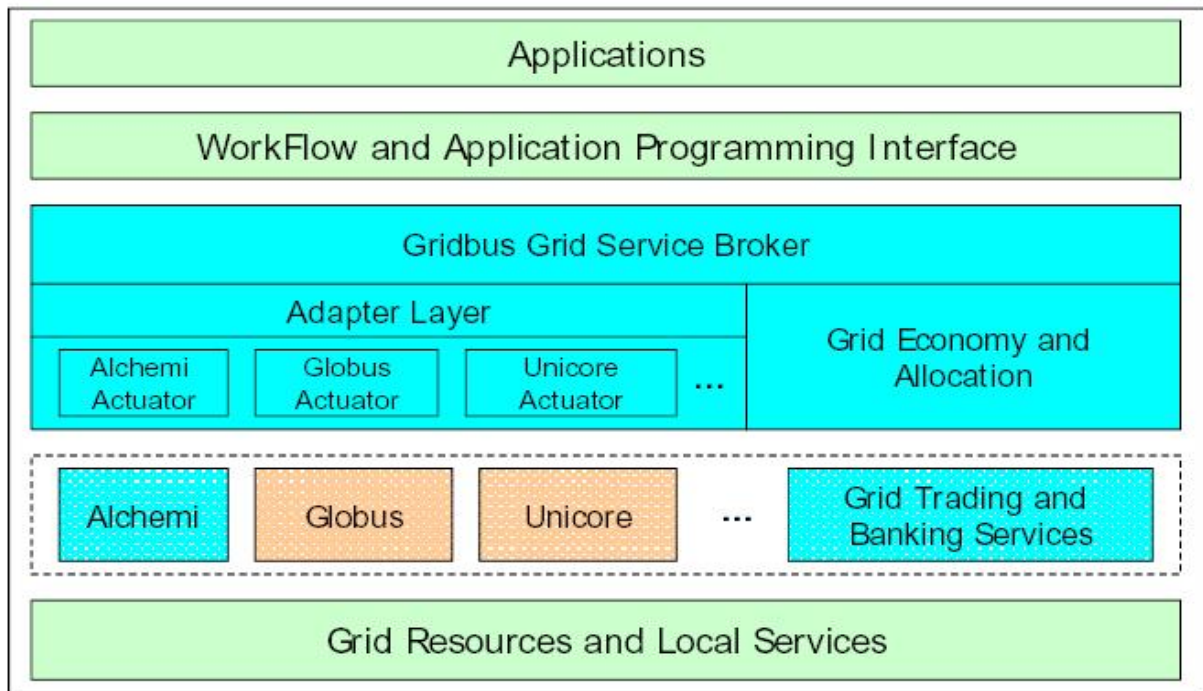


Fig 2.d Architecture of Gridbus

Gridbus emphasizes the end-to-end quality of services driven by computational economy at various levels - clusters, peer-to-peer (P2P) networks, and the Grid - for the management of distributed computational, data, and application services.

Gridbus provides software technologies that spread across the following categories:

- Enterprise Grid Infrastructure (Alchemi)
- Cluster Economy and Resource Allocation (Libra)
- Grid Economy and Virtual Enterprise (Grid Market Directory, Compute Power Market)
- Grid Trading and Accounting Services (GridBank)
- Grid Resource Brokering and Scheduling (Gridbus Broker)
- Grid Workflow Management (Gridbus Workflow Engine)
- Grid Application Programming Interface (Visual Parametric Modeller)
- Grid Portals (GMonitor, Gridscape)
- Grid Simulation (GridSim)

#### 2.4.5 *Alchemi*

Though scientific computing facilities have been heavy users of Unix-class OSes, the vast majority of computing infrastructure within enterprises is still based on Microsoft Windows. Alchemi was developed to address the need within enterprises for a desktop grid solution that utilizes the unused computational capacity represented by the vast number of PCs and workstation running Windows within an organization. Alchemi is implemented on top of the Microsoft .NET Framework and provides the runtime machinery for constructing and managing desktop grids. It also provides an object-oriented programming model along with web service interfaces that enable its services to be accessed from any programming environment that supports SOAP-XML abstraction.

#### *Reasons for Selecting Alchemi/Gridbus as an initial base for our middleware*

Alchemi which is an integral component of Gridbus (as described in the architecture) has following which makes it suitable for our use:

- objects are much easier to deal with than processes combined with
- a programming model that is familiar to developers and
- the use of language constructs across local and remote code makes this a superior approach

## Chapter 3

### *Multimedia Applications and Protocols*

Transmitting real-time multimedia streams over heterogeneous networks is a challenging task. Variation in network and system conditions can dramatically affect application performance. Therefore Applications and Protocols in use should have sufficient provisions like buffering, reservation etc. First section deals with various kinds of multimedia collaborations. Later two important protocols: RTSP and H.323 are described.

#### 3.1 Multimedia Collaborations

Media collaborations among the participants of a structured collaboration can be either interactive (as in a telephone conversation), or non-interactive (as in electronic mail) [10]. Whereas interactive collaboration involves contemporaneous exchange of media information, non-interactive collaborations are characterized by recipient's discretion delays after transmission by sender. During such intervening delays, the media information in transit may have to be managed by servers.

##### 3.1.1 Taxonomy of Multimedia Collaborations

Interactions involving communication among multiple participants are referred to as collaborations. These can be classified as follows:

- Simple Collaborations: It contains only simple participants, i.e. individual users
- Nested Collaborations: It contains at least one super-participant, i.e. another collaboration
- Interactive Collaborations: It involves contemporaneous exchange of media information among their participants.
- Static Collaborations: It keeps access rights of participants constant once initiated.
- Dynamic Collaborations: permits access rights of their participants to be changed at any time.
- Symmetric Collaborations: require all their participants to possess the same set of access rights.
- Homogeneous Collaborations: doesn't allow conversion between different media during communication between participants.
- Transient Collaborations: Short-lived, and their existence depends on the participation of each of their participants.

### 3.2 *Real Time Streaming Protocol* (RFC 2326)

The Real-Time Streaming Protocol establishes and controls either a single or several time-synchronized streams of continuous media such as audio or video [10].

RTSP\_URL = ( "rtsp:" | "rtspu:" ) "://" host [ ":" port ] [ absolute\_path ]

### 3.3 *H.323*

It is more of an architectural overview of Internet telephony than a specific protocol. It references a large number of specific protocols for speech coding, call setup, signaling, data transport, and other areas rather than specifying these things itself [10]. H.245 protocol is now used to negotiate the parameters of the call. Each side starts out by announcing its capabilities, for example, whether it can handle video (H.323 can handle video) or conference calls, which codec it supports, etc. Once each side knows what the other one can handle, two unidirectional data channels are set up and a codec and other parameters assigned to each one. After all negotiations are complete, data flow can begin using RTP. Q.931 call signaling channel is used to tear down connection when either party hangs up.

There are many tools used to implement QoS in H.323. Some of these are: Differential Services Code Point (DSCP), Resource Reservation Protocol (RSVP), Priority Queuing (PQ), Committed Access Rates (CAR), and Weighted Random Early Detection (WRED), among others. Using many or all of these tools, in conjunction with other technologies, creates a working QoS.

## Chapter 4

### *Quality of Service*

QoS is the set of techniques to manage network resources in a manner that enables the network to differentiate and handle traffic based on policy. This means providing consistent, predictable data delivery to users or applications that are supported within the network [3]. Achieving the required Quality of Service (QoS) by managing delay, delay variation (jitter), bandwidth, and packet loss parameters on a network becomes the secret to a successful end-to-end business solution. Implementing a successful QoS policy will ensure a predictable, measurable, and guaranteed treatment of the communications on the backbone [6]. This would provide a better means of handling specialized traffic on the limited and costly backbone resources. The demand for the use of bandwidth-intensive and delay/jitter-sensitive applications coupled with the ever-increasing use of backbone for “unsupported” traffic provides the need and reasoning for implementing QoS. The QoS term has been used primarily in the networking community to define a set of network performance characteristics such as delay, jitter, bit error rate, packet loss, and more [11]. A brief description of these parameters is provided below:

- Delay: Time it takes for a message to be transmitted
- Response Time: Round-trip time from request transmission to reply receipt
- Jitter: Variation in delay or response time
- Bit error rate: Proportion of total data that does not arrive as sent because of errors in the network transmission system.
- Packet loss rate: Proportion of total packets that do not arrive as sent

#### 4.1 QoS in Packet switched network

One of the basic requirements of QoS is to be able to qualitatively describe the pattern of traffic generated from a given source, so that network can appropriately allocate its resources to support the required QoS. The dynamics of bit rate over time describes the behaviors of a traffic source. Based on dynamics of bit rate all applications can be categorized into two main categories:

- Constant Bit Rate (CBR): These applications send traffic at a constant rate. Many multimedia applications fall under this category.

- Variable Bit Rate (VBR): Traffic rates of these applications are not constant. A good example is MPEG coded video. When there is lot of scene change, it generates many bits per second.

#### 4.1.1 *Traffic Parameters*

The following three parameters are commonly used to bound source traffic:

- Peak Rate: is the maximum data rate in any time interval. CBR can be completely described using the peak rate of the traffic.
- Average Rate: is the “long term” mean of the traffic rate for a VBR source.
- Burst Size: refers to the number of packets that can be delivered in the peak rate. A limit on this parameter is a useful parameter for a QoS contract.

#### 4.1.2 *Resource Reservation*

For absolute QoS guarantee, it may be required to reserve some resources in advance of the call and release them after the call ends. Generally desired resources are link bandwidth and buffer space. Adequate bandwidth reservation helps in maintaining delay and jitter requirements of critical calls. Reserving buffer space helps in maintaining any negotiated limit on the packet loss rate in the network [13]. RSVP is a network control protocol that enables Internet applications to obtain guaranteed quality of service (QoS) for their data flows. RSVP causes routers to reserve resources, if available, when they receive requests from hosts. These requests have to be checked by each router management system, RSVP process, policy control and admission control, to check resources availability. Routers that accept these requests are called soft states and are then kept alive by sending refreshing messages. RSVP can dynamically change the QoS parameters at any time without tearing down the connection. The two fundamental message types in RSVP are the Resv (reservation) message and the *Path* message. The Resvs are generated by receivers to create and maintain the reservation state in each node along the path and they contain information about the reservation style, Flow Spec used to set a node's packet scheduling process parameters, and Filter Spec which is used to set node's packet classifier process. The Path messages store path information, previous Hop IP address, and parameters that describe the sender's traffic, such as the sender traffic flow, that help identify the sender's flow from others.

#### 4.1.3 Admission Control

Since Best-effort networks do not guarantee QoS accept traffic from any source without any question, and a newly admitted source may jeopardize the QoS of an existing connection, hence admission of new connections must be carefully controlled in a QoS network to protect the QoS of the call currently in progress [6].

Hence as a result of admission control either a new call will be rejected or accepted.

#### 4.1.4 Traffic Policing

To be implemented at the edge / entrance of the network, this should detect all entering traffic to detect any violation of the negotiated contract [11].

- It must not discard or decrease the priority of packets that do not violate the negotiated contract
- It must detect every packet that violates the contract and take appropriate actions
- Real time operation should be there

#### 4.1.5 Traffic Shaping

To avoid any violation of negotiated traffic parameters, shaping is an essential during call set-up. Shaping of traffic means smoothing out any traffic bursts. Traffic shapers do not want to discard violating traffic, but to store them in actual buffers and smooth it out. There is no actual buffering in traffic policing. Fig 4.a shows traffic control points of packets.

#### 4.1.6 Queuing and Scheduling

Queuing refers to the process of buffering incoming packets at the entrance of a communication link. Link transmits one packet at a time from the buffer. When multiple packets wait in the buffer then scheduling algorithm comes into the picture. Packet loss rate, packet delay, and other QoS parameters of a given traffic may be significantly affected by the choice of queuing and scheduling techniques. Scheduling Goals are:

- Sharing bandwidth
- Providing fairness to competing flows
- Meeting bandwidth guarantees (minimum and maximum)
- Meeting loss guarantees
- Meeting delay guarantees
- Reducing delay variations

#### 4.1.7 Congestion Control and Buffer Management

Congestion in network devices such as routers and the switches is a major cause of packet loss in wired networks. A network can take either proactive or reactive measures to control congestion. BE model relies on reactive mode. Buffer management is a proactive technique whereby the networking devices monitor their queue length and once it exceeds a certain threshold, they start dropping packets [5].

