

**Investigations into Performance Evaluation of
Fabric level and Application level QoS
Guarantees in Grid Environment**

By

Sudeepta Das

ID No. 2003H103423



Birla Institute of Technology and Science

Pilani, Rajasthan, India

(December 2005)

**Investigations into Performance Evaluation of Fabric
level and Application level QoS Guarantees in Grid
Environment**

DISSERTATION

Submitted in partial fulfilment of the requirements of
BITS G629T

By

Sudeepta Das 2003H103423

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



Birla Institute of Technology and Science

Pilani, Rajasthan, India

(November 2005)

ACKNOWLEDGMENTS

I would like to express my gratitude to **Dr. Rahul Banerjee**, Associate Professor, Department of Computer Science and Information Systems, Assistant Dean DLPD, BITS, Pilani, for his meticulous guidance, keen interest, continuous encouragement and valuable suggestions throughout my present work. Apart from guiding me, he instilled in me the confidence and interest to pursue research. The regular brainstorming sessions helped in analysing the problem and devising the solution that was finally proposed. His critical assessments and feedback at each stage of the dissertation clarified my ideas and sustained the focus amidst the plethora of possible avenues opened up intermittently due to the nature of the topic. I learnt the value of diligence and sincerity from the examples that he himself would set. The periods fraught with indecision and apprehension when things seemed to be stuck at some stage of the problem were quickly dispensed off during our review sessions. While working with a team I learnt how to manage responsibilities and be accountable for one's actions. I was given a chance to work on a cutting edge technology that would never cease to excite me and the fact that it was so actively reciprocated by him made the effort worthwhile.

I had the opportunity to work with a talented group of people who were the members of the Grid-One Project team but more importantly great friends. Vikas Agrawal's contribution in the traffic control and multimedia QoS part along with his help in formulating the experiments and analysing the results enabled the fruitful completion of my work. I thank Amit Vyas for setting up the performance measurement tool. My friend Prabhu, with his effervescent optimism gave me the courage to surmount any difficulties. I appreciate the congenial company of my hostel mates during this dissertation period. Thanks to Amar, Rishi and the CSD teams for their co-operation. Last but not least I would like to extend my heartfelt gratitude to my parents for their constant encouragement helped me.

(SUDEEPTA DAS)



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

CERTIFICATE

This is to certify that the thesis entitled “Investigations into Performance Evaluation of Fabric level and Application level QoS Guarantees in Grid Environment” is the bona fide work by Sudepta Das (2003H103423) 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 course BITS G629T DISSERTATION, to my satisfaction.

Date:

Dr. Rahul Banerjee
(Supervisor)

ABSTRACT

This report examines the aspect of QoS in the Fabric layer and Application layer. I-Charak, a project that seeks to install a Grid catering to critical health services in rural areas was the motivating factor in seeking out methods of ensuring QoS for multimedia applications. The metrics of jitter, packet loss, delay and throughput are the relevant in this context. The dynamic nature of Grids 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. SNMP may be used for monitoring the network interfaces and protocols while traffic control is used to apply the traffic policy at intermediate routing devices. While the metrics are monitored locally at each of participating nodes and routers, a controlling node the Grid manager collects and collates this information for keeping track of the overall performance. It also takes decisions to allow any QoS seeking application to avail the services at any particular instant and making sure that the guarantees are adhered to for any particular transaction.

The test-bed is based on the assumption that the Grid is composed of entities communicating over the Internet that supports Diffserv framework. 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. Our experimental results show that our approach is a viable and simple way to increase the reliability and throughput of multimedia and other prioritized transactions.

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
SNMP	Simple Network Management protocol
PHB	Per Hop Behaviour
IETF	Internet Engineering Task Force
TOS	Type of Service

LIST OF FIGURES

Fig No.	Figure	PAGE
2.a	Simple Grid	9
2.b	Grid Layers	10
2.c	Globus Architecture	12
2.d	Gridbus Architecture	12
2.e	Integrated Gridbus Architecture	15
3.a	Traffic Control Schema	18
3.b	QoS levels	21
3.c	MIB – II Objects	27
4.a	Network Characteristics for Grid	30
4.b	Grid QoS Design	32
5.a	Laboratory Test-bed Diagram	34
5.b	Laboratory Test-bed Diagram with Link information	35
5.c	Case diagram for MIB-II ip Group	39
5.d	Average One-way delay for Test 1	43
5.e	Delay Variation (Jitter) of traffic in 1 minute interval	44
5.f	One-Way Delay over a period of 3 minutes for multimedia and a normal traffic	45
5.g	One-Way Delay Distribution for multimedia and a normal traffic	46
5.h	One Way Delay for two different traffic streams after being shaped	47
5.i	Traffic of Experiment 3 after being dynamically shaped	49

LIST OF TABLES

Fig No.	Table	PAGE
2.a	Globus Vs Gridbus	13
2.b	Network QoS Parameters	22
2.c	ICMP Packet Types	29
2.d	MIB-2 Objects used for Network Statistics	28
5.a	Experiment 1 Traffic Profile	43
5.b	Experiment 2 Traffic Profile	45
5.c	Experiment 3 Traffic Profile	47

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 Background	2
1.2 Motivation.....	2
1.3 Problem Description.....	3
1.4 Objectives.....	6
1.5 Report Structure.....	7
2. Grid Technology.....	8
2.1 Middleware.....	9
2.2 Architecture.....	10
2.3 Why Gridbus-Alchemi ?.....	12
3. Fabric Layer Quality of Service and Monitoring.....	16
3.1 Need for QoS	16
3.2 QoS Frameworks.....	20
3.2.1 Integrated Services.....	22
3.2.2 Differentiated Services.....	24
3.3 QoS Provisions in IPv4/v6.....	25
3.4 SNMP Monitoring Protocol.....	26
4. Design.....	29
4.1 Goals.....	29
4.2 Constraints.....	30

4.3 Issues.....	31
4.4 Description.....	32
5. Implementation, Results and Analysis.....	34
5.1 Test-bed Setup.....	34
5.2 Measurement Methods and Tools.....	36
5.2.1 SNMP Monitoring.....	37
5.2.2 Time Synchronization.....	40
5.2.3 Traffic Generation.....	41
5.3 Experimental Analysis.....	43
5.3.1 Test 1 – Normal Traffic.....	43
5.3.2 Test 2 – Bursty Multimedia Traffic.....	44
5.3.3 Test 3 – Static QoS.....	47
5.3.4 Test 4 – Dynamic QoS.....	48
6. Conclusion.....	50
<i>Appendix A : Traffic Control and Dynamic generation Scripts.....</i>	<i>51</i>
<i>Appendix B : Traffic Generation Scripts.....</i>	<i>66</i>
<i>Appendix C : Router Configuration Scripts.....</i>	<i>67</i>
REFERENCES.....	70

1: INTRODUCTION

The concept of Grid Computing reflects a paradigm shift in the manner in which the computing world has been traditionally approaching the means to extend, optimize and scale existing technology. Grid computing, instead of solely depending on architectural enhancements of the computing hardware, as manifested by supercomputers, aims to increase data storage and computing capabilities manifold by utilizing widely distributed, heterogeneous devices, running on varied platforms on numerous interconnected networks. It provides highly scalable, highly secure and extreme high performance mechanisms for discovering and negotiating access to remote computing resources in a seamless manner.

Grid computing has the design goal of solving problems too big for any single supercomputer, whilst retaining the flexibility to work on multiple smaller problems. Thus grid computing provides a multi-user environment. Its secondary aims are: better exploitation of the available computing power, and catering for the intermittent demands of large computational exercises.

This implies the use of secure authorization techniques to allow remote users to control computing resources.

Grid computing involves sharing heterogeneous resources (based on different platforms, hardware/software architectures, and computer languages), located in different places belonging to different administrative domains over a network using open standards. In short, it involves virtualizing computing resources. [1]

Grid computing environments must be constructed upon the following foundations:

- Coordinated resources
- Open standard protocols and frameworks
- Provision of Quality of Service (QoS) for end-users

This report deals with the QoS issues with specific focus on the network and application layer issues as manifested with respect to corresponding grid architecture layers.

1.1 BACKGROUND

Project Grid-One, initiative taken at CSD, BITS - Pilani, is a two-part experimental research project. It has been conceptualized to be implemented in two phases [2].

First Phase: Setting up a campus-wide IPv6 native support based grid supporting heterogeneous platforms and devices including mobile computing nodes. [3]

Second Phase: Scaling the initial grid to a larger IPv6 enabled grid. “i-Charak” – A Grid Enabled Healthcare Support System, will be the initial real world project which would be a part of the grid initiative and enable us to test its efficacy in the targeted environment, more of which is discussed in the next sub-section. The specifics of the project including the member listing can be accessed at Project Grid-One website.

The work that was the focus of this dissertation was to investigate into the performance of Grid fabric layer and try to formulate Application and Fabric level QoS provisions for a middleware. We also tried to study the performance benefits of dynamic traffic control based on monitored values from the Grid network as compared to statically deciding on a traffic conditioning policy at the start of the transaction or computation job. The evaluation was to be done on IPv4/v6 testbed. However the results enumerated in the report are based on IPv4 traffic. We feel the use of IPv6 would reduce processing delays in the nodes since the header formats are fixed. We have used fields that do not change in their positioning in the IPv4 header and hence our results are not influenced by this problem of IPv4.

1.2 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 Grid-One’s endeavour 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. [4]

1.3 PROBLEM DESCRIPTION

Large scale grids are complex systems, composed of a large number of components belonging to disjoint domains. Planning the capacity to guarantee quality of service (QoS) in these environments is a challenge because global Service Level Agreements (SLAs) depend on local SLAs, i.e., SLAs established with components that make up the grid. Hence performance monitoring of distributed components is critical for enabling high-performance distributed computing. Menasce and Casalicchio [30] have investigated some of the relevant issues that must be considered in designing grid applications that deliver appropriate QoS and described latency, availability and throughput as some of the core QoS Grid metrics.

Monitoring of data is needed to determine the source of performance problems and to tune the system and application. Fault detection and recovery mechanisms need monitoring data to determine whether a server is down and to decide whether to restart the server or to redirect service requests elsewhere. A performance prediction service takes monitoring data as input to a prediction model, which is in turn used by a scheduler to determine which resources to assign to a job. A Grid monitoring system is differentiated from a general monitoring system in that it must be scalable across wide-area networks and encompass a large number of heterogeneous resources. The monitoring system's naming and security mechanisms must also be integrated with other Grid middleware.

There are different types of network traffic depending upon the type of data that is being transferred. Each type of traffic has its own requirements for the various parameters such as bandwidth, delay, jitter, packet loss rate etc. Data Grids are used to transfer huge amounts of data amongst nodes that are part of the Grid. To achieve high performance but with strict quality control and to aid in propped scheduling decisions, we need a QoS mechanism for Grids. In order to enable the middleware to negotiate and control QoS some basic functionality is needed to present as primitive functions that can be called by application programs being executed by the middleware.

The Application QoS and Fabric QoS are thus closely interrelated in Grids since the upper layer service is based on the lower level's capabilities. We have tried to enable such sort of capabilities in a keeping in mind the Alchemi architecture and hence based it

on the .NET framework. Although Best Effort services is the default QoS on the Internet, Grid infrastructures would require these features to enhance their performance and efficiency along with user satisfaction. Well defined QoS measures also enable the Grid to become more economically viable with appropriate pricing and service.

The Global Grid Forum(GGF)[19] describes core Grid Monitoring Architecture (GMA) components and models for high-level communication between components of different types[23] and discusses in one of its documents [24] a case study of a Grid monitoring system. GridRM [25] is a resource monitoring system for the Grid that is being developed by the Distributed Systems Group at the University of Portsmouth. It can gather data from endpoints filter and fuse this data for subsequent use by a variety of clients. The monitoring system needs to distribute information over the wide area between, so called, GridRM gateways.

Mark Baker et al presented a prototype Grid Monitor that allows the monitoring of Grid information servers at sites registered with the UK Grid Support Centre [21]. The prototype presents metadata from individual Grid sites in a coherent, easy to use format. Fabric resource monitoring is distinguished as that being done on two levels, local and global. The globally monitored data needs to be published using Monitoring and Discovery Services (MDS) [28]. A pluggable driver infrastructure[22][26] used by GridRM to interact with heterogeneous data sources, such as local SNMP or Ganglia agents, provides a homogeneous view of the underlying heterogeneous data . A Java based modular lightweight GMA [27] for GridRM has been developed though the issues of controlling message generation rate and delivery and devising a means of setting the buffer size based on feedback is still unaddressed. Our dynamic traffic control method seeks to address this problem.

Gloperf [32], a network monitoring tool is an integral part of the Globus Grid computing toolkit capable of making end to end TCP measurements requiring no special host permissions. However network infrastructure monitoring exhibits scalability problems in Grids. In principle in a Grid composed of n resources we need to keep record of n^2 end to end paths. Andreozzi et al have suggested a solution [31] that seeks to partition the Grid into domains so that network monitoring is limited to domain to domain connectivity.

Yang et al [29] have mentioned active networks as a possible solution to the monitoring problem for fabric layer resources and used policy based management to have flexible control of QoS parameters. But active networks are a still to be widely implemented and they have their own security and authorization issues. Grids typically handle both Constant Bit Rate (CBR) and Variable Bit Rate (VBR) applications. With Constant Bit Rate (CBR) applications, simple summation of pre-specified bandwidths is a sufficient solution for resource optimization in DiffServ networks. However, the pre-specified bandwidth technique is an inefficient approach to DiffServ bandwidth allocation for VBR applications which are generally unaware in advance of their specific bandwidth requirements. Kulatunga et al [20] have described an adaptive Quality of Service management technique that provides an efficient resource allocation scheme for VBR applications in aggregate traffic flows. Ahmed et al [21] have proposed a method of dynamic QoS adaptation in DiffServ networks using policy based management systems at each of the Diffserv domain nodes. Piyush Gupta has proposed in his dissertation report [38], which was done as a part of the Project Grid-One initiative[2], IPv6 -QoS enabled VPN—IP-in-IPv6 encapsulation with encryption and an elevated quality of network service using Diffserv architecture and IPv6 flow label, useful to connect sites that want to participate privately in a large-scale multi -media grid.

1.4 OBJECTIVES

A basic requirement of a Grid Computing system is the ability to provide the QoS requirements necessary for end-user community. These QoS validations must be a basic feature in any Grid system, and must be done in congruence with the available resources. These QoS features include response time measures, aggregated performance, security fulfilment, resource capability, and availability, autonomic features such as event correlation and configuration management and partial fail over mechanisms.

The key value of a grid is often evaluated on the basis of its business merits and respective user satisfaction. User satisfaction is measured based on the QoS provided by the grid, such as availability, performance, simplicity of access, management aspects, business values, and flexibility in pricing. The business merits often relate to and indicate the problem being solved by the grid. For instance they may be job executions, management aspects, simulation workflows and other key technology based foundations. This report addresses the specific aspect of QoS performance evaluation in grids, for the application and fabric layer. There is huge amount of work being done in various aspects of grids and also on network layer QoS issues but there has been an absence of adequate literature with respect to evaluating high performance optimizations for network QoS issues as applied to grids i.e. the grid fabric layer.

The application layer of a grid middleware has also been analysed with respect to its interactions with the middleware to provide QoS control measures to the end users.

The objectives are enumerated as follows:

- 1: Propose a method to implement QoS in a Data Grid with multimedia traffic, the emphasis being on Fabric level QoS which can be extended for a Grid supporting health care services keeping in mind the critical nature of the problem.***
- 2: Investigate into the monitoring of performance metrics for the Grid Fabric layer and suggest a manner of collecting this information to be used for providing Application QoS through the middleware.***
- 3: Implement traffic control of the data being transmitted with parameters based on the dynamic performance evaluation.***

1.5 REPORT STRUCTURE

The report has been organised in a manner in which the background and motivation for the umbrella project, the specific issues I am dealing with under the aegis of Grid-One , the problem definition , previous work done as per the literature surveyed and objectives are enumerated in the Introduction chapter.

The second chapter briefly defines data and computational grid aspects, describes the various grid layers and their functionalities, takes a look at some of the major middleware initiatives, discusses the tradeoffs between Globus and Gridbus, and gives a brief overview of the Alchemi framework.

The third chapter starts with a description of Grid Quality of Service and then defines QoS frameworks. SNMP protocol as a method of monitoring networks is also briefly explained. Chapter four gives the detailed design along with the goals, constraints and issues of concern to this work.

Chapter five describes the test-bed setup, experiments that were conducted and the results are enumerated. The methods and tools used are also mentioned. The results are analysed and inferences obtained to validate the approach.

Traffic Control Scripts are given in Appendix A. The scripts for generating traffic are in Appendix B while Appendix C contains the routing configuration scripts.

2: GRID TECHNOLOGY

Grids have been broadly classified as

- **Data Grids**
- **Computational Grids**

The core functional data requirements for Grid computing applications are [1], [13]:

- a. Ability to integrate multiple, distributed, heterogeneous, and independently managed data sources.
- b. Ability to provide efficient data transfer mechanisms and to provide data where computation will take place for better scalability and efficiency.
- c. Ability to provide data caching and replication mechanisms to minimize network traffic.
- d. Provision of data discovery mechanisms.
- e. Data encryption capabilities and integrity checks to ensure security in data transfers.
- f. Backup/restore mechanisms and policies to prevent data loss and minimize unplanned downtime.

The core computational data requirements for Grid computing applications are:

- a. Independent management of computing resources
- b. Mechanisms to intelligently and transparently selecting computing resources.
- c. Estimation of current and predicted loads on grid resources, resource availability, dynamic resource configuration and provisioning.
- d. Failure detection and failover mechanisms.
- e. Secure resource management, access control and integrity assurances.

An important requirement common to both is the ability to sustain the required level of QoS while adhering to the anticipated and necessary SLAs (Service Level Agreements).

2.1 MIDDLEWARE

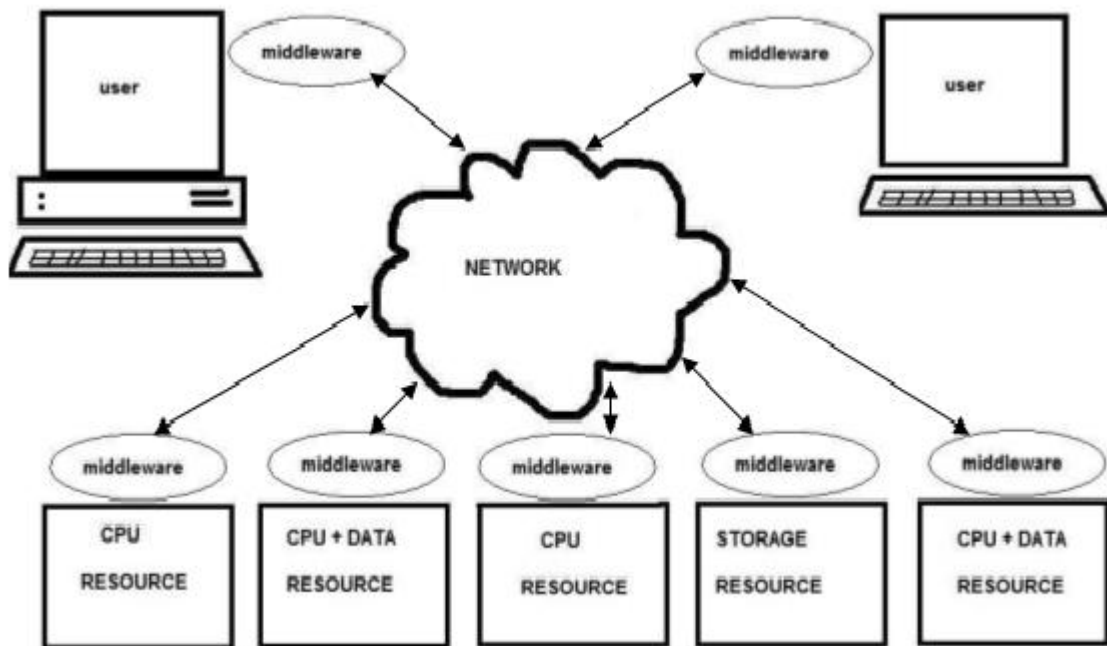


Fig 2.a A Simple Grid

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

- Nimrod
- Legion
- UNICORE (Uniform Interface to Computer Resource)
- NMI (NSF Middleware Initiative)

- Gridbus [6]
- Globus [7]

Both Globus and Gridbus claim to portability and are open source. Globus is the most widely used of all grid middlewares.

2.2 ARCHITECTURE

A new architecture model and technology was developed for the establishment, management, and cross organisational resource sharing within a virtual organisation. The concept of virtual organisation is the key to Grid Computing. It's defined as a dynamic set of individuals and/or institutions defined around a set of resource sharing rules and conditions. 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. The fig 2.d 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. Fig 2.b also shows the corresponding Internet protocol layers.

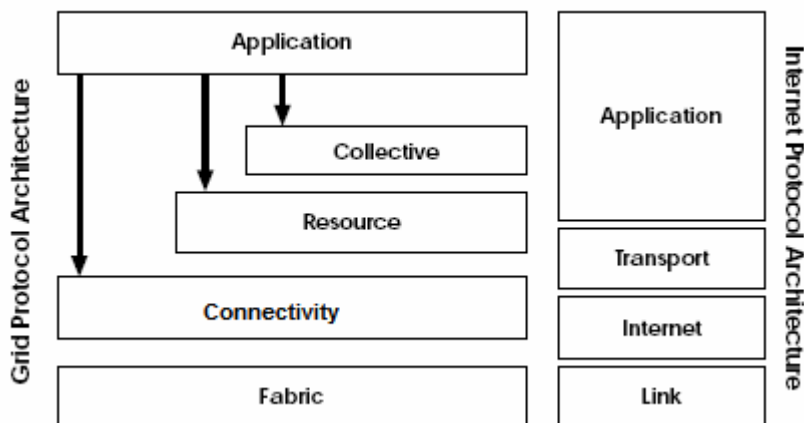


Fig 2.b Grid Layers [1]

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.
- **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 behaviours 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.

2.3 WHY GRIDBUS-ALCHEMI?

Let us take a comparative look at Globus and Gridbus. Fig 2.b and 2.c are diagrammatic representation of the Globus and Gridbus architecture respectively.