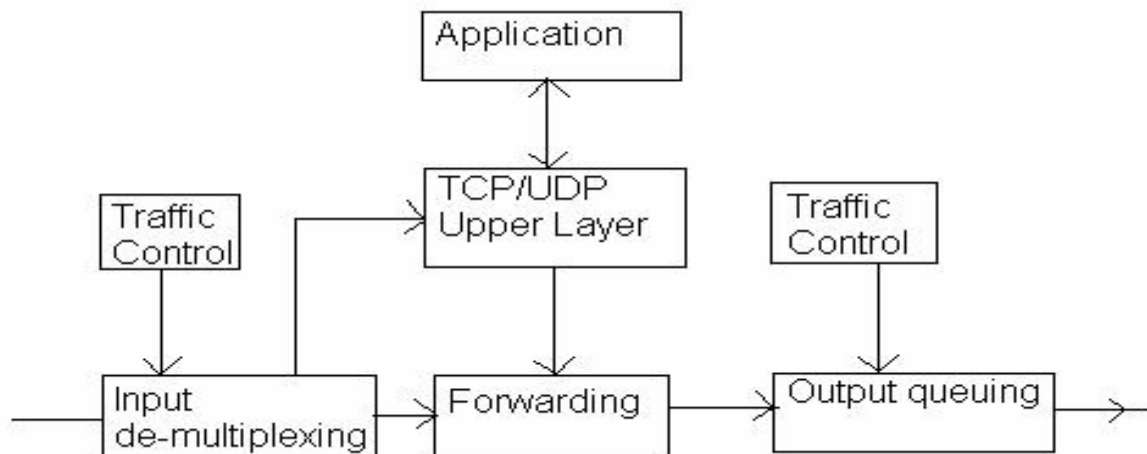


Fig 4.a Traffic Control

#### 4.2 QoS Problems in IPv4

Fragmentation poses a great deal of problem since it produces congestion, consumes bandwidth, and CPU resources. Inefficient routing is a direct consequence of fragmentation. Further, since ICMPv4 has too many options, it results in control overhead.

#### 4.3 QoS in IPv6

An efficient packet processing permits fast forwarding, reducing queuing delays, hence keeping these in mind, IPv6 packet was specially designed to be efficiently managed by routers [12]. It has less number of fields; flow label is placed just before address, which eases the case of flow routing. As compared to ICMPv4, ICMPv6 is lighter and more concrete. IPv6 provides auto configuration.

Contribution of 20 bit flow label in IPv6 is quite huge. A flow is traffic with common semantics; therefore packets with same flow are treated equally. Characteristics of Flow label field are:

- Identifies IPv6 packets with the same origin and destination so they can be treated equally
- Packets may not be inspected and classified every time because of encryption

Other IPv6 advantages include mobility support, and Security.

#### 4.4 Major QoS Framework

There exist two major QoS framework defined in the internet world: one in Integrated Services (IntServ) and the other one is Differentiated Services (DiffServ) [11].

##### 4.4.1 DiffServ

DiffServ goes for aggregated flow, since it recognizes the cost associated with maintenance of per flow state. Hence some accuracy is lost but significantly reduces overhead associated with IntServ concepts [4].

In an IPv6 packet header, there is a field called Traffic Class (8 bits). In an IPv4 packet header, this field resides in the TOS (Type of Service) area. Traffic Class and TOS are renamed as the DS-Field in the context of DiffServ, the coded value is referred to as the DS Code Points (DSCP). The DS Field has a six-bit DiffServ Code Point and two “currently unused” bits. In DiffServ, the edge nodes of a network classify traffic and set the DSCP (DiffServ value). Edge nodes are also responsible for any Policing and/or shaping of traffic. In the core of the network, traffic is forwarded based on a PHB (per hop behavior) associated with a particular DSCP. This means that DiffServ core networks should be sized to deal with the peak sum of priority traffic from all entry points.

##### 4.4.2 IntServ

Introduced in 1994 by IETF, IntServ goes for accurate traffic flow management, uses signaling mechanism to reserve resources along the path. IntServ can for example be used to allow video and sound to reach the receiver without interruption. It specifies a fine-grained QoS system, which is often contrasted with DiffServ's coarse-grained control system

#### 4.5 Multimedia Application QoS

Application QoS parameters are used to describe end-to-end application performance in accordance with software and hardware resources of end systems/services. High-demanding audio/video applications that operate in internet environment experience frequent quality-of-service (QoS) violations due to packet loss and delay variations [3]. The reasons are mainly network congestion, handovers between different wired or wireless network technologies, and jitter.

##### 4.5.1 Video Traffic

Traffic pattern for these traffics generally include a normal flow with burst at certain regular (in some cases random) intervals, during which there are high chances of packet loss [10]. So prescribed value of TOS is 0x8, which means minimum delay, and is used by DiffServ domains for prioritizing this traffic. In this case, more than the actual delay, jitter or delay variation is important since synchronization of audio and picture is essential. About 10ms delay in variation can cause problems during playback.

##### 4.5.2 Audio Traffic

Traffic pattern for this traffic does not follow similar pattern [10]. In case of high data rate, it becomes difficult to prioritize that traffic. Latency doesn't affect the quality of the call in the same way tat packet loss does, but it can make conversation difficult. TOS field for this kind of traffic is 0x40.

##### 4.5.3 Interactive Traffic

This kind of traffic prefers minimum delay, say for example telephone conversation. In case a party hangs up, call gets disconnected making it interactive in nature. A question is generally followed by answer to the same question and not for the previous one. So Delay should be well within a limit [10].

## Chapter 5

### *Design to provide Grid QoS*

Factors that affect Grid QoS are primarily focused in this chapter. Few of the hurdles in providing QoS to Grid, that we come across are:

- Performance information has a fixed, often short, lifetime of utility. Most monitoring data goes stale quickly, making rapid read access important but obviating the need for long-term storage. The notable exception to this is data that gets archived for accounting or postmortem analysis.
- Updates are frequent. Unlike the more static forms of “metadata” dynamic performance information is typically updated more frequently than it is read. Since most extant information-base technologies are optimized for query and not for update, they are potentially unsuitable for dynamic information storage.
- Performance information is often stochastic. It is frequently impossible to characterize the performance of a resource or an application component by using a single value. Therefore, dynamic performance information may carry quality-of-information metrics quantifying its accuracy, distribution, lifetime, and so forth, which may need to be calculated from the raw data.

#### 5.1 Goals

**Resource Discovery and Monitoring:** In order to support fabric layer QoS, tools used for resource discovery and monitoring is included in Application level, which can easily be integrated to Grid middleware. SNMP agents do the job of querying status of participating stations, intermediate routers.

**Resource Management:** To deliver what has been negotiated between Grid Manager and requesting node, there has to be a mechanism to control a resource for optimal manageability, such as start and stop activations, problem resolution, configuration management, load balancing, work-flow, complex event correlation, and scheduling. This translates to the ability to provide for traffic control configuration and implementation with respect to the network layer. The design that is proposed conforms to these pre-requisites.

The Global Grid Forum has defined the network performance characteristics for Grid Applications and Services, represented as the figure below:

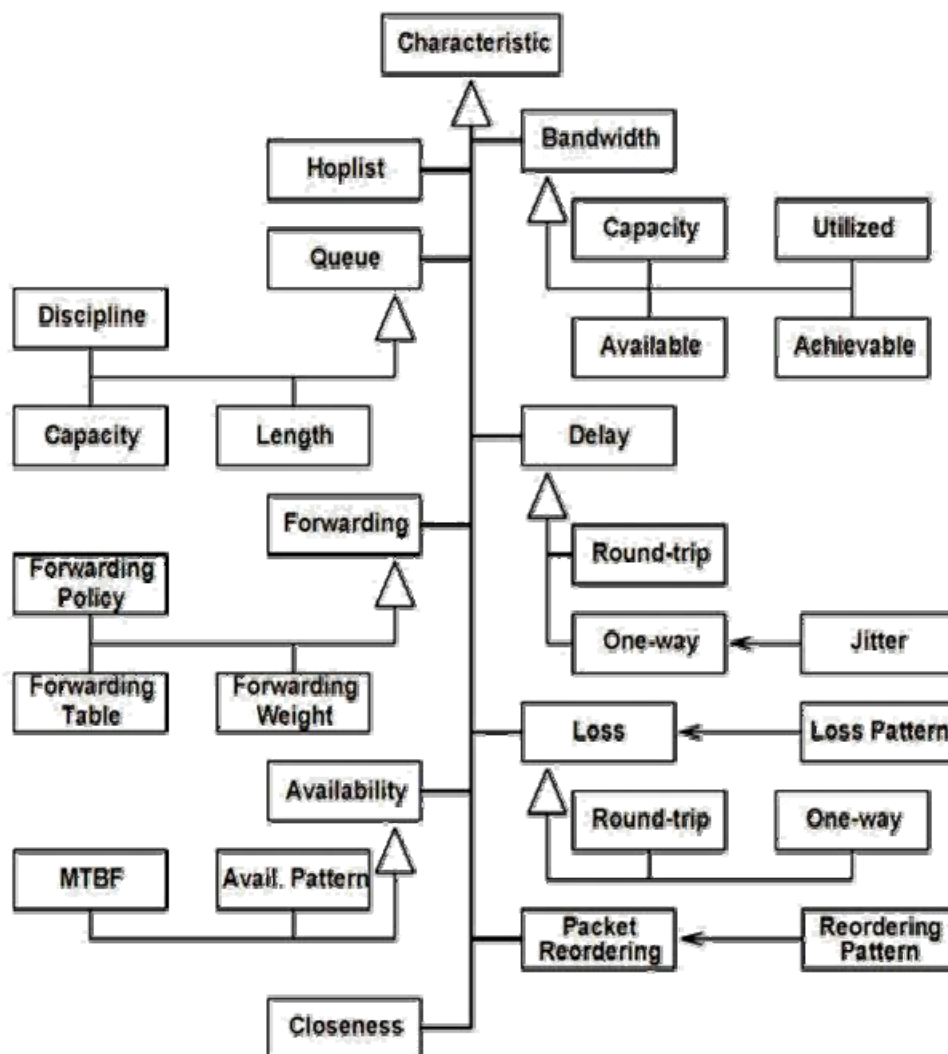


Fig 5.a Network Characteristics for Grid

## 5.2 Constraints

- We were not having our customized middleware due to which we could not integrate the Application and Fabric level QoS feature. Alchemi was taken as a reference and implementation of the application module was done on the .NET framework.
- Since laboratory test bed could not be geographically diverse, and with limitation in participating nodes, Results were obtained by treating intermediate routers DiffServ compliant. Our test bed is described in length in next chapter. In the real world the minimum QoS level will be decided by the intermediate router configurations which may or may not support DiffServ framework.
- An application running on the Grid can access information as extracted through the middleware's user level functions but the interpretation will vary as per the application's

specific needs. For example IP packet loss values may be critical for some application only after some threshold is crossed and it is the application's responsibility to inform the middleware about this. A better approach would be assigning the middleware component on the user's node this responsibility to takes decisions based on the QoS levels the application communicates at the start when it needs some service from the Grid. This also implies storing and processing this information and more computation for the middleware component which may not scale well.

### 5.3 Various Issues concerning Grid QoS

- Low latency: As previously stated, performance data is typically relevant for only a short period of time. Therefore, it must be transmitted from where it is measured to where it is needed with low latency.
- High data rate: Performance data can be generated at high data rates. The performance monitoring system should be able to handle such operating conditions.
- Minimal measurement overhead: If measurements are taken often, the measurement itself must not be intrusive. Further, there must be a way for monitoring facilities to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.
- Secure. Typical user actions will include queries to the directory service concerning event data availability, subscriptions for event data, and requests to instantiate new event monitors or to adjust collection parameters on existing monitors. The data gathered by the system may itself have access restrictions placed upon it by the owners of the monitors. The monitoring system must be able to ensure its own integrity and to preserve the access control policies imposed by the ultimate owners of the data.
- Scalable. Because there are potentially thousands of resources, services, and applications to monitor and thousands of potential entities that would like to receive this information, it is important that a performance monitoring system provide scalable measurement, transmission of information, and security.

#### 5.4 *Architecture*

The architecture of Alchemi that we started with is:

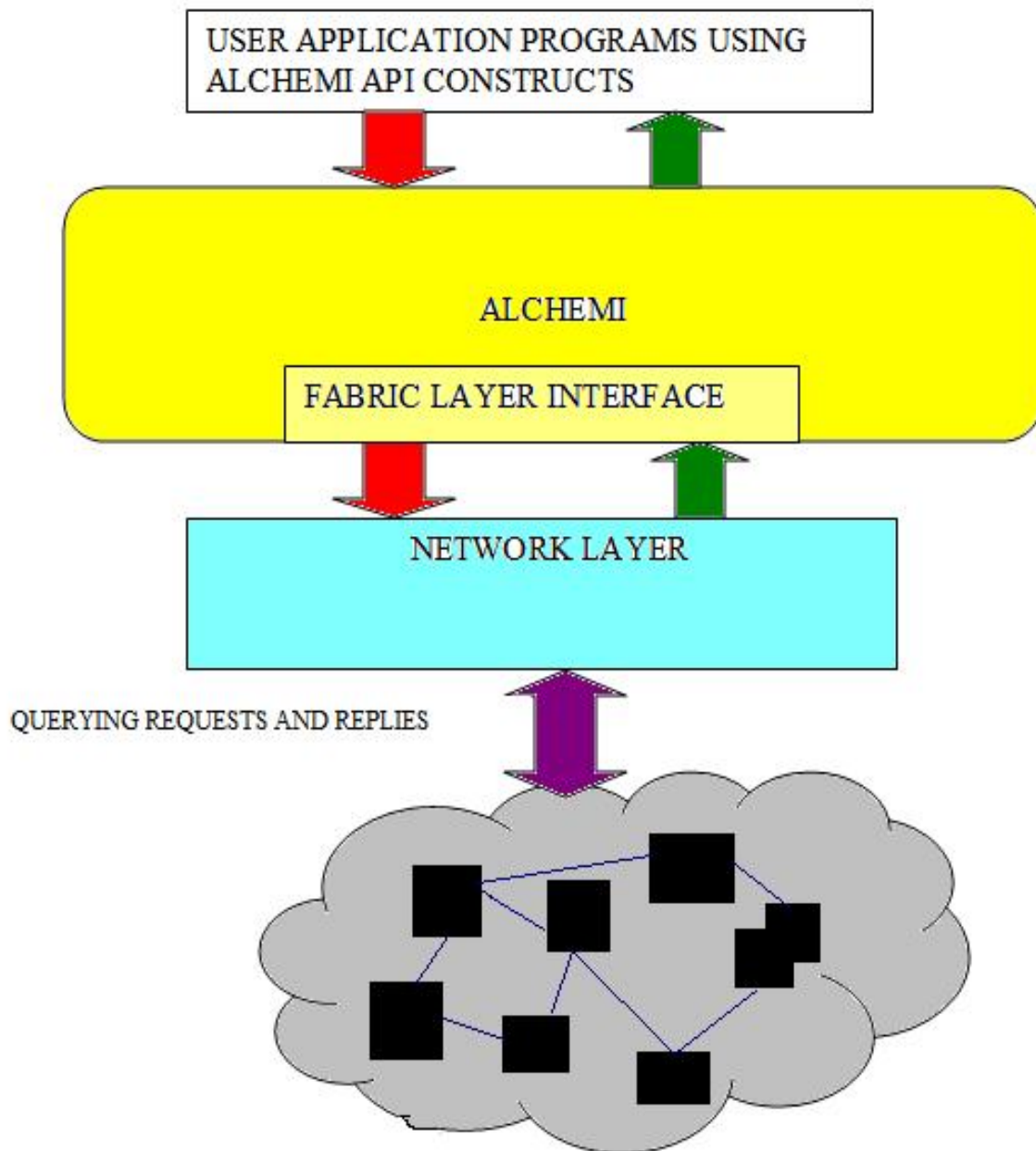


Fig 5.b Grid Architecture

## Chapter 6

### *Implementation, Measurement and Analysis*

#### 6.1 Network Setup

Figure 6.a and 6.b show the network setup for the test-bed in the laboratory. Hosts A and B run Windows XP SP2 operating system while Host C has the Red Hat Linux Fedora Core 3 (FC3) operating system. DSRouter1 (R1), DSRouter2 (R2), and DSRouter3 (R3) are software routers, which run on the FC3 operating system. R1 and R2 have two 10/100 Mbps Ethernet network interface cards while R3 has three 10/100Mbps NICs with no quality of service support. Hosts-to-router links are 100 Mbps point-to-point links. Router-to-router links are 10 Mbps point-to-point links. Hosts use default routing to send traffic to the first - hop routers. Routers are configured for static routing because we did not use any middleware that could do resource allocation based on resource utilization measures and the paths were therefore known beforehand. Our experiments are not affected in any way through this choice because we are not concerned with the job scheduling strategies in a Grid.

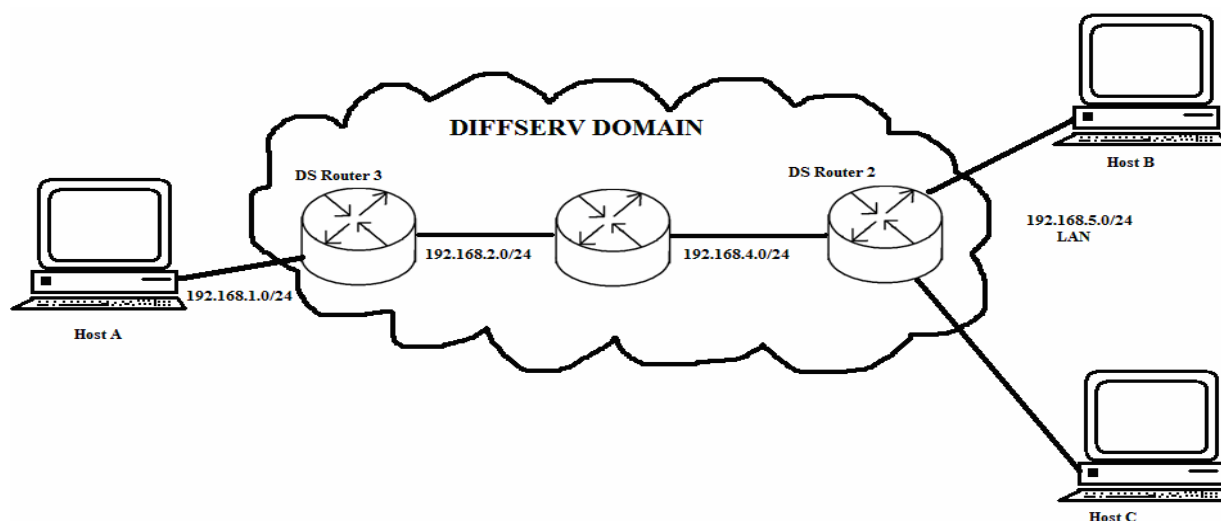


Fig 6.a Test bed Setup of our Network in Laboratory

Since we tried to setup a DiffServ domain the various network entities represent the components of a real world DiffServ enabled environment. Routers R1, R2, and R3 represent the core routers of the DiffServ domain. The hosts also represent the edge routers of the domain where they do the marking of the TOS fields of the packets.

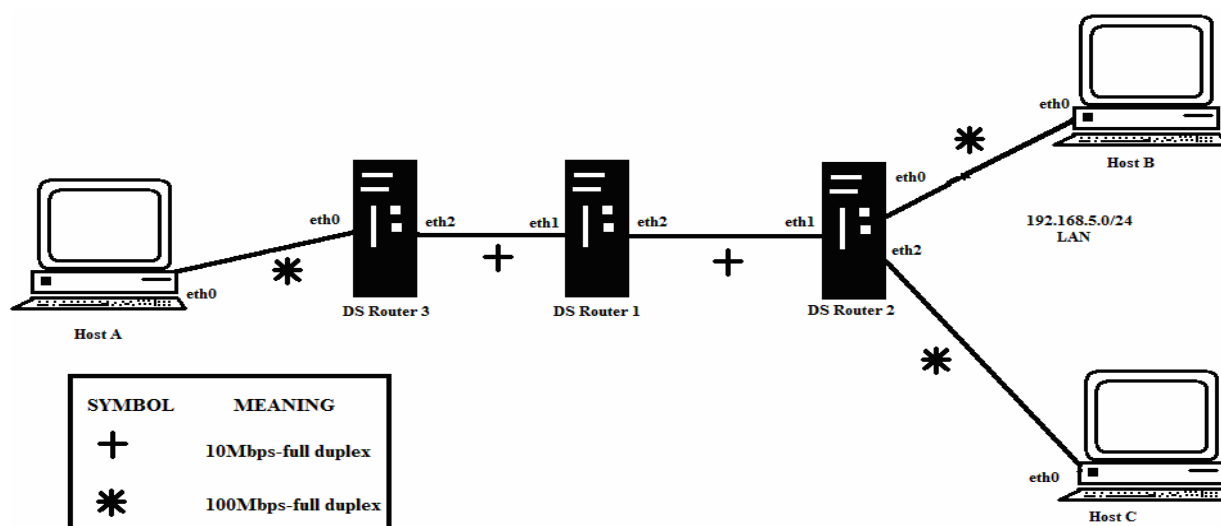


Fig 6.b Network Diagram of Setup showing details of Links

## 6.2 *Measurement Strategies*

Following tools have been chosen keeping in mind their open-source origins and flexibility of use.

- OpenIMP: This tool is used to measure one-way delay and jitter follows the IETF measurement standards and is compliant for both IPv6 and IPv4.
- MGEN: This tool [appendix C] is used to generate traffic flows. Its log files are also helpful in gauging packet loss.
- TC: The tc tool [17, 18] that we have used to implement traffic control (shaping, dropping, policing)
- Ping: ICMP packets were used when setting up the test bed to check the link characteristics of loss and delay.
- SNMP: An application running at host B was used to extract SNMP information that was used to derive IP packet loss and packet losses at the interfaces.

The OpenIMP clients were run on the routers and the data was collected at the end of the experiment at the monitoring host C where the tool used the captured statistics to derive one-way delay, one-way delay distribution and jitter with separate graphs for traffic with different TOS field.

MGEN was run at the transmitting host to generate traffic with different characteristics, more of which is discussed later when its functionalities are defined. A MGEN client was run at the receiver to receive the transmitted data.

Since the measurements are relative to time, we had to make sure the skew between the clocks at the routers and hosts did not affect the results obtained. We used NTPD to synchronize time and reduce skew. More of it is described later. NTPD was run at regular intervals of 1 second when the time at Host C was used as reference point for all the other test-bed entities.

### 6.2.1 Time Synchronization

A key component of any computer analysis may be it is in terms of security or network performance analysis is time. If computers are running on different times, it becomes almost impossible to accurately log packets timings, and hence results in errors in calculation.

The Network Time Protocol (NTP) is used to synchronize the time of a computer client or server to another server or reference time source, in our case a machine in our network. It provides accuracy typically within a millisecond on LAN. On request, the server sends a message including its current clock value or timestamp and the client records its own timestamp upon arrival of the message. For the best accuracy, the client needs to measure the server-client propagation delay to determine its clock offset relative to the server. There are few security issues concerned with the server to prevent any malicious attack which is not considered on our case.

To update from a server: “ntpd -u 192.168.5.3”

To constantly update each two seconds: “watch ntpd -u 192.168.5.3”

NTP version 3 is an internet draft standard, formalized in RFC 1305.