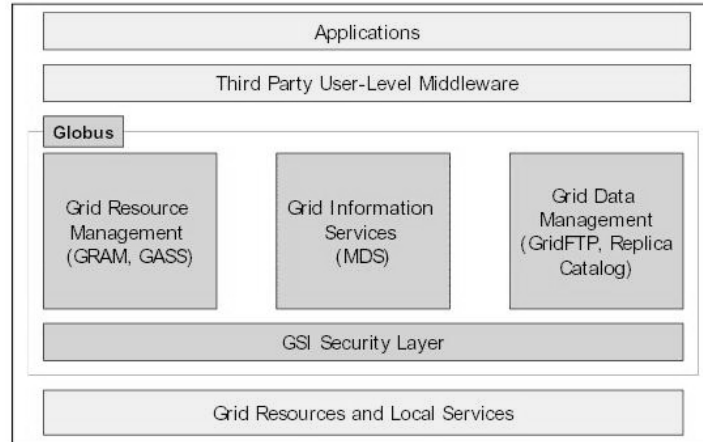


Fig 2.c Globus Architecture [9]

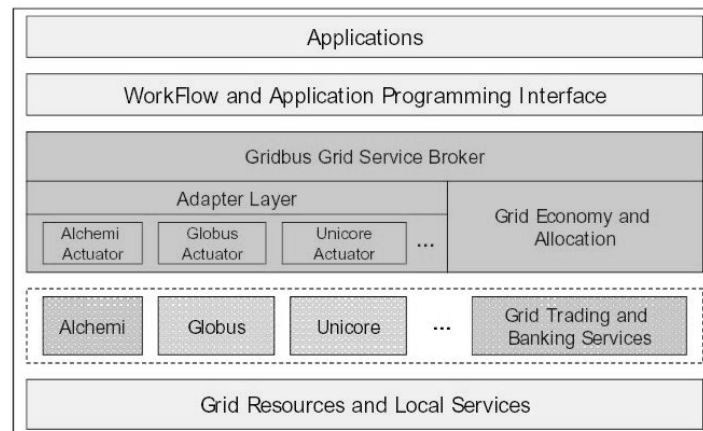


Fig 2.d Gridbus Architecture [1]

<http://172.22.1.127/wiki/index.php/Comparison> discusses in more detail the characteristics. The following table is a checklist representing the features of both the middlewares :-

Middleware Property	Globus	Gridbus
Focus	Low level services	Abstractions and market models
Category	Generic computational	Generic computational
Architecture	Layered and modular toolkit	Layered component and utility model
Implementation Model	Hourglass model at system level	Hourglass model at user level
Implementation Technologies	C and Java	C, Java, C# and Perl
Runtime Platform	Unix	Unix and Windows with .NET (Alchemi)
Programming Environment	Replacement libraries for Unix & C libraries. Special MPI library (MPICH-G), CoG (Commodity Grid) kits in Java, Python, CORBA, Matlab, Java Server Pages, Perl and Web Services	Broker Java API XML-based parameter-sweep language Grid Thread model via Alchemi.
Distribution Model	Open source	Open source

Table 2.a Globus Vs Gridbus [9]

ALCHEMI

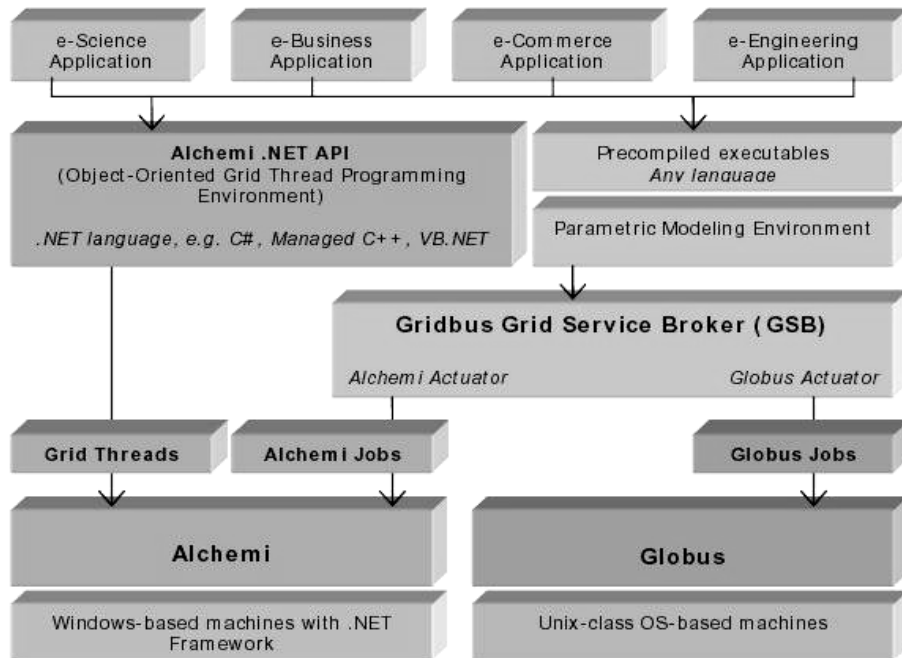
Though scientific computing facilities have been heavy users of Unix-class OSes, the vast majority of computing infrastructure within enterprises is still based on Microsoft Windows. Alchemi was developed to address the need within enterprises for a desktop grid solution that utilizes the unused computational capacity represented by the vast number of PCs and workstation running Windows within an organization.[10] 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.

ALCHEMI Framework

- Course-Grained Abstraction - Grid Jobs: Traditional grid implementations have only offered a high-level abstraction of the "virtual machine", where the smallest unit of parallel execution is a process (typically referred to as a job, with many jobs constituting a task). In this scenario, writing software to run on a grid involves dealing with processes, an approach that can be complicated and inflexible. [14]
- Fine-Grained Abstraction - Grid Threads: Alchemi on the other hand, primarily offers a more low-level (and hence more powerful) abstraction of the underlying grid by providing a programming model that is object-oriented and that imitates traditional multi-threaded programming. The smallest unit of parallel execution in this case is a grid thread (object), where a grid thread is very similar to a "normal" thread. A grid application is defined simply as an application that is to be executed on a grid and that consists of a number of grid threads executing in parallel [14]. The developer deals only with application and thread objects and any other custom objects, allowing him/her to concentrate on the application. Furthermore, abstraction at this level allows the use of programming language constructs such as events between local and remote code. All of this is offered via the Alchemi API.

The fact that a) objects are much easier to deal with than processes combined with b) a programming model that is familiar to developers and c) the use of language constructs across local and remote code makes this a superior approach. The additional benefit of this approach is that it does not limit the developer to applications that are completely parallel. Indeed, it allows development of grid applications where inter-thread communication is required.

The ".NET" in ".NET Grid Computing Framework" & Cross-Platform Support Alchemi is written for the .NET CLR. Hence machines running any Alchemi software component must have the .NET Framework installed. While Alchemi has only been tested on Windows, it is conceivable that it could run on Unix-class operating systems as well. [9]



A layered architecture for integration of distributed Windows and Unix-class resources.

Fig 2.e Integrated Gridbus Architecture [12]

The major Grid tools and application projects making use of Gridbus components within their middleware include: ePhysics Portal, Australian Virtual Observatory, Belle Analysis Data Grid, Global Data Intensive Grid Collaboration, NeuroGrid, Natural Language Engineering on the Grid, HydroGrid, and Amsterdam Private Grid.

3. FABRIC LAYER QoS AND MONITORING

In the fields of packet-switched networks and computer networking, the traffic engineering term Quality of Service (QoS) refers to the probability of the telecommunication network meeting a given traffic contract, or in many cases is used informally to refer the probability of a packet succeeding in passing between two points in the network. In the scenario of Grid Fabric Layer QoS similar issues arise.

3.1 NEED FOR QOS

Initially all networks used to run on a “Best Effort” system with no QoS measures. However, there were many things that could happen to packets as they travelled from origin to destination and that resulted in the following problems [16], [17], [18] :

- **Dropped Packets** - The routers might fail to deliver (drop) some packets if they arrive when their buffers are already full. Some, none, or all of the packets might be dropped, depending on the state of the network, and it is impossible to determine what happened in advance. The receiving application must ask for this information to be retransmitted, possibly causing severe delays in the overall transmission.
- **Delay** - It might take a long time for a packet to reach its destination, because it gets held up in long queues, or takes a less direct route to avoid congestion. Alternatively, it might follow a fast, direct route. Thus delay is very unpredictable.
- **Out-of-Order Delivery** - When a collection of related packets are routed through the Internet, different packets may take different routes, each resulting in a different delay. The result is that the packets arrive in a different order to the one with which they were sent. This problem necessitates special additional protocols responsible for rearranging out-of-order packets once they reach their destination.

- **Error** - sometimes packets are misdirected, or combined together, or corrupted, while en route. The receiver has to detect this and, just as if the packet was dropped, ask the sender to repeat itself.

A traffic contract SLA (Service Level Agreement), specifies guarantees for the ability of a network/protocol to give guaranteed performance/throughput/latency bounds based on mutually agreed measures, usually by prioritising traffic. A defined Quality of Service may be required for certain types of network traffic, for example:

- streaming multimedia may require guaranteed throughput
- IP telephony may require strict limits on jitter and delay
- dedicated link emulation requires both guaranteed throughput and imposes limits on maximum delay
- a safety-critical application, such as remote surgery may require a guaranteed level of availability (hard 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. 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

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.

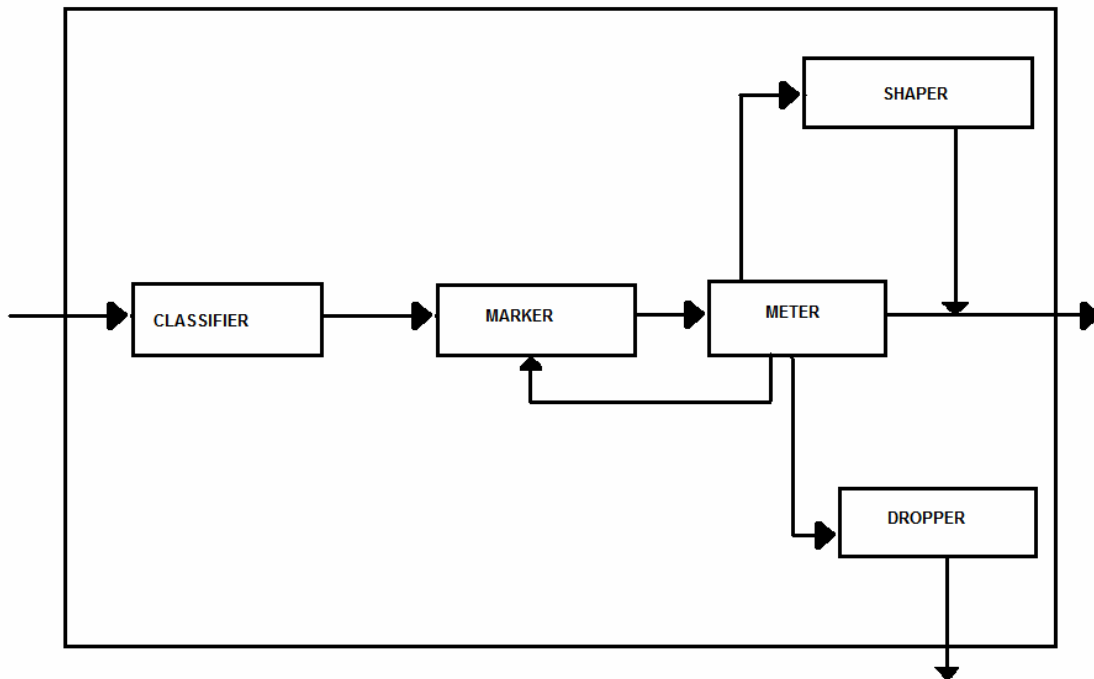


Fig. 3.a. Traffic Control Schema

Traffic Parameters :

The following three parameters are commonly used to bound source traffic [15]:

- *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.