### 6.2.2 Traffic Generation

The Multi-Generator (MGEN) is open source software by the Naval Research Laboratory (NRL) Protocol Engineering Advanced Networking (PROTEAN) group which provides the ability to perform IP network performance tests and measurements using UDP/IP traffic [16]. The toolset generates real-time traffic patterns so that the network can be loaded in a variety of ways. The generated traffic can also be received and logged for analyses. Script files are used to drive the generated loading patterns over the course of time. MGEN log data can be used to calculate performance statistics on throughput, packet loss rates, communication delay, and more. MGEN currently runs on various Unix-based and WIN32 platforms.

Syntax :

```
Mgen[ipv4][ipv6][input<scriptFile>][save<saveFile>][outputlogFile>][log<logFile>][binary][txlog][nolog][flush][hostAddr{on|off}][event"<mgenevent>"][portrecvPortList>][instance<na
```

```
me>][command<cmdInput>][sink<sinkFile>][block][source<sourceFile>][interface<interface  
Name>][ttl<timeToLive>][tos<typeOfService>][label<value>][txbuffer<txSocketBufferSize>]  
[rxbuffer<rxSocketBufferSize>]
```

Since MGEN allows us to set various parameters that define a particular traffic, like video, audio, bulk data etc. we used this tool to generate the required traffic with varying characteristics. This allowed us the flexibility to try out various types of streams for our experiments and validation. For example we defined a multimedia video traffic as having bursts at irregular intervals through MGEN's parameters. The bursts can be generated with varying parameters. The BURST command which is used to write the burst traffic activity script is as follows:

```
BURST [REGULAR|RANDOM <aveInterval (sec)> <patternType> [<patternParams>]  
FIXED|EXPONENTIAL <aveDuration (sec)>]
```

The BURST pattern generates bursts of other MGEN pattern types at a specified average interval. The first parameter of the BURST pattern is either "REGULAR" resulting in periodic burst uniformly distributed in time by the <aveInterval> value, or "RANDOM" which exponentially distributes the traffic generation bursts in time with an average burst interval as specified by the <aveInterval> parameter value. The characteristics of the MGEN messages generated during a burst are given by the <patternType> and associated <patternParams> parameters. The <patternType> may any MGEN pattern type including PERIODIC, POISSON, or, yes, even BURST. The <patternParams> must be appropriate for the given <patternType>. When a traffic generation burst occurs, its duration is either of a FIXED value as given by the <aveDuration> or a randomly varying duration with EXPONENTIAL statistics and an average duration as given by the <aveDuration> parameter. An example use of the BURST pattern would be to roughly emulate the "talk spurts" which might result from Voice Over IP (VOIP) applications. As a voice conversation commences, a user's burst of activity (talk spurts) might be RANDOM with some average interval and the duration talk spurts approximate EXPONENTIAL statistics. When the talk spurt (burst) occurs, the voice compression codec might generate messages following something like a PERIODIC flow with packet rates and packet sizes dependent upon the voice codec in use. Other uses of the BURST pattern might be to roughly model message/packet generation occurring with random use of a network such as web browsing, etc. The BURST model provided by MGEN does not presuppose any specific traffic model, but might be useful in approximating some models of regular or intermittent network activity.

## Chapter 7

### *Results and Analysis*

All the experiments were done by sending traffic between Host A→Host B and Host A→Host C. The OpenIMP tool was used to measure the various characteristics of One-way delay, One-way delay distribution and jitter with the traffic being monitored at the interfaces of DS-Router 2 and DS-Router 3. MGEN was used to generate traffic flows while ICMP ping packets were used to measure delays and average packet losses at various intervals to check the Grid network's state. Time synchronization was done to get consistent results. This chapter describes the experimental setup, time synchronization technique used, traffic generation strategy. Then we discuss our results and analyze their significance in the providing application and fabric layer QoS for critical services. All the experiments discussed in this chapter were jointly performed by a group of research students including present scholar and his associate Mr. Sudeepta Das [19].

#### 7.1 Experiment 1

In this experiment we sought to see the performance of the network when 5 different streams are transmitted from Host B to Host A with a total bandwidth of 10Mbps. Since this is same as the bandwidth offered between the routers and well below 100Mbps capacity of the links connecting the hosts to the routers the very low delay was expected.

The profile of the traffic sent is given in Table 7.a:

Stream	Type	Rate(packets/second)	Size(in bytes)
X1	Periodic	1000	1024
X2	Poisson	2500	1024
X3	Periodic	4000	1024
X4	Poisson	500	1024
X5	Poisson	2000	1024

Table 7.a: Profile of Traffic for Experiment 1

The following figures show the delay and jitter recorded at the two points M and N in the network.

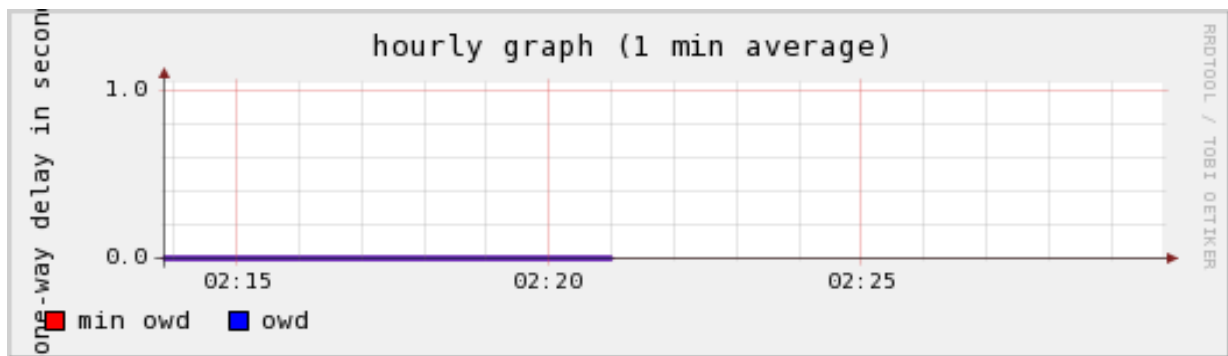


Fig. 7.a One Way Delay Measurement of a band of traffic in 7 min. Interval

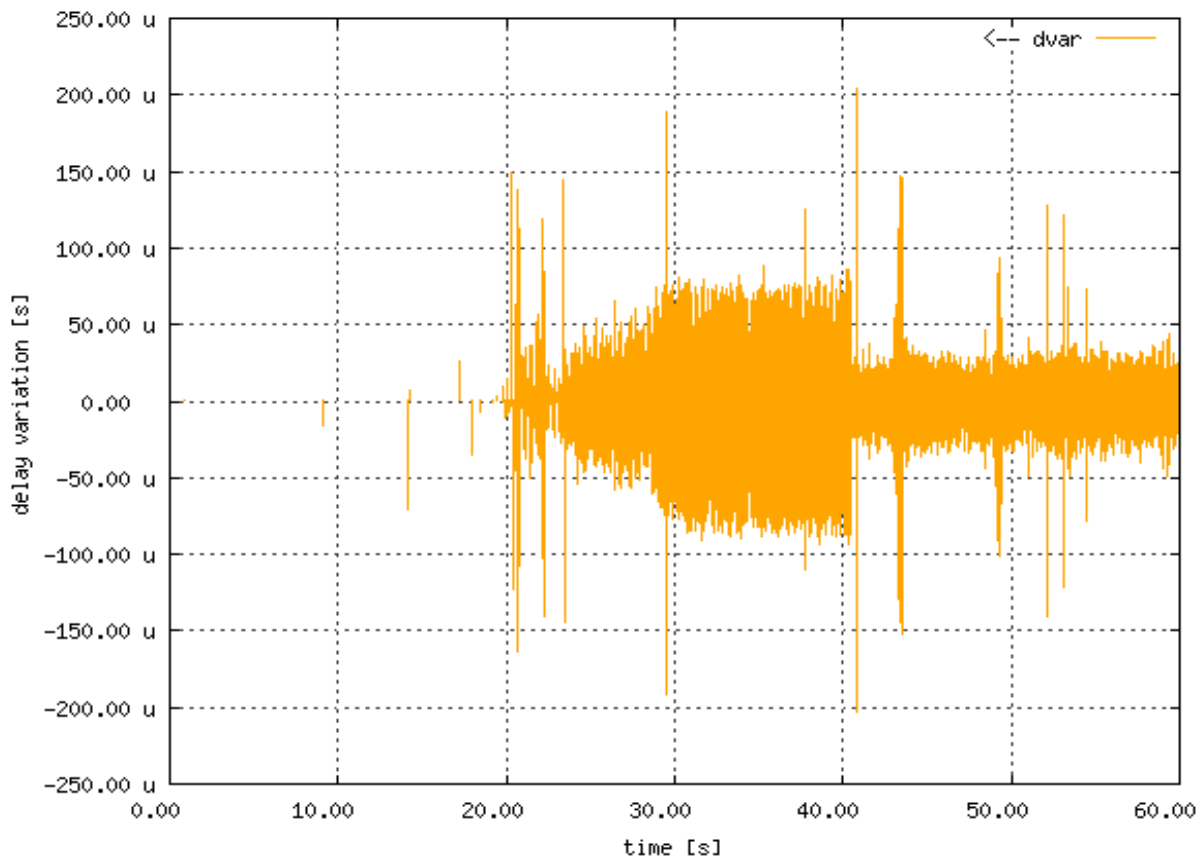


Fig. 7.b Delay Variation (Jitter) of traffic in an interval of 1 min.

We observed from Fig 7.a that the minimum delay and average delay are almost same which is what should be the case when traffic is sent without interference on adequate capacity networks.

The jitter graph displayed in Fig 7.b shows the effect of the traffic profile of the streams. As seen in table 7.a we have used 3 Poisson flows which send packets at statistically varying intervals.

## 7.2 Experiment 2

This experiment was conducted using 2 streams, of which one was multimedia traffic while the other was normal periodic traffic. The bandwidth required is 12Mbps which is greater than 10Mbps link capacity and hence the delays increase as compared to experiment 1. The multimedia traffic being Periodic with regular bursts, it was expected the delays would increase abruptly at burst instances when the traffic bandwidth required would be much greater than that available. The queuing delays increase and average delay shoots up. Table 7.b is the traffic profile while the multimedia traffic's characteristics are described in the MGEN script for Experiment 2 in Appendix 2.