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.

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

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.

- 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

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.

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

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.

3.2 QoS FRAMEWORKS

The QoS term has been primarily used in the networking community to define a set of network performance characteristics. *While speaking of Grids and hence fabric layer, which also provides multimedia services, the concept of QoS not only involves the network but also end-to-end systems.* The following figure shows the abstracted view of the common elements of these architectures to support end-to-end QoS consisting of user, application and system levels. At each of these levels QoS needs to be specified from the level below.

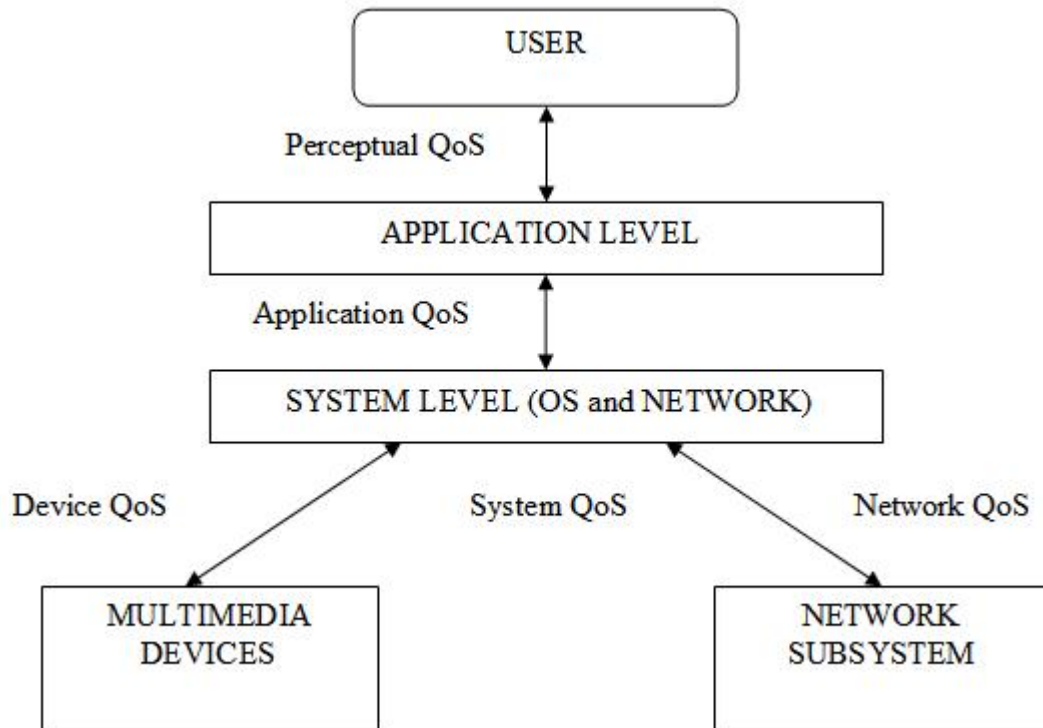


Fig 3.b QoS Levels [15]

The user perception parameter need to be mapped to lower level parameters.

- **Application QoS parameters** include media quality, end-to-end delay requirements, inter/intra stream synchronization and others derived from user's QoS specifications.
- **System level QoS** has two components.
 - Device level QoS specifies timing and throughput requirements.
 - Network QoS parameters are defined in Table 3.a

TIMELINESS	→ Delay Response Time Jitter
BANDWIDTH	→ Systems level data rate Application level data rate Transaction rate
RELIABILITY	→ MTTF(Mean Time to Failure) MTTR(Mean Time to Repair) MTBF(Mean Time between Failures) Percentage of time available Packet loss rate Bit error rate

Table 3.a Network QoS Parameters [15],[17], [18]

There are essentially **two ways** to provide QoS guarantees.

1. The first is simply to provide lots of resources, enough to meet the expected peak demand with a substantial safety margin. This is nice and simple, but it may be expensive in practice, and can't cope if the peak demand increases faster than predicted since deploying the extra resources takes time.
2. The second one is to require people to make reservations, and only accept the reservations if the routers are able to serve them reliably. Thus users can then be charged money for making reservations. There are two popular variations on this:
 - **IntServ**
 - **DiffServ**

3.2.1 Integrated services

IntServ or integrated services is an architecture, which specifies the elements to guarantee quality of service (QoS) on networks. 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. The idea of IntServ is that every router in the system implements IntServ and every application that requires some kind of guarantees has to make an individual reservation. "Flow Specs" describe what the reservation is for, while "RSVP" is the underlying mechanism to signal it across the network.

RSVP

The Resource Reservation Protocol (RSVP) is described in RFC 2205. All machines on the network capable of sending QoS data send a PATH message every 30 seconds, which spreads out through the network. Those who want to listen to them send a corresponding RESV (short for "Reserve") message which then traces the path backwards to the sender. The RESV message contains the flow specs. The routers between the sender and listener have to decide if they can support the reservation being requested, and if they cannot then send a reject message to let the listener know about it. Otherwise, once they accept the reservation they have to carry the traffic. The routers then store the nature of the flow, and also police it. This is all done in soft state, so if nothing is heard for a certain length of time, then the reader will time out and the reservation will be cancelled. This solves the problem if either the sender or the receiver crash or are shut down incorrectly without first canceling the reservation. The individual routers may, at their option, police the traffic to check that it conforms to the flow specs.

Disadvantage: The problem with IntServ is that many states must be stored in each router. As a result, IntServ works on a small-scale, but as it scales up to a system the size of the Internet, it is difficult to keep track of all of the reservations. As a result, IntServ is not very popular.

3.2.2 DIFFERENTIATED SERVICES

DiffServ or differentiated services is a method of trying to guarantee quality of service on large networks such as the Internet. DiffServ deals with bulk flows of data rather than single flows and single reservations. This means that a single negotiation will be made for all of the packets from, for example, a single ISP. The contracts resulting from these negotiations are called "service level agreements", and involve economic concerns. These service level agreements specify what classes of traffic will be provided, what guarantees are needed for each class, and how much data will be sent for each class.

A "DiffServ cloud" is a collection of DiffServ routers. When packets enter a DiffServ cloud they are first classified by the sender. The sender sets the "type of service"(TOS) field (which hence is also called DiffServ Code Point - DSCP), in the IP header according to the class of the data, so that the better classes get higher numbers. 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.

As the packets enter the DiffServ cloud they are policed by the receiver. If there is so much traffic that it breaches the service level agreement, then the sender may be liable for fines, according to the details of the contract. Within the DiffServ cloud, all the individual routers need to do is to give highest priority to the packets with the highest value in the type of service field, which is simple to implement. There may also be a discard policy on the frequencies with which each type of packet is discarded if the router runs out of buffer space.

PHB: In contrast to Intserv, the Diffserv model does not define any service; it defines certain behaviours a packet may receive at each hop. This is called Per Hop Behaviour (PHB). Several traffic flows are aggregated to one of a small number of behaviour aggregates (BA) each of which is treated using the same PHB. Flows identified by the same Diffserv Code Point (DSCP) belong to a BA. RFC 3086 defines Per Domain Behaviour (PDB) and this metric is useful for use in SLAs between domains.

Advantage: All the policing and classifying is done at the boundaries between DiffServ clouds. This means that in the core of the Internet, routers can get on with doing the job of routing, and not care about the complexities of collecting payment or enforcing agreements.

Disadvantage: End-to-end and peering problems: Details of how individual routers deal with the type of service field is somewhat arbitrary and it is difficult to predict end-to-end behaviour. This is complicated further if a packet crosses two or more DiffServ clouds before reaching its destination. Commercially this is a major flaw since it is impossible to sell different classes of end-to-end connectivity to end users.

3.3: QoS PROVISIONS IN IPv4/v6

QoS 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.

QoS in IPv6

An efficient packet processing permits fast forwarding, reducing queuing delays, hence keeping these in mind, an IPv6 packet is specially designed to be efficiently managed by routers. 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.

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

- 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

3.4: SNMP : A MONITORING PROTOCOL

The Simple Network Management Protocol (SNMP) forms part of the internet protocol suite as defined by the Internet Engineering Task Force. The protocol can support monitoring of network-attached devices for any conditions that warrant administrative attention [35]. It's used to query and control network devices from a central management station. The principle behind SNMP is that each network device maintains a database of network statistics that can be queried from a remote device. The three foundation specifications are Structure and Identification of Management Information for TCP/IP-based networks [RFC 1155], Management Information Base (MIB) for Network Management: MIB-II [RFC 1213] and Simple Network Management Protocol [RFC 1157] defining the protocol to manage these objects. Every network device that implements SNMP contains the same core MIB structure. The MIB contains records for simple network statistics such as these:

- The network name of the device
- The number of network interfaces on the device
- The in and out packet counts of each interface
- Error rates of each network interface
- Protocol-specific counts such as TCP and UDP packets

The SNMP protocol operates at the application layer. It specified (in version 1) five core protocol data units (PDUs):

1. GET REQUEST, used to retrieve a piece of management information.

2. GETNEXT REQUEST, used iteratively to retrieve sequences of management information.

3. GET RESPONSE

4. SET, used to make a change to a managed subsystem.

5. TRAP, used to report an alert or other asynchronous event about a managed subsystem.

Other PDUs were added in SNMPv2, include:

1. GETBULK REQUEST, a faster iterator used to retrieve sequences of management information.

2. INFORM, an acknowledged trap.

All managed objects in the SNMP environment are arranged in a hierarchical or tree structure. The leaf nodes are the managed objects, each representing some resource, activity, or related information that is to be managed. Fig below shows the MIB-II object tree structure.

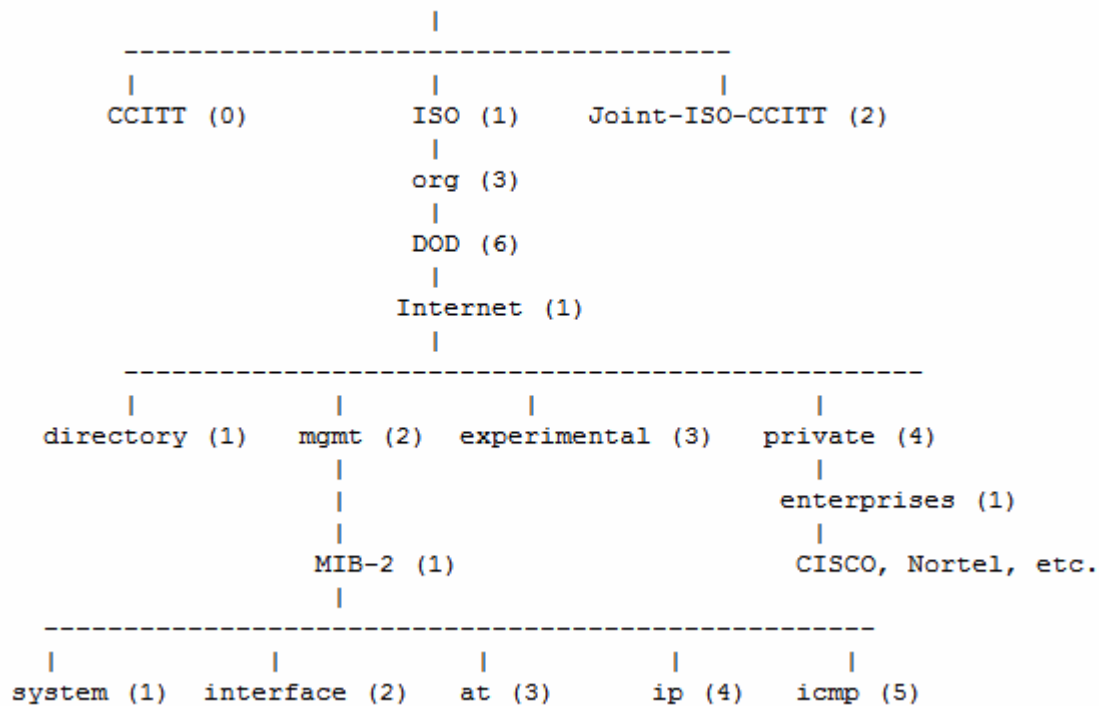


Fig.3.c. MIB-II Objects

The 'private' sub tree has an 'enterprises' child node which is useful in imparting extensibility to SNMP. This portion of the sub tree allows vendors to enhance the management of their devices and share information with other vendors for interoperability. Table below shows the object groups under the MIB-II node.