Stream	Type	Rate(packets/second)	Size(in bytes)
Multimedia	Periodic	4500	1024
	Burst (every 4.5 sec.)	4500	1024
Normal	Periodic	3000	1024

Table 7.b: Profile of Traffic for Experiment 2

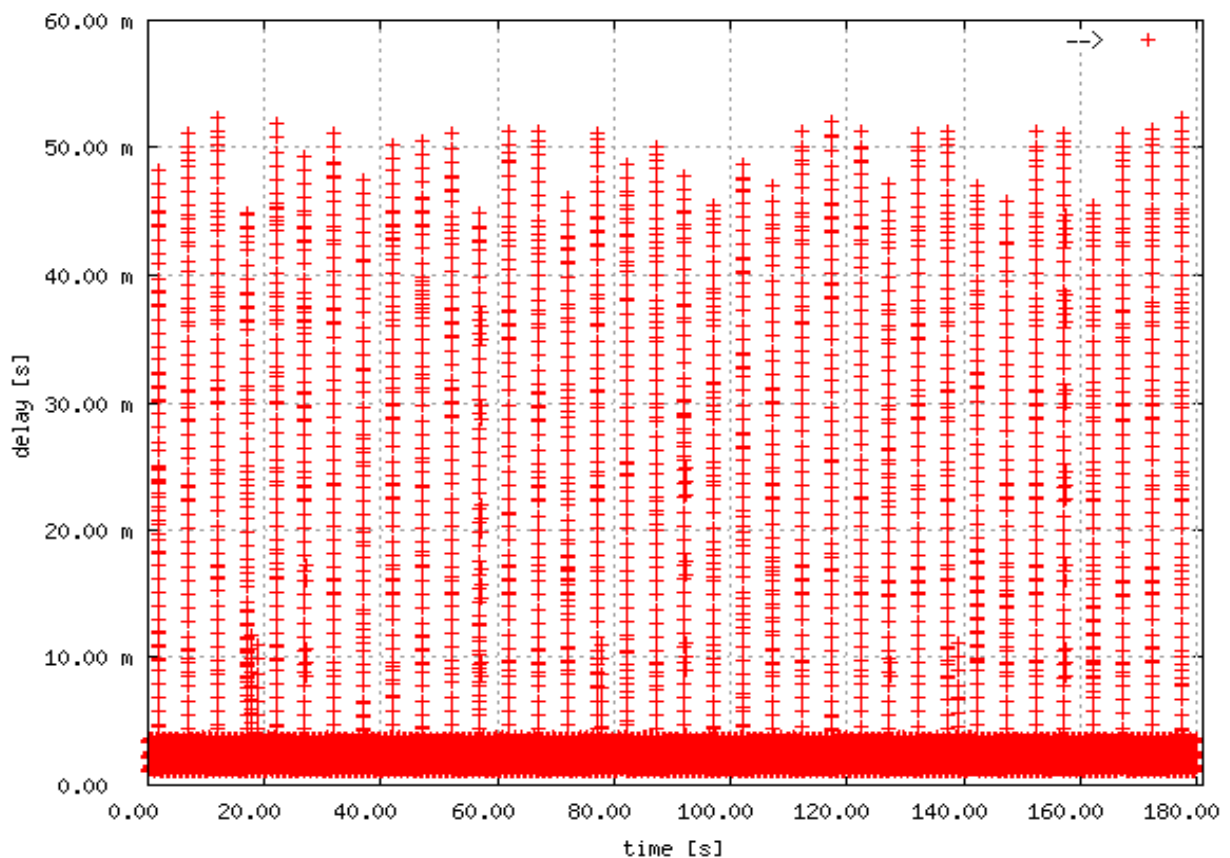


Fig 7.c One-Way Delay over a period of 3 minutes for multimedia and a normal traffic

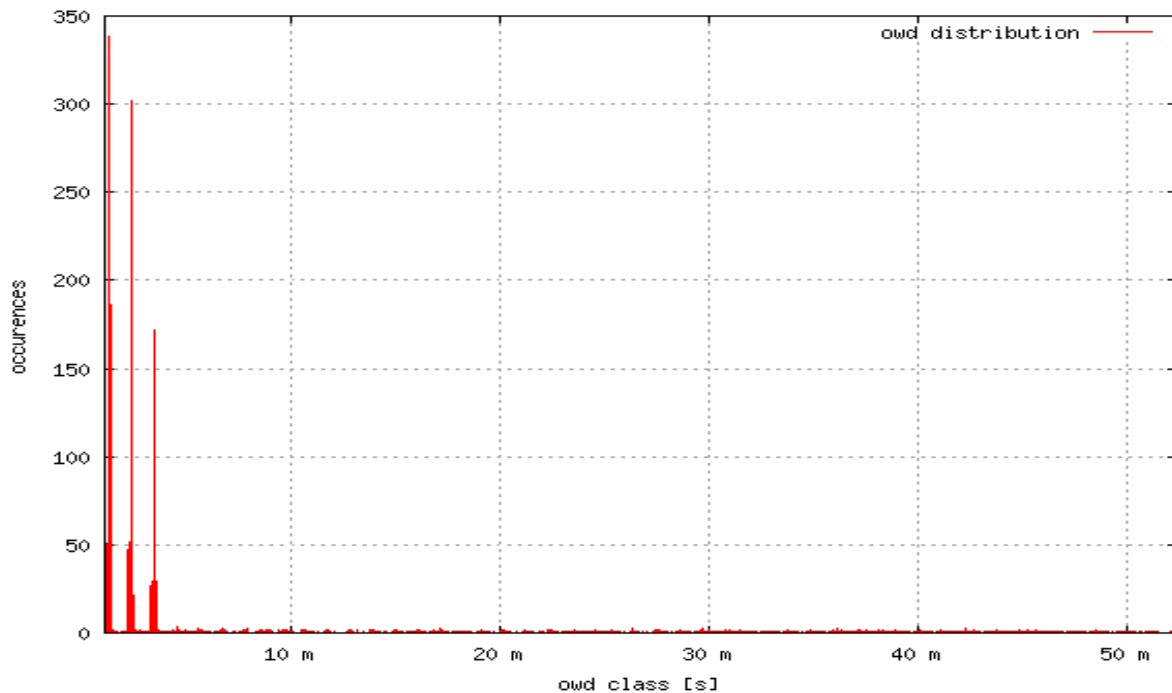


Fig.7.d One-Way Delay Distribution for multimedia and a normal traffic

Fig 7.c shows the one-way delay over a period of 3 minutes with the burst instances being observable by the peaks at regular intervals. Some packet loss was also observed at such bursts when the queue lengths proved to be inadequate to sustain the incoming flows.

Fig 7.d is the one-way delay distribution graph and shows that while majority of the traffic has low delay, the burst instances do cause some packets to be delayed for quite a long time.

Thus it can be observed that some sort of traffic control is required if we are to assure QoS for a flow. If bandwidth were to be reserved for the 2<sup>nd</sup> flow in this experiment, none of its packets would have suffered increased delay when the multimedia flow demonstrated bursts.

### 7.3 Experiment 3

This experiment was conducted using 2 streams, of which one was multimedia traffic while the other was normal Poisson traffic. The bandwidth required is 13Mbps which is greater than 10Mbps link capacity and hence the delays increase as compared to experiment 1. The multimedia traffic being Periodic with random bursts, it was expected the delays would increase abruptly at burst instances when the traffic bandwidth required would be much greater than that available. The queuing delays increase and average delay shoots up. Table 7.c is the traffic profile while the multimedia traffic's characteristics are described in the MGEN script for Experiment 3 in Appendix C.

We implemented static traffic control at the router. Three classes were created under root, multimedia traffic being filtered to first class (sfq) with higher priority and 60% of available bandwidth, normal traffic to next class (tbf) with 20 % of available bandwidth(not bounded), and rest traffic to third default class

The traffic flows are as follows:

Stream	Type	Rate(packets/second)	Size(in bytes)
Multimedia	Periodic	4500	1024
	Burst (randomized)	5000	1024
X1(destination port: 5454)	Periodic	3500	1024

Table 7.c: Profile of Traffic for Experiment 3

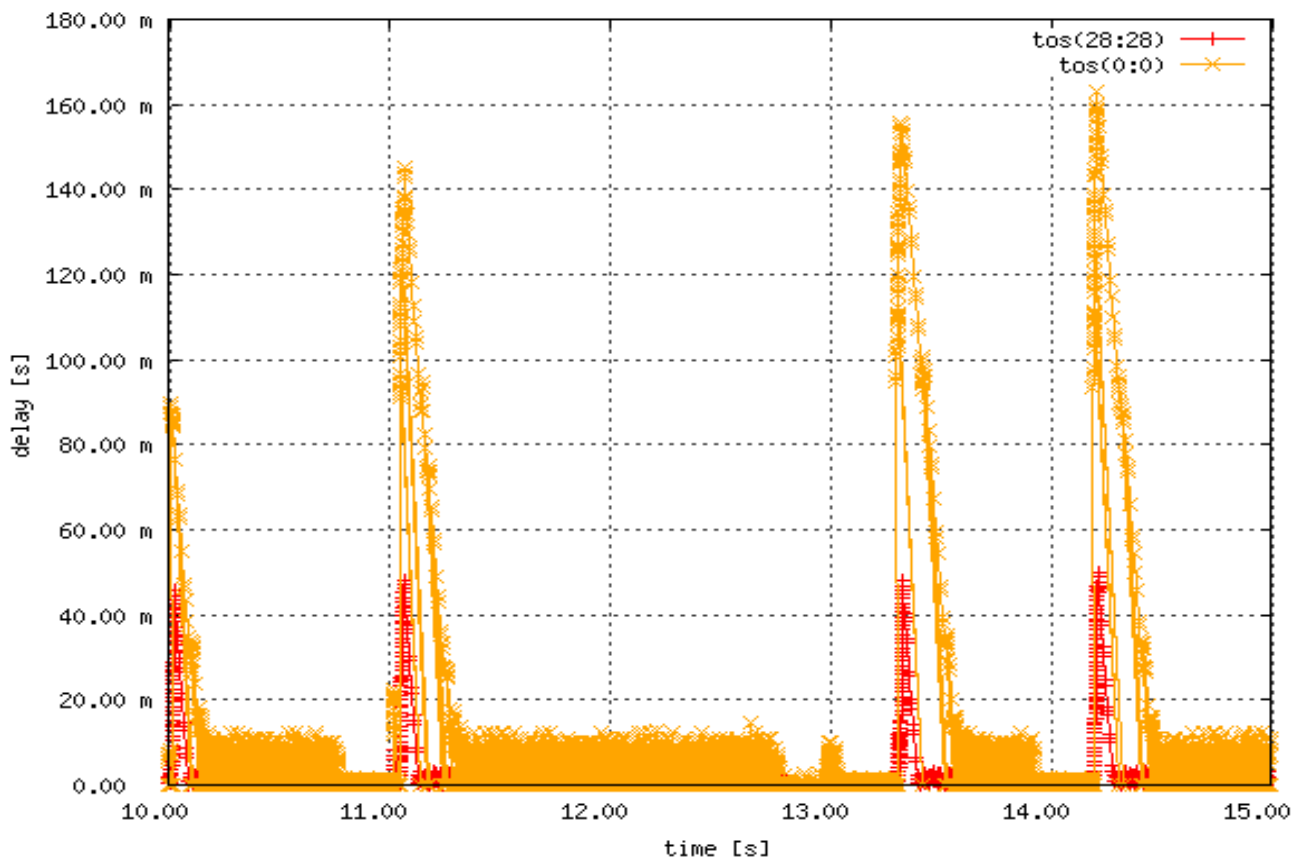


Fig.7.e One Way Delay for two different traffic streams after being shaped

Fig 7.e shows the one-way delay graph. The TOS field of the packets was used to filter the flows and assign them to different classes. The orange stream is the burst traffic while the red

stream is a non-burst one. When there is a burst in orange flow, the delays increase for both the flows. As we observed in experiment 2, these bursts would often lead to packet losses. We also wanted that red traffic should not have delays beyond a certain threshold. So it was assigned a higher priority through the traffic control policy at the routers. This led to higher packet losses for orange flow because the red packets would get queued and transmitted earlier and thus the QoS guarantee of low delay and negligible packet loss was maintained.

#### 7.4 *Experiment 4*

The traffic profile is same as that of Experiment 3.

As we saw in experiment 3, statically setting the traffic policy helps in ensuring the QoS for a particular stream. But since we are speaking of a heterogeneous environment where the user levels may change dynamically and new nodes with varied QoS requirements join the Grid from time-to-time, traffic policy also needs to reflect this dynamic character that is inherent in such cases. For example, suppose the orange stream, which was earlier getting the lowest grade of QoS, suddenly becomes eligible for higher QoS due to the improvement in the Grid membership grade of the node receiving it then there needs to be a way to change the traffic control rules at the routers.

The SNMP monitoring data that is collected by the Grid manager gives account of various performance metrics related to the interfaces and protocols. In this experiment, when the manager realizes that overall traffic is witnessing excessive packet losses due to the current traffic policy it signals the routers to change the policy. In this case, host A which we have assumed to be a virtual manager in our test bed scenario, sends a file to the routers with the new parameters that is utilized by a script running at the routers to increase buffer size for the class containing orange traffic.

Thus we can see that its burst delays spreads and increases slightly and packet losses too decrease as a result of this. The script that was used is given in Appendix C.