Object	Node identifier	Description
System	1	System-specific values
Interfaces	2	Network interface statistics
At	3	Address translation table
Ip	4	IP statistics
Icmp	5	ICMP statistics
Tcp	6	TCP statistics
Udp	7	UDP statistics
Egp	8	EGP statistics
Transmission	10	Transmission statistics
Snmp	11	SNMP statistics

Table 3.c: MIB-2 Objects Used for Network Statistics

4: DESIGN

In general, the following characteristics distinguish performance-monitoring information from other forms of system or program-produced data.

- **Performance information has a fixed, often short, lifetime of utility.**
- **Updates are frequent.**
- **Performance information is often stochastic:** 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.

4.1 GOALS

The design of a Grid where the fabric layer gives QoS guarantees, some of which are abstracted as application level QoS, must be done so that it can be easily integrated into the middleware. But there are some pre-requisites that grid components must possess, often referred to as “best practices” . They must :

I. Provide an “inquiry” mechanism whereby it allows for the discovery against its own resource capabilities, structure and state of operations. These are value-added features for resource discovery and monitoring. SNMP software agents are an example.

II. Provide appropriate “resource management” capabilities to control the QoS the grid solution promises or has been contracted to deliver. This enables the service provider 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 [23], represented as the figure below:

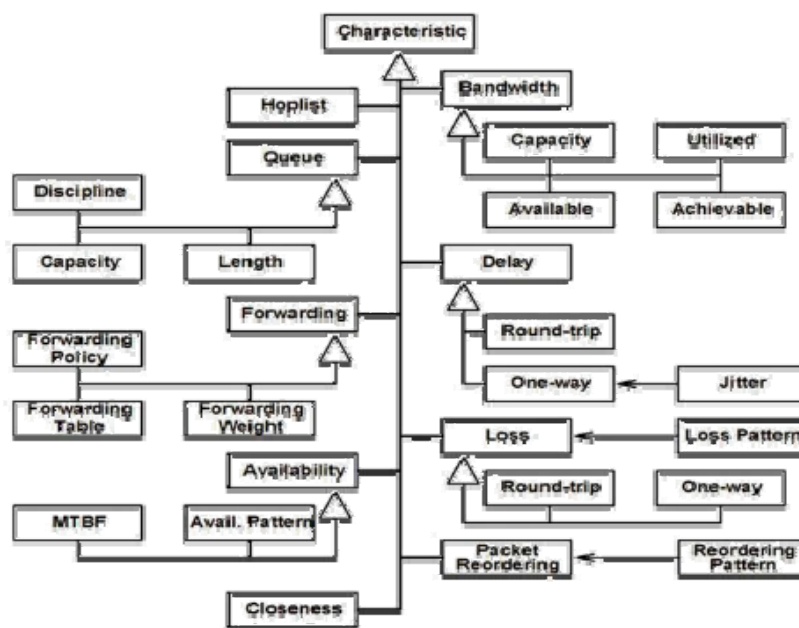


Fig 4.a Network Characteristics for Grid

4.2 CONSTRAINTS

1: We did not have our own custom middleware to integrate the features of Application and fabric level QoS. Alchemi was taken as a reference and implementation of the application module was done on the .NET framework.

2: A Grid environment which is geographically diverse is dependent on the performance of the intermediate ISP routers and their policies. It was assumed that this network which is beyond the Grid's control domain is Diffserv compliant. The next section describes the testbed in detailed manner. In the real world the minimum QoS level will be decided by the intermediate router configurations which may or may not support Diffserv framework.

3: 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 take 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.

4.3: ISSUES

Systems that collect and distribute performance information should satisfy certain requirements:

- **Low latency.** This is because performance data is typically relevant for only a short period of time.
- **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.** Due to potentially thousands of resources, services, and applications to monitor and entities that would like to receive this information, it is important that a performance monitoring system provide scalable measurement, transmission of information, and security.

4.4 DESIGN DESCRIPTION

The final architectural design as it maps to a middleware, which will be Alchemi architecture based, looks like :

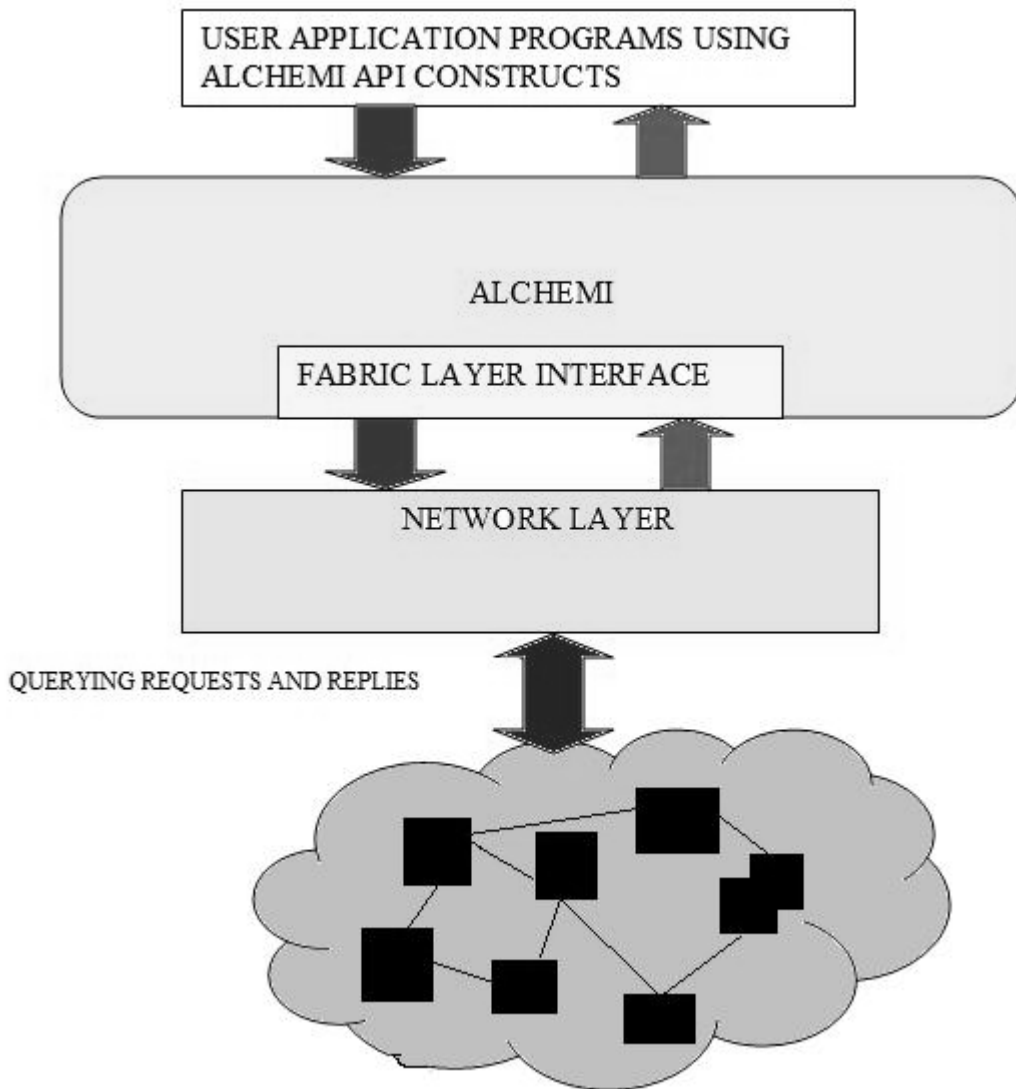


Fig 4.b Grid QoS Design

Workflow at user node

- 1: Node enters the Grid after authentication and registering with the Grid Authorization manager
- 2: The application that seeks QoS informs the middleware some well defined values in QoS metric terms like packet loss, delay, and throughput.
- 3: The middleware component on the node checks to see if the requested QoS can be provided based on the user's credentials and Grid's capabilities. On positive confirmation from the manager, the application is allowed to use the Grid's resources else the application is informed of the lack of requisite QoS guarantee.
- 4: After the job starts executing, the middleware has to keep on checking if the assured QoS is being provided by seeking the fabric QoS metrics from manager at regular intervals.
- 5: On violation, middleware informs manager.

Workflow at Grid Manager

- 1: Receive query from node about desired QoS availability and provisioning.
- 2: Check user credentials, service status and Grid network's present performance status to see if the desired QoS is feasible and reply.
- 3: Signal all intermediate routers with appropriate parameters based on the application's requirements if changes have to be made in traffic control policies for the new traffic that the application will generate.
- 4: If Grid node reports some violation, signal routers with modified parameters.

Workflow at routers

- 1: On receiving values from Grid manager, check if traffic control policy needs to be modified.
- 2: If yes, then implement this change dynamically.
- 3: Police traffic and shape as per the new policy by reading the appropriate packet fields.

5: IMPLEMENTATION, RESULTS AND ANALYSIS

This chapter describes the details of the test-bed used for the experiments and the measurement tools and methods that were utilized. The experiments are also documented along with their results and our analysis. All the experiments discussed were jointly performed by a group of research students including present scholar and his associate Mr. Vikas Agrawal [39].

5.1: TESTBED SETUP

Figures 5.a and 5.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.

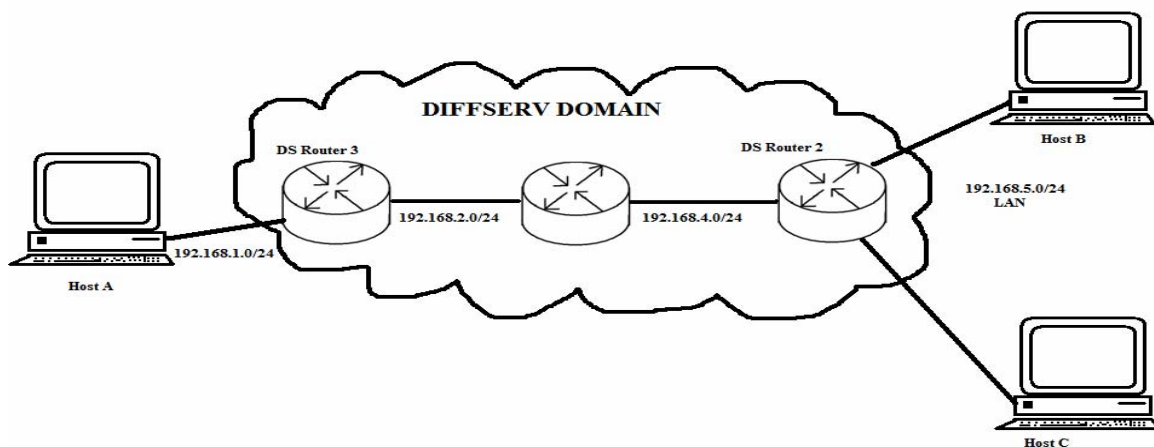


Fig. 5.a. Laboratory Test-bed Diagram

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.

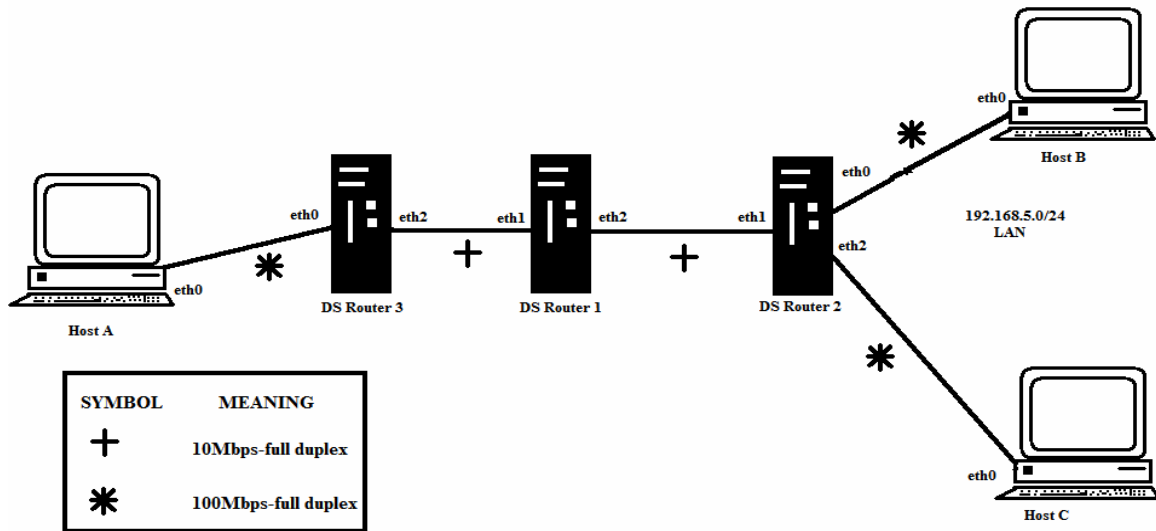


Fig. 5.b. Laboratory Test-bed Diagram with Link information

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.

5.2 : MEASUREMENT STRATEGIES AND TOOLS

METRICS: In our testbed each router is configured to use priority packet scheduling. The network parameters relevant to multimedia traffic that have been considered are one-way delay and packet loss. Packet loss must be less so that information is not lost. For interactive, traffic jitter and one-way delay are relevant metrics otherwise the value of the information is lost since timeliness is an important consideration wherever human-computer interaction comes in the picture.

TOOLS

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

- OpenIMP: The OpenIMP tool [33] which 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 [34] is used to generate traffic flows. Its log files are also helpful in gauging packet loss.
- TC: The tc tool [37] that we have used to implement traffic control also gives packet loss information.
- Ping: ICMP packets were used when setting up the testbed to check the link characteristics of loss and delay.
- SNMP: An application running at host B was used to extract SNMP information.

METHOD

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.

The information extracted from the MIBS through SNMP used to derive error statistics, IP packet loss and packet losses at the interfaces. The test-bed never demonstrated any

sort of transmission errors during the experiments. Fragmentation was also not detected since the packet size that we used was below the link MTUs.

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.

5.2.1: SNMP MONITORING

The SNMP information extraction application was run at the host which we considered as the virtual grid manager. The following values were extracted from the MIBs of the Grid nodes:

System Group:

- sysUpTime: The time since the network management portion of the node was last reinitialized.
- sysServices: A value that indicates the set of services the entity offers. The value is interpreted as a 7 bit code, with each bit corresponding to a layer in the TCP/IP or OSI architecture, starting with the LSB which denotes Layer 1. We got the value 76 for all nodes implying network, transport and application layers.

Interface Group

- Ides: Information about the interface which includes manufacturer name, product name, and hardware interface version.
- ifType: Type of interface distinguished according to physical/link protocol [35].
- ifMtu: The size of the largest PDU in octets that can be received/sent on the interface.
- ifSpeed : An estimate of the interface's current data rate capacity.

- ifPhysicalAddress : Physical address
- ifAdminStatus : Desired interface state (up(1), down(2), testing(3))
- ifOperStatus : Current interface state
- ifInDiscards: Number of inbound packets discarded, even though there were no detected errors. Indicates buffer overflow conditions.
- ifInErrors : Number of inbound error packets that were prevented from being delivered to the higher layer protocols.
- ifOutDiscards : Number of outbound packets discarded, even though there were no detected errors. Indicates buffer overflow conditions.
- ifOutErrors : Number of outbound packets that could not be transmitted because of errors.
- ifOutQlen : Length of the output packet queue.

IP group

- ipForwarding
- ipInReceives
- ipInHdrErrors
- ipInAddrErrors
- ipForwDatagrams
- ipInUnknownProtos
- ipInDiscards
- ipInDelivers
- ipOutRequests
- ipOutDiscards
- ipOutNoRoutes
- ipReasmReqds
- ipReasmOKs
- ipReasmFails
- ipFragOKs

- ipFragFails
- ipFragCreates

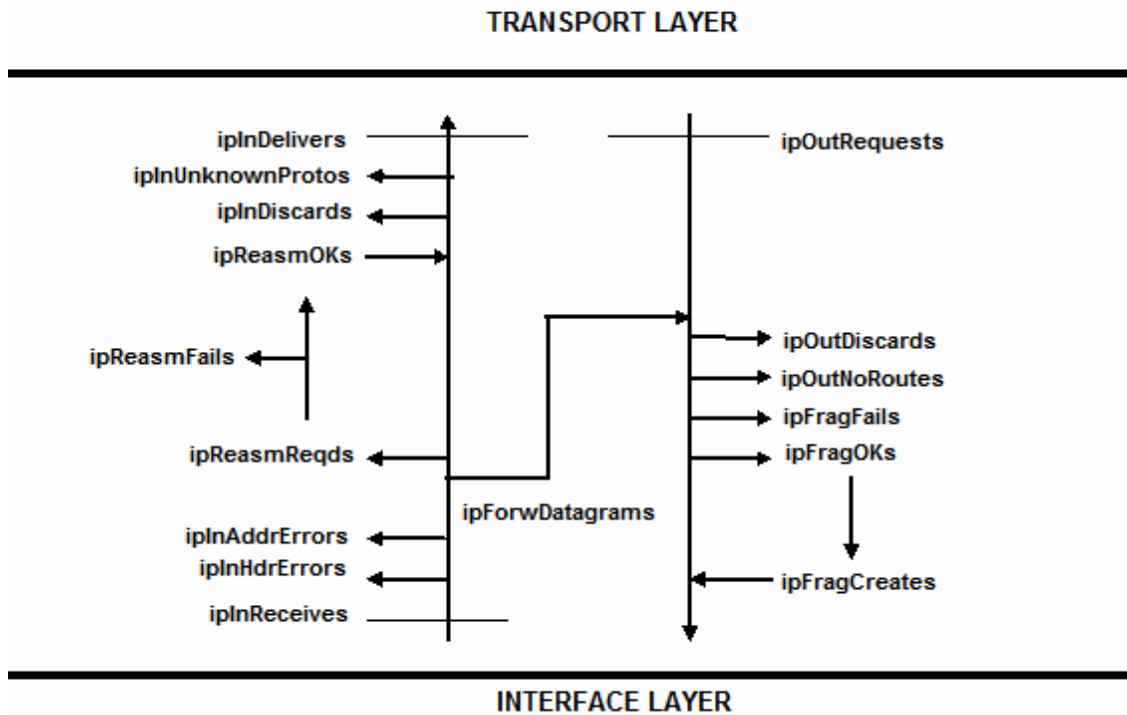


Fig. 5.c. Case diagram for MIB-II ip Group

Case diagrams were proposed by Jeffrey Case who was a part of the IETF chartered MIB working group (1989) to devise methods to make sure all conditions were represented in the MIB groups. Fig. 5 shows the Case diagram for some of the IP group identifiers. We can observe that

$$ipInReceives = ipInHdrErrors + ipInAddrErrors + ipReasmReqds + ipForwDatagrams - ipReasmOKs + ipInDiscards + ipInUnknownProtos + ipInDelivers.$$

a: The terms *ipInHdrErrors*, *ipInAddrErrors* denote header and address errors giving an indication of the number of errors.

b: The *ipReasmReqds* object identifier is a count of the IP fragments received at the entity which had to be reassembled. This is an indication of the proportion of traffic that has to undergo fragmentation which is an overhead and adds to the delay.

c: The *ipReasmFails* value is an indicator of the reliability of the network since the IP packets that failed reassembly were probably tampered before reaching the entity.

5.2.2: TIME SYNCHRONIZATION

NTP is a protocol designed to synchronize the clocks of a computers over a network. NTP version 3 is an internet draft standard, formalized in RFC 1305.

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:

```
ntpdate -u 192.168.5.3
```

To constantly update each two seconds:

```
watch ntpdate -u 192.168.5.3
```

5.2.3: 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. 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. These script files can be used to emulate the traffic patterns of unicast and/or multicast applications. The receive portion of this tool set can be scripted to dynamically join and leave IP multicast groups. 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.

The principal tool is the **mgen** program which can generate, receive, and log test traffic.