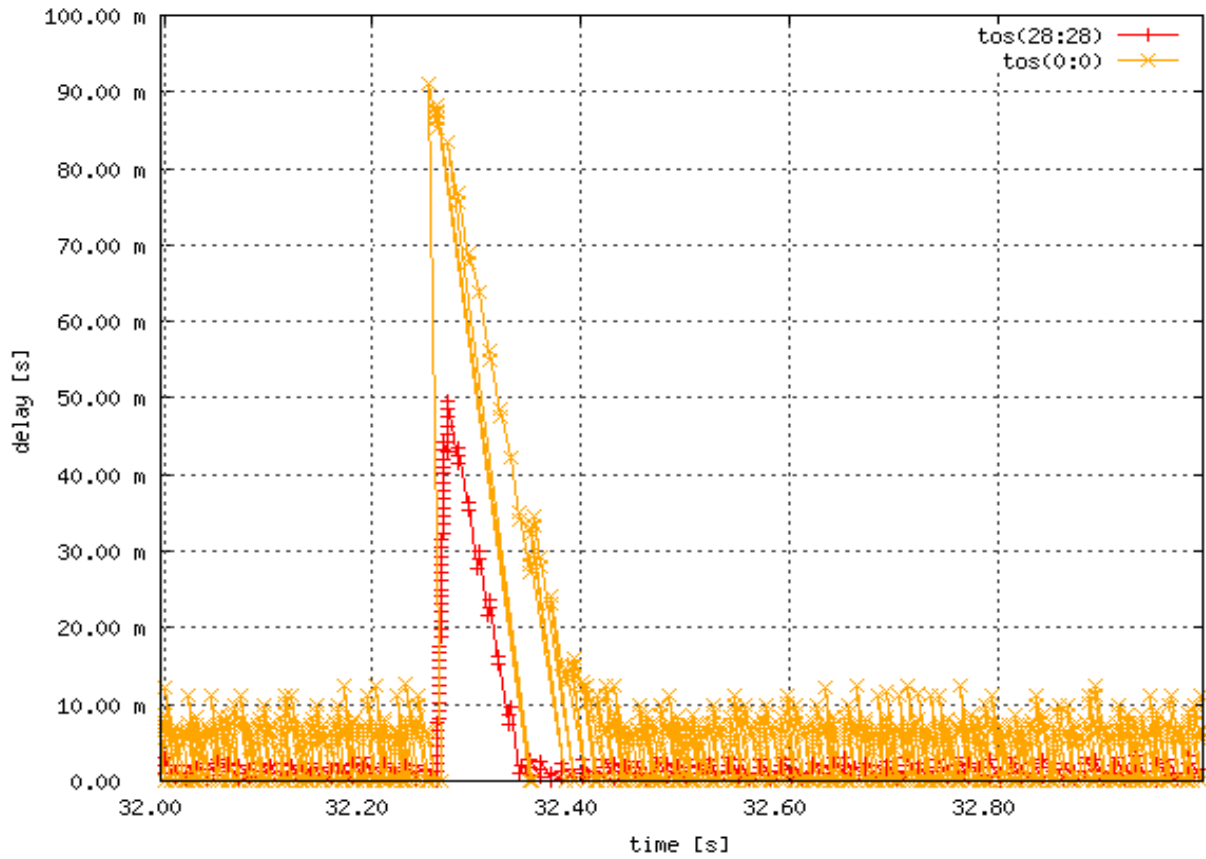


Fig 7.f Traffic of Experiment 3 after being dynamically shaped

The burst delay for orange traffic is now seen to be of the magnitude of 90 milliseconds instead of the average 150 milliseconds witnessed in static traffic control experiment.

## *Conclusion*

A problem of interest in the area of distributed processing and dynamic Grid provisioning in the context of a health care Grid, i-Charak has been examined. We strongly believe that in order to be successful, Grids will need the ability to manage QoS parameters provisioning dynamically. Rather than imposing one implementation, these environments should be able to rely on a mix of technologies as acceptable to site administrators, users and suitable for specific problems. Grid fabrics today present the foremost challenges in this area and there is evidence that dynamic reconfiguration can help greatly.

The project has illustrated several aspects of fabric layer and application layer QoS relevant to deployment of Grid requiring Quality of Service provisions. Experiments investigating methods to provide negotiated QoS for grid applications were carried out. Internet was simulated partially in the test bed through three DiffServ enabled Routers, and three hosts. End to end traffic performance was monitored and controlled in a dynamic manner and compared with the static approach. The feedback obtained by monitoring the dynamic nature of network traffic and prioritized flows was used to modify traffic control scripts at DiffServ Router shaping our traffic. It was observed that an optimal reconfiguration policy can be computed and tabulated subject to complexity constraints imposed by the size of the domain and the ranges of parameter values. Our experimental results show that our approach is a viable and easily implementable way to increase the reliability and throughput of multimedia and other prioritized transactions without bringing any change to existing infrastructure. The encouraging experimental results suggest that its performance compares quite favorably with that of an optimal policy that i-Charak will finally require.

## Appendix A

### *Traffic Control Scripts*

Experiment 3: The following scripts classifies root into two different classes. One with sfq (Stochastic Fairness Queuing) with 75 % of bandwidth, and other being tbf (Token Bucket Filter) with remaining 25 % of the available bandwidth. Multimedia packets with DS field 0x28 (40) go to sfq class and other packets goes to tbf class.

```
#!/bin/bash
TC=/sbin/tc
DV=eth0
$TC qdisc del dev $DV root
$TC qdisc add dev $DV root handle 1 cbq bandwidth 10Mbit avpkt 1000 cell 8
$TC class change dev $DV root cbq weight 1Mbit allot 1514
$TC class add dev $DV parent 1: classid 1:4 cbq bandwidth 100Mbit rate 75Mbit weight
10Mbit prio 1 allot 1514 cell 8 maxburst 20 avpkt 1000
$TC qdisc add dev $DV parent 1:4 handle 4 sfq perturb 10
#Following filters video and audio real time traffic
$TC filter add dev $DV parent 1:0 protocol ip prio 100 u32 match ip dsfield 0x28 classid 1:4
$TC qdisc add dev $DV parent 1: classid 1:5 cbq bandwidth 100Mbit rate 25Mbit weight
2.5Mbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded
```

## Appendix B

### *Shell script for dynamically generating tc scripts.*

```
#!/bin/bash
#
#  cbq.init v0.7.4
#  Copyright (C) 1999 Pavel Golubev <pg@ksi-linux.com>
#  Copyright (C) 2001-2004 Lubomir Bulej <pallas@kadan.cz>
#
#  chkconfig: 2345 11 89
#  description: sets up CBQ-based traffic control
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  To get the latest version, check on Freshmeat for actual location:
#      http://freshmeat.net/projects/cbq.init
#
#  VERSION HISTORY
#  -----
#
#v0.7.4-Vikas Agrawal<to.vikas@gmail.com>
#      -Adds DSCP filtering option in hexadecimal format

set +vx

export LC_ALL=C

#### Command locations
TC=/sbin/tc
IP=/sbin/ip
MP=/sbin/modprobe

#### Default filter priorities (must be different)
PRIO_RULE_DEFAULT=${PRIO_RULE:-100}
PRIO_MARK_DEFAULT=${PRIO_MARK:-200}
PRIO_REALM_DEFAULT=${PRIO_REALM:-300}

#### Default CBQ_PATH & CBQ_CACHE settings
CBQ_PATH=${CBQ_PATH:-/etc/sysconfig/cbq}
CBQ_CACHE=${CBQ_CACHE:-/var/cache/cbq.init}

#### Uncomment to enable logfile for debugging
#CBQ_DEBUG="/var/run/cbq-$1"
```

```
### Modules to probe for. Uncomment the last CBQ_PROBE
### line if you have QoS support compiled into kernel
CBQ_PROBE="sch_cbq sch_tbf sch_sfq sch_prio"
CBQ_PROBE="$CBQ_PROBE cls_fw cls_u32 cls_route"
#CBQ_PROBE=""

### Keywords required for qdisc & class configuration
CBQ_WORDS="DEVICE|RATE|WEIGHT|PRIO|PARENT|LEAF|BOUNDED|ISOLATED"
CBQ_WORDS="$CBQ_WORDS|PRIO_MARK|PRIO_RULE|PRIO_REALM|BUFFER"
CBQ_WORDS="$CBQ_WORDS|LIMIT|PEAK|MTU|QUANTUM|PERTURB"

##### SUPPORT FUNCTIONS #####

### Get list of network devices
cbq_device_list () {
    ip link show | sed -n "/^[0-9]/\
    { s/^[0-9]\+: \([a-z0-9._]\+\):[:@].*\^1/; p; }"
} # cbq_device_list

### Remove root class from device $1
cbq_device_off () {
    tc qdisc del dev $1 root 2> /dev/null
} # cbq_device_off

### Remove CBQ from all devices
cbq_off () {
    for dev in `cbq_device_list`; do
        cbq_device_off $dev
    done
} # cbq_off

### Prefixed message
cbq_message () {
    echo -e "***CBQ: $@"
} # cbq_message

### Failure message
cbq_failure () {
    cbq_message "$@"
    exit 1
} # cbq_failure

### Failure w/ cbq-off
cbq_fail_off () {
    cbq_message "$@"
    cbq_off
    exit 1
}
```

```
} # cbq_fail_off

### Convert time to absolute value
cbq_time2abs () {
    local min=${1##*:}; min=${min##0}
    local hrs=${1%%:*}; hrs=${hrs##0}
    echo $[hrs*60 + min]
} # cbq_time2abs

### Display CBQ setup
cbq_show () {
    for dev in `cbq_device_list`; do
        [ `tc qdisc show dev $dev| wc -l` -eq 0 ] && continue
        echo -e "### $dev: queueing disciplines\n"
        tc $1 qdisc show dev $dev; echo

        [ `tc class show dev $dev| wc -l` -eq 0 ] && continue
        echo -e "### $dev: traffic classes\n"
        tc $1 class show dev $dev; echo

        [ `tc filter show dev $dev| wc -l` -eq 0 ] && continue
        echo -e "### $dev: filtering rules\n"
        tc $1 filter show dev $dev; echo
    done
} # cbq_show

### Check configuration and load DEVICES, DEVFIELDS and CLASSLIST from $1
cbq_init () {
    ### Get a list of configured classes
    CLASSLIST=`find $1 \( -type f -or -type l \) -name 'cbq-*' \
        -not -name '*~' -maxdepth 1 -printf "%f\n"| sort`
    [ -z "$CLASSLIST" ] &&
        cbq_failure "no configuration files found in $1!"

    ### Gather all DEVICE fields from $1/cbq-*
    DEVFIELDS=`find $1 \( -type f -or -type l \) -name 'cbq-*' \
        -not -name '*~' -maxdepth 1 | xargs sed -n 's/#.*//; \
        s/[[:space:]]//g; /^DEVICE=[^,]*,[^,]*\([^,]*\)?/\
        { s/.*/=//; p; }| sort -u`
    [ -z "$DEVFIELDS" ] &&
        cbq_failure "no DEVICE field found in $1/cbq-*!"

    ### Check for different DEVICE fields for the same device
    DEVICES=`echo "$DEVFIELDS"| sed 's/.,*/|' sort -u`
    [ `echo "$DEVICES"| wc -l` -ne `echo "$DEVFIELDS"| wc -l` ] &&
        cbq_failure "different DEVICE fields for single device!\n$DEVFIELDS"
} # cbq_init
```

```
### Load class configuration from $1/$2
cbq_load_class () {
    CLASS=`echo $2 | sed 's/^cbq-0*//; s/^([0-9a-fA-F]\+).*/1/'`
    CFILE=`sed -n 's/#.*//; s/[[:space:]]//g; /^[[:alnum:]]\+=[[:alnum:]].*@\-_\]+$/ p'`
$1/$2`

    ### Check class number
    IDVAL=`/usr/bin/printf "%d" 0x$CLASS 2> /dev/null`
    [ $? -ne 0 -o $IDVAL -lt 2 -o $IDVAL -gt 65535 ] &&
        cbq_fail_off "class ID of $2 must be in range <0002-FFFF>!"

    ### Set defaults & load class
    RATE=""; WEIGHT=""; PARENT=""; PRIO=5
    LEAF=tbfb; BOUNDED=yes; ISOLATED=no
    BUFFER=10Kb/8; LIMIT=15Kb; MTU=1500
    PEAK=""; PERTURB=10; QUANTUM=""

    PRIO_RULE=$PRIO_RULE_DEFAULT
    PRIO_MARK=$PRIO_MARK_DEFAULT
    PRIO_REALM=$PRIO_REALM_DEFAULT

    eval `echo "$CFILE" | grep -E "^( $CBQ_WORDS)="$`

    ### Require RATE/WEIGHT
    [ -z "$RATE" -o -z "$WEIGHT" ] &&
        cbq_fail_off "missing RATE or WEIGHT in $2!"

    ### Class device
    DEVICE=${DEVICE%%,*}
    [ -z "$DEVICE" ] && cbq_fail_off "missing DEVICE field in $2!"

    BANDWIDTH=`echo "$DEVFIELDS" | sed -n "/^$DEVICE,/ \
        { s/[^,]*,\([^,]*\).*/1/; p; q; }"`

    ### Convert to "tc" options
    PEAK=${PEAK:+peakrate $PEAK}
    PERTURB=${PERTURB:+perturb $PERTURB}
    QUANTUM=${QUANTUM:+quantum $QUANTUM}

    [ "$BOUNDED" = "no" ] && BOUNDED="" || BOUNDED="bounded"
    [ "$ISOLATED" = "yes" ] && ISOLATED="isolated" || ISOLATED=""
} # cbq_load_class

##### INIT #####

### Check for presence of ip-route2 in usual place
[ -x $TC -a -x $IP ] ||
    cbq_failure "ip-route2 utilities not installed or executable!"
```

```
### ip/tc wrappers
if [ "$1" = "compile" ]; then
    ### no module probing
    CBQ_PROBE=""

    ip () {
        $IP "$@"
    } # ip

    ### echo-only version of "tc" command
    tc () {
        echo "$TC "$@"
    } # tc

elif [ -n "$CBQ_DEBUG" ]; then
    echo -e "# `date`" > $CBQ_DEBUG

    ### Logging version of "ip" command
    ip () {
        echo -e "\n# ip "$@" >> $CBQ_DEBUG
        $IP "$@" 2>&1 | tee -a $CBQ_DEBUG
    } # ip

    ### Logging version of "tc" command
    tc () {
        echo -e "\n# tc "$@" >> $CBQ_DEBUG
        $TC "$@" 2>&1 | tee -a $CBQ_DEBUG
    } # tc
else
    ### Default wrappers

    ip () {
        $IP "$@"
    } # ip

    tc () {
        $TC "$@"
    } # tc
fi # ip/tc wrappers

case "$1" in

##### START/COMPILE #####

start|compile)

### Probe QoS modules (start only)
```

```
for module in $CBQ_PROBE; do
    $MP $module || cbq_failure "failed to load module $module"
done

### If we are in compile/nocache/logging mode, don't bother with cache
if [ "$1" != "compile" -a "$2" != "nocache" -a -z "$CBQ_DEBUG" ]; then
    VALID=1

    ### validate the cache
    [ "$2" = "invalidate" -o ! -f $CBQ_CACHE ] && VALID=0
    if [ $VALID -eq 1 ]; then
        [ `find $CBQ_PATH -maxdepth 1 -newer $CBQ_CACHE | \
            wc -l` -gt 0 ] && VALID=0
    fi

    ### compile the config if the cache is invalid
    if [ $VALID -ne 1 ]; then
        $0 compile > $CBQ_CACHE ||
            cbq_fail_off "failed to compile CBQ configuration!"
    fi

    ### run the cached commands
    exec /bin/sh $CBQ_CACHE 2> /dev/null
fi

### Load DEVICES, DEVFIELDS and CLASSLIST
cbq_init $CBQ_PATH

### Setup root qdisc on all configured devices
for dev in $DEVICES; do
    ### Retrieve device bandwidth and, optionally, weight
    DEVTEMP=`echo "$DEVFIELDS" | sed -n "/^$dev,/ { s/$dev,;/; p; q; }"`
    DEVBWDT=${DEVTEMP%%,*}; DEWVGHT=${DEVTEMP###*,}
    [ "$DEVBWDT" = "$DEWVGHT" ] && DEWVGHT=""

    ### Device bandwidth is required
    if [ -z "$DEVBWDT" ]; then
        cbq_message "could not determine bandwidth for device $dev!"
        cbq_failure "please set up the DEVICE fields properly!"
    fi

    ### Check if the device is there
    ip link show $dev &> /dev/null ||
        cbq_fail_off "device $dev not found!"

    ### Remove old root qdisc from device
    cbq_device_off $dev
done
```