Syntax:

```
Mgen[ipv4][ipv6][input<scriptFile>][save<saveFile>][outputlogFile>][log<logFile>][binary][txlog][nolog][flush][hostAddr{on|off}][event"<mgenevent>"][portrecvPortList>][instance<name>][command<cmdInput>][sink<sinkFile>][block][source<sourceFile>][interface<interfaceName>][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, VOIP, 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 random 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 scripts 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 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.

5.3: EXPERIMENTAL ANALYSIS

5.3.1: TEST 1 – NORMAL TRAFFIC

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 5.a:

Stream	Type	Rate(packet/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 5.a Test 1 Traffic profile

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

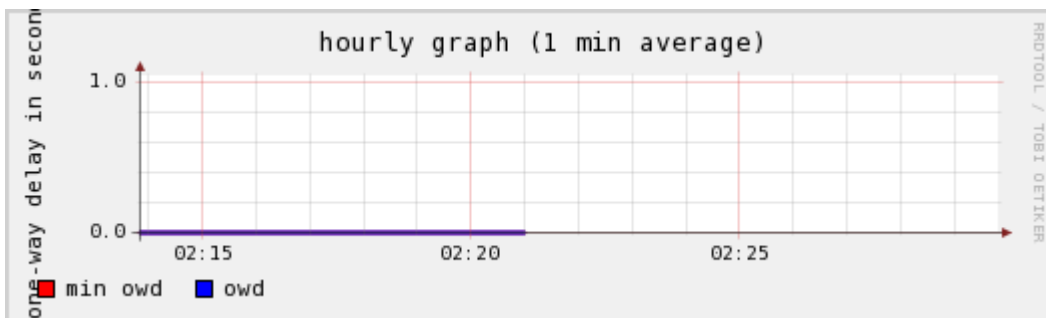


Fig. 5. d Average One-way delay for Test 1

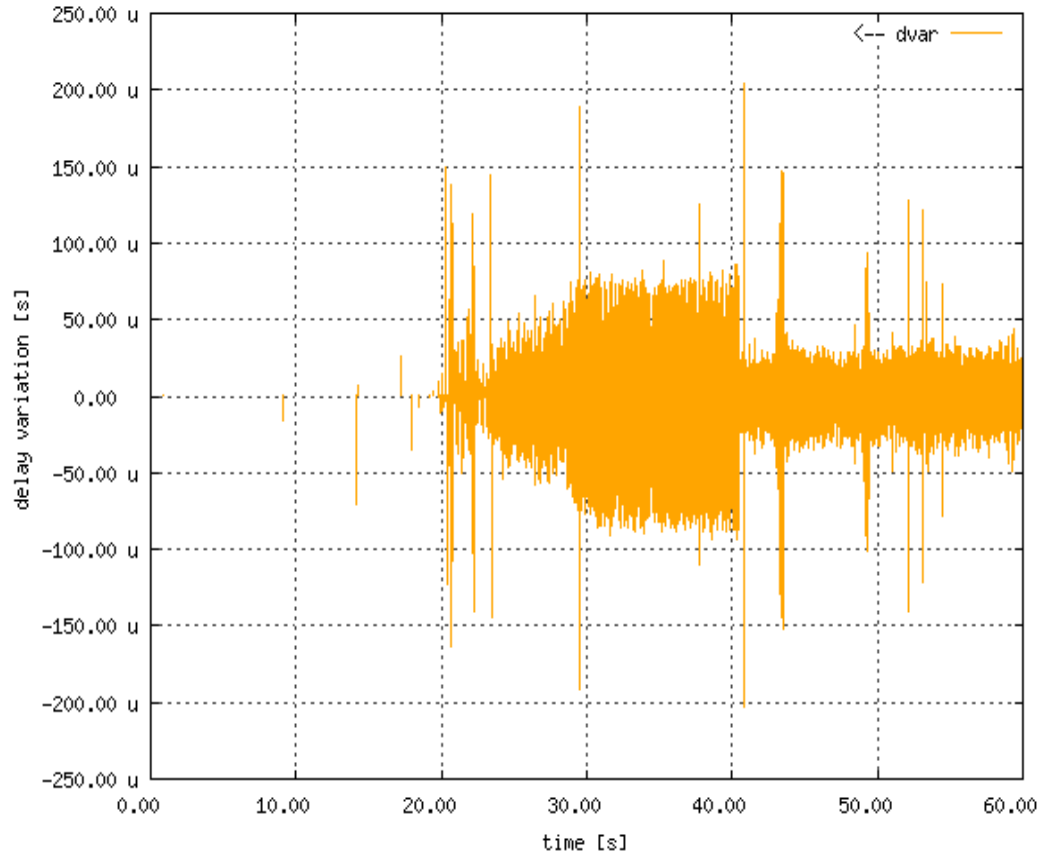


Fig. 5. e Delay Variation (Jitter) of traffic in an interval of 1 min

We observed from Fig 5.d 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. 5. e shows the effect of the traffic profile of the streams. As seen in table 5.a we have used 3 Poisson flows which send packets at statistically varying intervals.

5.3.2: TEST 2 – BURSTY MULTIMEDIA TRAFFIC

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 5.b is the traffic profile while the multimedia traffic's characteristics are described in the MGEN script for Experiment 2 in Appendix C.

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

Table 5.b: Profile of Traffic for Experiment 2

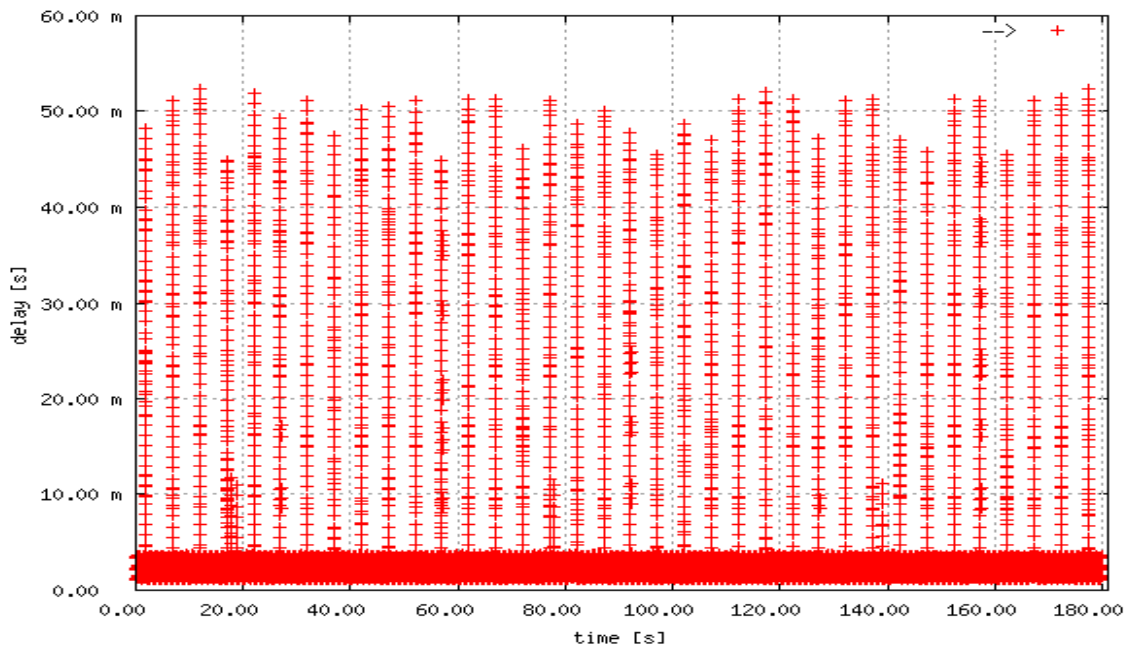


Fig. 5. f One-Way Delay over a period of 3 minutes for multimedia and a normal traffic

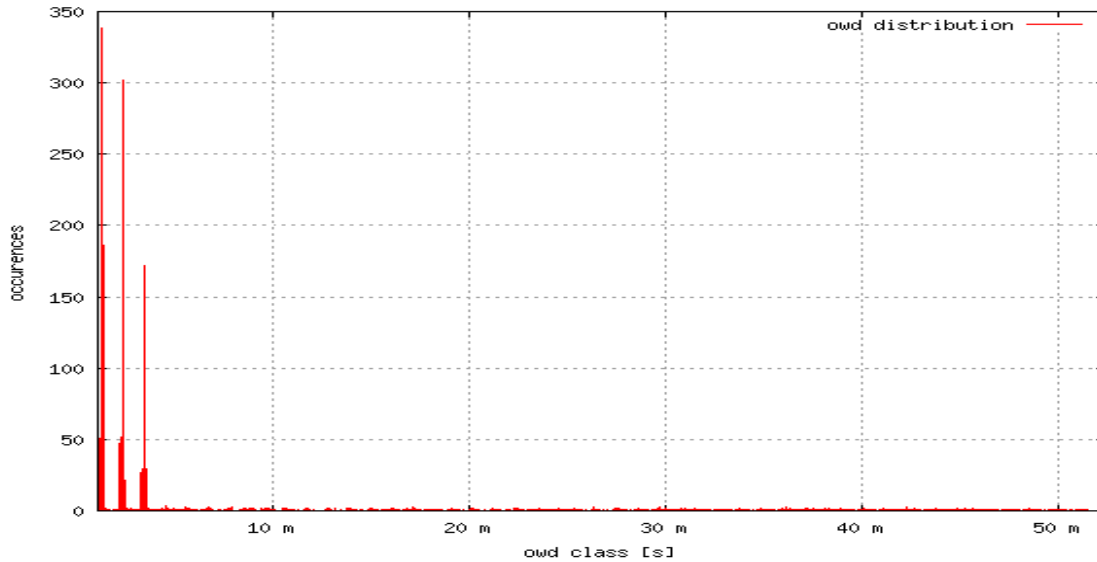


Fig. 5. g One-Way Delay Distribution for multimedia and a normal traffic

Fig 5.f 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 5.g 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 2nd flow in this experiment, none of its packets would have suffered increased delay when the multimedia flow demonstrated bursts.

5.3.3: TEST 3 – STATIC QoS

We implemented static traffic control in this case, with the following parameters for the traffic flows. 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 5.c: Profile of Traffic for Experiment 3

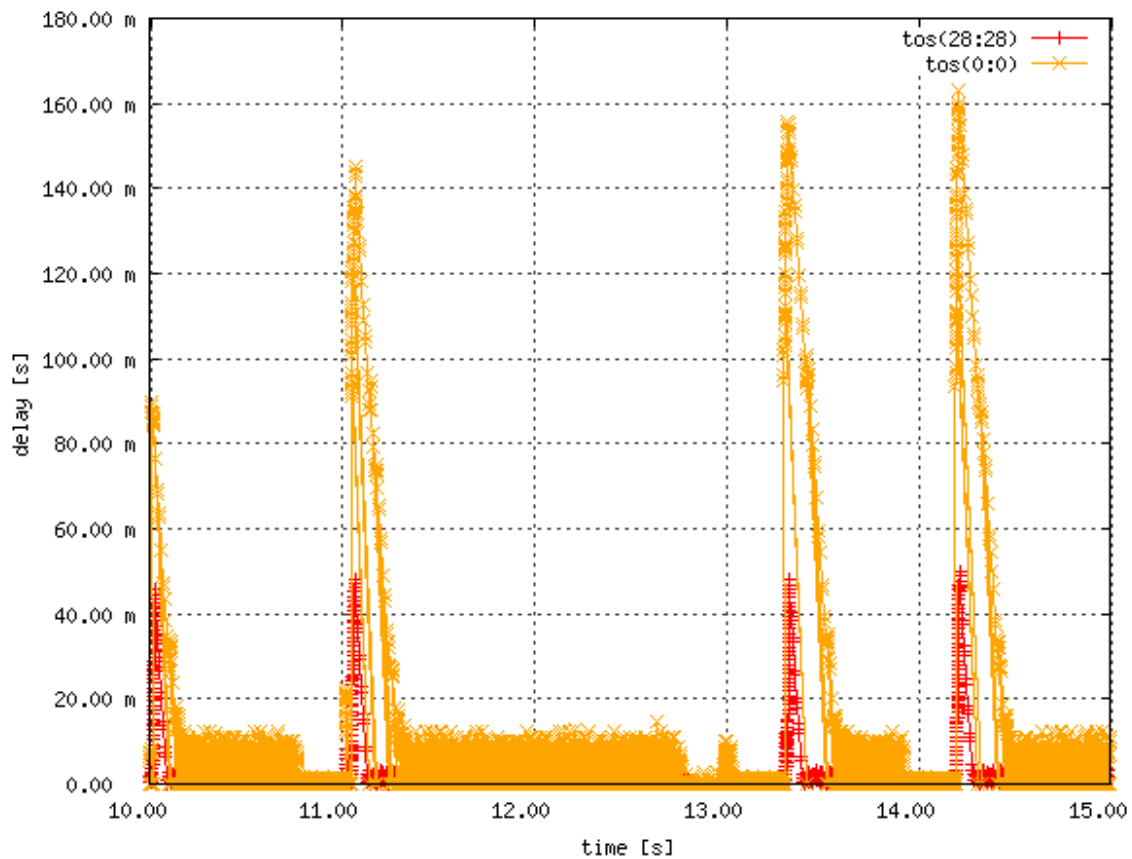


Fig.5.h One Way Delay for two different traffic streams after being shaped

Fig 5.h shows the one-way delay graph. The TOS field the packets was used to filter the flows and assign them to different classes. The orange stream is the bursty traffic while the red stream is a non-bursty 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.

5.3.4: TEST 4 – DYNAMIC QoS

The traffic profile is same as that of Test 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 heterogenous 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 testbed scenario, sends a file to the routers with the new parameters that is utilized by a script running at the routers to allocate more bandwidth capacity to the orange traffic.

Thus we can see that its burst delays decrease and packet losses too decrease as a result of this. The scripts that were used are given in Appendix B.

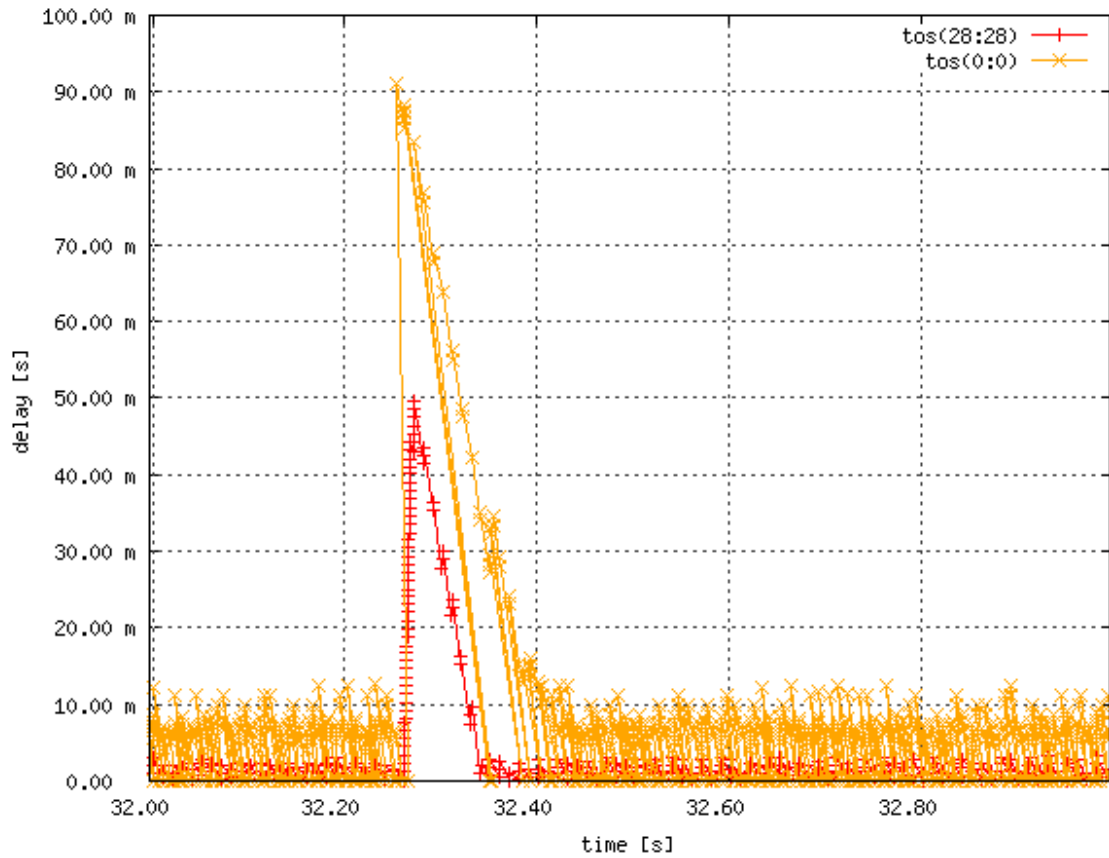


Fig. 5.i Traffic of Experiment 3 after being dynamically shaped

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.

6: 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. Dynamic reconfiguration provides a promising alternative approach to configuration management of complex and evolving Grids. 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 persistent negotiated QoS for grid applications were carried out. A Grid with a Diffserv supporting intermediate internet was simulated partially in the testbed through 3 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 cost effective way to increase the reliability and throughput of multimedia and other prioritized transactions. The encouraging experimental results suggest that its performance compares quite favourably 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
```