## Establishment of QoS enabled Multimedia Collaboration Grid over native IPv6 fabric

```
### Setup root qdisc + class for device
tc qdisc add dev $dev root handle 1 cbq \
bandwidth $DEVBWDT avpkt 1000 cell 8

### Set weight of the root class if set
[ -n "$DEVWGHT" ] &&
    tc class change dev $dev root cbq weight $DEVWGHT allot 1514

[ "$1" = "compile" ] && echo
done # dev

### Setup traffic classes
for classfile in $CLASSLIST; do
    cbq_load_class $CBQ_PATH $classfile

### Create the class
tc class add dev $DEVICE parent 1:$PARENT classid 1:$CLASS cbq \
bandwidth $BANDWIDTH rate $RATE weight $WEIGHT prio $PRIO \
allot 1514 cell 8 maxburst 20 avpkt 1000 $BOUNDED $ISOLATED ||
    cbq_fail_off "failed to add class $CLASS with parent $PARENT on
$DEVICE!"

### Create leaf qdisc if set
if [ "$LEAF" = "tbf" ]; then
    tc qdisc add dev $DEVICE parent 1:$CLASS handle $CLASS tbf \
rate $RATE buffer $BUFFER limit $LIMIT mtu $MTU $PEAK
elif [ "$LEAF" = "sfq" ]; then
    tc qdisc add dev $DEVICE parent 1:$CLASS handle $CLASS sfq \
$PERTURB $QUANTUM
fi

### Create fw filter for MARK fields
for mark in `echo "$CFILE"| sed -n '/^MARK/ { s/.*=//; p; }`; do
    ### Attach fw filter to root class
    tc filter add dev $DEVICE parent 1:0 protocol ip \
prio $PRIO_MARK handle $mark fw classid 1:$CLASS
done ### mark

### Create route filter for REALM fields
for realm in `echo "$CFILE"| sed -n '/^REALM/ { s/.*=//; p; }`; do
    ### Split realm into source & destination realms
    SREALM=${realm%%,*}; DREALM=${realm##*,}
    [ "$SREALM" = "$DREALM" ] && SREALM=""

    ### Convert asterisks to empty strings
    SREALM=${SREALM#\*}; DREALM=${DREALM#\*}

    ### Attach route filter to the root class
```

```
tc filter add dev $DEVICE parent 1:0 protocol ip \
prio $PRIO_REALM route ${SREALM:+from $SREALM} \
${DREALM:+to $DREALM} classid 1:$CLASS
done ### realm

### Create u32 filter for RULE fields
for rule in `echo "$CFILE"| sed -n '/^RULE/ { s/.*=//; p; }'^; do
### Split rule into source & destination

#echo "rule is $rule"

SRC=${rule%%,*}; DST=${rule##*,}
[ "$SRC" = "$rule" ] && SRC=""

### Split destination into address, port & mask fields
DADDR=${DST%%:*}; DTEMP=${DST##*:}
[ "$DADDR" = "$DST" ] && DTEMP=""

DPORT=${DTEMP%/*}; DMASK=${DTEMP##*/}
[ "$DPORT" = "$DTEMP" ] && DMASK="0xffff"

### Split up source (if specified)
SADDR=""; SPORT=""
if [ -n "$SRC" ]; then
    SADDR=${SRC%%:*}; STEMP=${SRC##*:}
    [ "$SADDR" = "$SRC" ] && STEMP=""

    SPORT=${STEMP%/*}; SMASK=${STEMP##*/}
    [ "$SPORT" = "$STEMP" ] && SMASK="0xffff"
fi

### Convert asterisks to empty strings
SADDR=${SADDR#\*}; DADDR=${DADDR#\*}

### Compose u32 filter rules
u32_s="${SPORT:+match ip sport $SPORT $SMASK}"
u32_s="${SADDR:+match ip src $SADDR} $u32_s"
u32_d="${DPORT:+match ip dport $DPORT $DMASK}"
u32_d="${DADDR:+match ip dst $DADDR} $u32_d"

### Uncomment the following if you want to see parsed rules
#echo "$rule: $u32_s $u32_d"
```

```
##### Attach u32 filter to the appropriate class
tc filter add dev $DEVICE parent 1:0 protocol ip \
prio $PRIO_RULE u32 $u32_s $u32_d classid 1:$CLASS
done ##### rule

##### Create u32 filter for DSCP fields
for variable in `echo "$CFILE"| sed -n '/^DSCP/ { s/.*=//; p; }'^; do
##### Split rule into source & destination

SRC1=${variable%% };
[ "$SRC1" = "$variable" ] ##&& SRC1=""
#echo "SRC1 is $SRC1"
if [ "$SRC1" ]; then
    echo "Error with DSCP value...Not Present"
fi

##### Convert asterisks to empty strings
SRC1="${variable:+match ip dsfield $variable}"
echo "Extracted Value of SRC1 is $SRC1"
##### Uncomment the following if you want to see parsed rules
#echo "$rule: $u32_s $u32_d"

##### Attach u32 filter to the appropriate class
tc filter add dev $DEVICE parent 1:0 protocol ip \
prio $PRIO_RULE u32 $SRC1 classid 1:$CLASS

done ##### DSCP variable

[ "$1" = "compile" ] ##&& echo
done ##### classfile
;;

##### TIME CHECK #####

timecheck)

##### Get time + weekday
TIME_TMP=`date +%w/%k:%M`
TIME_DOW=${TIME_TMP%/*}
TIME_NOW=${TIME_TMP##*/}

##### Load DEVICES, DEVFIELDS and CLASSLIST
cbq_init $CBQ_PATH

##### Run through all classes
for classfile in $CLASSLIST; do
##### Gather all TIME rules from class config
TIMESSET=`sed -n 's/#.*//; s/[[:space:]]//g; /^TIME/ { s/.*=//; p; }' \
$CBQ_PATH/$classfile`
```

```
[ -z "$TIMESET" ] && continue

MATCH=0; CHANGE=0
for timerule in $TIMESET; do
    TIME_ABS=`cbq_time2abs $TIME_NOW`

    ### Split TIME rule to pieces
    TIMESPEC=${timerule%%;*}; PARAMS=${timerule###;}
    WEEKDAYS=${TIMESPEC%%/*}; INTERVAL=${TIMESPEC###}/
    BEG_TIME=${INTERVAL%%-*}; END_TIME=${INTERVAL###-}

    ### Check the day-of-week (if present)
    [ "$WEEKDAYS" != "$INTERVAL" -a \
      -n "${WEEKDAYS##*$TIME_DOW*}" ] && continue
    BEG_ABS=`cbq_time2abs $BEG_TIME`
    END_ABS=`cbq_time2abs $END_TIME`

    ### Midnight wrap fixup
    if [ $BEG_ABS -gt $END_ABS ]; then
        [ $TIME_ABS -le $END_ABS ] &&
            TIME_ABS=${TIME_ABS + 24*60}

        END_ABS=${END_ABS + 24*60}
    fi

    ### If the time matches, remember params and set MATCH flag
    if [ $TIME_ABS -ge $BEG_ABS -a $TIME_ABS -lt $END_ABS ]; then
        TMP_RATE=${PARAMS%%/*}; PARAMS=${PARAMS###}/
        TMP_WGHT=${PARAMS%%/*}; TMP_PEAK=${PARAMS###}/

        [ "$TMP_PEAK" = "$TMP_WGHT" ] && TMP_PEAK=""
        TMP_PEAK=${TMP_PEAK:+peakrate $TMP_PEAK}

        MATCH=1
    fi
done ### timerule

cbq_load_class $CBQ_PATH $classfile

### Get current RATE of CBQ class
RATE_NOW=`tc class show dev $DEVICE | sed -n \
  "/cbq 1:$CLASS / { s/.*rate //; s/ .*//; p; q; }"`
[ -z "$RATE_NOW" ] && continue

### Time interval matched
if [ $MATCH -ne 0 ]; then

    ### Check if there is any change in class RATE
    if [ "$RATE_NOW" != "$TMP_RATE" ]; then
```

```
        NEW_RATE="$TMP_RATE"
        NEW_WGHT="$TMP_WGHT"
        NEW_PEAK="$TMP_PEAK"
        CHANGE=1
    fi

    ### Match not found, reset to default RATE if necessary
    elif [ "$RATE_NOW" != "$RATE" ]; then
        NEW_WGHT="$WEIGHT"
        NEW_RATE="$RATE"
        NEW_PEAK="$PEAK"
        CHANGE=1
    fi

    ### If there are no changes, go for next class
    [ $CHANGE -eq 0 ] && continue

    ### Replace CBQ class
    tc class replace dev $DEVICE classid 1:$CLASS cbq \
    bandwidth $BANDWIDTH rate $NEW_RATE weight $NEW_WGHT prio $PRIO \
    allot 1514 cell 8 maxburst 20 avpkt 1000 $BOUNDED $ISOLATED

    ### Replace leaf qdisc (if any)
    if [ "$LEAF" = "tbft" ]; then
        tc qdisc replace dev $DEVICE handle $CLASS tbft \
        rate $NEW_RATE buffer $BUFFER limit $LIMIT mtu $MTU $NEW_PEAK
    fi

    cbq_message "$TIME_NOW: class $CLASS on $DEVICE changed rate
($RATE_NOW -> $NEW_RATE)"
done ### class file
;;

##### THE REST #####

stop)
    cbq_off
    ;;

list)
    cbq_show
    ;;

stats)
    cbq_show -s
    ;;

restart)
    shift
    $0 stop
```

## Establishment of QoS enabled Multimedia Collaboration Grid over native IPv6 fabric

```
$0 start "$@"  
;;  
  
*)  
    echo "Usage: `basename $0` {start|compile|stop|restart|timecheck|list|stats}"  
esac
```

The two sample files used for generating a tc script are written below.

```
PATH = "/etc/sysconfig/cbq/cbq-0004.root_shaper"
```

```
DEVICE=eth0,100Mbit  
RATE=95Mbit  
WEIGHT=10Mbit  
PRIO=1  
LIMIT=100Kb  
LEAF=tbf  
LEAF=sfq  
MTU=1500  
BUFFER=50Kb/4  
BOUNDED=no  
DSCP=0x28
```

```
PATH = "/etc/sysconfig/cbq/cbq-0005.sfq_shaper"
```

```
DEVICE=eth0,100Mbit  
RATE=5Mbit  
WEIGHT=10Mbit  
PRIO=5  
PERTURB=10  
LEAF=no
```

## APPENDIX C

### *Traffic Generation Scripts*

#### Experiment 1

##### *MGEN script to simulate experimental traffic*

```
0.0 ON 1 UDP DST 192.168.1.2/5001 PERIODIC [1000 982]
0.0 ON 2 UDP DST 192.168.1.2/5002 PERIODIC [2500 982]
0.0 ON 3 UDP DST 192.168.1.2/5003 PERIODIC [4000 982]
0.0 ON 4 UDP DST 192.168.1.2/5004 PERIODIC [500 982]
0.0 ON 5 UDP DST 192.168.1.2/5005 PERIODIC [2000 982]
```

```
450.0 OFF 1
450.0 OFF 2
450.0 OFF 3
450.0 OFF 4
450.0 OFF 5
```

#### Experiment 2

##### *MGEN script to simulate UDP Video traffic and Junk traffic*

```
0.0 ON 1 UDP DST 192.168.1.2/5006 PERIODIC [4500 982]
0.0 ON 2 UDP DST 192.168.1.2/5007 BURST [REGULAR 4.5 PERIODIC [4500 982]
EXPONENTIAL 0.005]
0.0 ON 3 UDP DST 192.168.1.2/5008 PERIODIC [3000 982]
```

```
200.0 OFF 1
200.0 OFF 2
200.0 OFF 3
```

#### Experiment 3

##### *MGEN script to simulate UDP Video traffic and Junk traffic which is shaped*

```
0.0 ON 1 UDP DST 192.168.1.2/5009 PERIODIC [4500 982]
0.0 ON 2 UDP DST 192.168.1.2/5010 BURST [RANDOM 5.0 POISSON [5000 982]
FIXED 0.005]
0.0 ON 3 UDP DST 192.168.1.2/5011 PERIODIC [3500 982]
```

```
200.0 OFF 1
200.0 OFF 2
200.0 OFF 3
```

## APPENDIX D

### *Route Configuration*

#### Router 1

*“/etc/sysconfig/network-scripts/route-eth1”*

192.168.4.0/24 via 192.168.4.2 dev eth2  
192.168.1.0/24 via 192.168.2.2  
fec0:0:1:3::/64 via fec0:0:1:300::2  
fec0:0:1:200::2/128 via fec0:0:1:300::2

*“/etc/sysconfig/network-scripts/route-eth2”*

192.168.2.0/24 via 192.168.2.2 dev eth1  
192.168.1.0/24 via 192.168.2.2  
192.168.5.0/24 via 192.168.4.3 dev eth2

*“/etc/sysconfig/network”*

HOSTNAME=QoSRouter1  
NETWORKING=yes  
FORWARD\_IPV4=yes  
IPV6INIT=yes  
NETWORKING\_IPV6=yes  
IPV6FORWARDING=yes

*“/etc/sysconfig/network-scripts/ifcfg-eth1”*

DEVICE=eth1  
ONBOOT=yes  
BOOTPROTO=static  
HWADDR=00:10:B5:A3:F1:A6  
IPADDR=192.168.2.3  
NETMASK=255.255.255.0  
IPV6INIT=yes  
IPV6ADDR=fec0:0:1:300::1/56

*“/etc/sysconfig/network-scripts/ifcfg-eth2”*

DEVICE=eth2  
ONBOOT=yes  
BOOTPROTO=static  
HWADDR=00:10:B5:EE:83:94  
IPADDR=192.168.4.2  
NETMASK=255.255.255.0  
IPV6INIT=yes  
IPV6ADDR=fec0:0:1:1::1/64

## Router 2

*“/etc/sysconfig/network-scripts/route-eth0”*

fec0:0:1:2::/64 via fec0:0:1:0200::1  
fec0:0:1:0100::1/128 via fec0:0:1:0200::1

*“/etc/sysconfig/network-scripts/route-eth1”*

192.168.2.0/24 via 192.168.4.2  
192.168.1.0/24 via 192.168.4.2  
fec0:0:1:1::/64 via fec0:0:1:0300::1  
fec0:0:1:0100::2/128 via fec0:0:1:0300::1

*“/etc/sysconfig/network”*

NETWORKING=yes  
HOSTNAME=QoSRouter2  
IPV4\_FORWARD=yes  
IPV6INIT=yes  
NETWORKING\_IPV6=yes  
IPV6FORWARDING=yes

*“/etc/sysconfig/network-scripts/ifcfg-eth0”*

DEVICE=eth0  
BOOTPROTO=static  
ONBOOT=yes  
TYPE=Ethernet  
HOSTNAME=QoSRouter2  
IPADDR=192.168.5.2  
NETMASK=255.255.255.0  
IPV6INIT=yes  
IPV6ADDR=fec0:0:1:200::2/56

*“/etc/sysconfig/network-scripts/ifcfg-eth1”*

DEVICE=eth1  
BOOTPROTO=static  
ONBOOT=yes  
TYPE=Ethernet  
HOSTNAME=QoSRouter2  
IPADDR=192.168.4.3  
NETMASK=255.255.255.0  
IPV6INIT=yes  
IPV6ADDR=fec0:0:1:300::2/56

### Router 3

*“/etc/sysconfig/network-scripts/route-eth0”*

fec0:0:1:1::/64 via fec0:0:1:100::2

fec0:0:1:300::1/128 via fec0:0:1:100::2

*“/etc/sysconfig/network-scripts/route-eth2”*

192.168.4.0/24 via 192.168.2.3

192.168.5.0/24 via 192.168.2.3

*“/etc/sysconfig/network”*

NETWORKING=yes

FORWARD\_IPV4=yes

HOSTNAME= QoSRouter3

IPV6INIT=yes

NETWORKING\_IPV6=yes

IPV6FORWARDING=yes

*“/etc/sysconfig/network-scripts/ifcfg-eth0”*

DEVICE=eth0

ONBOOT=yes

BOOTPROTO=static

IPADDR=192.168.1.3

NETMASK=255.255.255.0

IPV6INIT=yes

IPV6ADDR=fec0:0:1:100::1/56

*“/etc/sysconfig/network-scripts/ifcfg-eth2”*

DEVICE=eth2

ONBOOT=yes

BOOTPROTO=static

IPADDR=192.168.2.2

NETMASK=255.255.255.0

IPV6INIT=yes

IPV6ADDR=fec0:0:1:2::1/64

## *References*

1. Rahul Banerjee: "GridOne: An IPv6-aware GRID Computing Architecture", EuroIndia-2004, 24-26 March 2004, C8: 1200-1230
2. Mark Baker, Rajkumar Buyya, and Domenico Laforenza, Grids and Grid Technologies for Wide-Area Distributed Computing, International Journal of Software: Practice and Experience (SPE), Volume 32, Issue 15, Wiley Press, USA, Nov. 2002
3. Amit sheth, Jorge Cardoso, "QoS for Service-oriented Middleware", Proceedings of the conference on Systemics, Cybernetics, and Informatics, Orlando, FL, July 2002
4. S.Blake, M.Carlson, E.Davies, Z.Wang, "An Architecture for Differentiated Service", RFC 2475
5. Volker Sander, "Networking Issues for Grid Infrastructure, *GFD-I.037*", Grid High Performance Networking Research Group, November 2004
6. R. Yavatkar, D. Pendarakis, R. Guerin, "A Framework for Policy-based Admission Control", RFC 2753
7. Mark Baker, Rajkumar Buyya, and Domenico Laforenza, "Grids and Grid Technologies for Wide-Area Distributed Computing, International Journal of Software: Practice and Experience (SPE)", Volume 32, Issue 15, Wiley Press, USA, Nov. 2002.
8. Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal, "Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies, High Performance Computing: Paradigm and Infrastructure", ISBN: 0-471-65471-X, Wiley Press, New Jersey, USA, June 2005
9. I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International J. Supercomputer Applications, vol. 15, no. 3, pp. 200-222, 2001.
10. N.K. Sharda, "Multimedia Information Networking", Prentice-Hall, 2002.
11. S. Jha, M. Hassan, "Engineering Internet QoS", Artech House, 2002.
12. X. Tang et al., "QoS Provisioning Using IPv6 Flow Label In the Internet, " Proceedings of the 2003 Joint Conference of the Fourth International Conference on Information, Communications & Signal Processing and Fourth Pacific-Rim Conference on Multimedia, vol. 2, pp. 1253 - 1257, 15-18 Dec. 2003.
13. D. Miras et al., *Network QoS Needs of Advanced Internet Applications - A Survey*, Internet2 QoS working group draft, work in progress, Nov. 2002.

14. Joshy Joseph, Craig Fellenstein, Grid Computing, IBM Press 2004
15. The Alchemi Open Source Grid Project, [www.alchemi.net](http://www.alchemi.net)
16. MGEN-The Multi-Generator Toolset homepage; <http://mgen.pf.itd.nrl.navy.mil/>
17. B. Hubert et al., Linux Advanced Routing & Traffic Control HOWTO home page; <http://www.lartc.org/>
18. F. Yin, “IPv6 DiffServ Study and Test Report”, STC Internal Report, Jan. 2003.
19. Sudepta Das, “Investigations into Performance Evaluation of Fabric level and Application level QoS Guarantee in Grid Environment” Dissertation Report, 71p, BITS Pilani, India, December 2005
20. Piyush Gupta, “Investigations into Design and Implementation of an IPv6-QoS-Aware Grid”, Master’s Thesis, Chalmers University of Technology, November 2005