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 "$DEVFIELDS" | sed 's/./#/g' | sort -u`
[ `echo "$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=tbf; 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=${DEVTEMP%%,*}; DEVWGHT=${DEVTEMP##*,}
    [ "$DEVBWDT" = "$DEVWGHT" ] && DEVWGHT=""

    ### 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

    ### 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 [ "$SRC1" ]; then
#echo "src 1 found..."
#SADDR=${SRC%:*}; STEMP=${SRC##*:}
#[ "$SADDR" = "$SRC" ] && STEMP=""
#SPORT=${STEMP%/*}; SMASK=${STEMP##*/}
#[ "$SPORT" = "$STEMP" ] && SMASK="0xffff"

echo "Error with DSCP value...Not Present"
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} $u32_d"
SRC1="${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
    TIMESET=`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" = "tb" ]; then
    tc qdisc replace dev $DEVICE handle $CLASS tb \
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
    $0 start "$@"
    ;;

```

```
*)  
    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 B

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 C

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. Joshy Joseph, Craig Fellenstein, Grid Computing, IBM Press 2004
2. Project GridOne homepage, <http://discovery.bits-pilani.ac.in/gridone/index.htm>
3. Rahul Banerjee, GridOne : An IPv6 QoS Aware Grid Computing Architecture, Euro India 2004 Summit, New Delhi
4. Position Paper 1.0 , Project GridOne, <http://discovery.bits-pilani.ac.in/gridone/index.htm>
5. Xiaoshan He, Xian-He Sun, Gregor Von Laszewski, "A QoS Guided Scheduling Algorithm for Grid Computing", Proceedings of the International Workshop on Grid and Cooperative Computing, Sanya, China, Dec, 2002
6. Gridbus Project Homepage , www.gridbus.org
7. Globus Homepage, www.globus.org
8. Foster, Kesselmann, Nick, Tuecke, The Physiology of the Grid , Version 6//22/2002
9. P Asadzadeh, R Buyya, Chun Ling Kei, D Nayyar, S Venugopal, Global Grids and Software Toolkits : A Study of Four Grid Middleware Technologies
10. The Alchemi Open Source Grid Project, www.alchemi.net
11. A.Luther, R Buyya, R Ranjan, S Venugopal, Alchemi : A .NET based Enterprise Grid Computing System
12. A.Luther, R Buyya, R Ranjan, S Venugopal, Peer-to-Peer Grid Computing and a .NET based Alchemi framework
13. A Finkelstein, C Gryce, J Lewis-Bowen, Relating Requirements and Architecture: A Study of Data Grids, Journal of Grid Computing, Springer 2005
14. A Luther, K Nadiminti, R Buyya, User Guide for Alchemi 1.0, July 2005
15. Sanjay Jha, Mahbub Hassan, Engineering Internet QoS, Artech House Inc. 2002
16. Wikipedia-The Free Encyclopedia, http://en.wikipedia.org/wiki/Quality_of_Service
17. Lowekamp, Swany et al. , A Hierarchy of Network Performance Characteristics for Grid Applications and Services, GFD-R-P.023 (Proposed Recommendation), May 2004
18. Volker Sander , Networking Issues for Grid Infrastructure, GFD-I.037, Grid High Performance Networking Research Group, November 2004
19. Global Grid Forum, www.ggf.org
20. Chamil P. W. Kulatunga, Paul Malone, Mícheál Ó Foghlú, Adaptive Measurement Based QoS Management in DiffServ Networks, IST-Intermon Workshop 2004
21. T. Ahmed, R. Boutaba and A. Mehaoua, A Measurement-Based Approach for Dynamic QoS Adaptation in DiffServ Network, Journal of Computer Communications, Special

- issue on End-to-End Quality of Service Differentiation, Elsevier Science, Vol 28/18, pp 2020-2033, 2004
22. Mark Baker, Hong Ong, Garry Smith, A Prototype Grid-site Monitoring System, Version 1, Distributed Systems Group Technical Report, January 2002
 23. B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski, A Grid Monitoring Architecture, GFD-I.07,GGF Performance Working Group, January 2002
 24. R. Aydt, D. Gunter, W. Smith, B. Tierney, M. Swany, V. Taylor, A Simple Case Study of a Grid Performance System, GFD-I.08, GGF Performance Working Group, May 2002
 25. Mark Baker, Garry Smith, GridRM: A Resource Monitoring Architecture for the Grid, Grid Computing Workshop - Grid 2002, 3rd International Workshop, Baltimore, MD, USA, November 18, 2002, Proceedings, Lecture Notes in Computer Science 2536 Springer 2002, ISBN 3-540-00133-6
 26. Mark Baker, Garry Smith, GridRM: An Extensible Resource Monitoring System, IEEE International Conference on Cluster Computing (Cluster 2003), December 2003, Hong Kong, China
 27. Mathew Grove, M Baker, jGMA: A lightweight implementation of the Grid Monitoring Architecture, UKUUG LISA/Winter Conference, February 2004
 28. Rich Baker, Dantong Yu, Jason Smith, and Anthony Chan, GridMonitor: Integration of Large Scale Facility Fabric Monitoring with Meta Data Service in Grid Environment, Computing in High Energy and Nuclear Physics, La Jolla, California, March 2003
 29. Kun Yang, Xin Guo, Alex Galis, Bo Yang, Dayou Liu, Network Engineering towards Efficient Resource on-Demand in Grid Computing, Proceedings of ICCT 2003
 30. Daniel A. Menasce, Emiliano Casalicchio, Quality of Service Aspects and Metrics in Grid Computing, Proceedings of the Computer Measurement Group Conference, Las Vegas, December, 2004
 31. Sergio Androozzi, A. Ciuffolletti, A. Ghiselli, Monitoring the Connectivity of a Grid, 2nd ACM Workshop on Middleware for Grid Computing, Toronto, Canada, 2004
 32. Craig A. Lee, James Stepanek, Rich Wolski, Carl Kesselman, Ian Foster, A Network Performance Tool for Grid Environments, Proceedings of SC99, November, 1999
 33. OpenIMP, www.ip-measurement.org/openimp/
 34. MGEN The Multi-Generator Toolset, <http://pf.itd.nrl.navy.mil/mgen/>
 35. William Stallings, SNMP, SNMPv2, SNMPv3 and RMON 1 and 2, Third Edition, Pearson Education, 2003

36. F. Yin, “IPv6 DiffServ Study and Test Report”, STC Internal Report, Jan. 2003
37. B. Hubert et al., Linux Advanced Routing & Traffic Control HOWTO home page; <http://www.lartc.org/>
38. Piyush Gupta, “Investigations into Design and Implementation of an IPv6-QoS-Aware Grid”, Master’s Thesis, Chalmers University of Technology, November 2005
39. Vikas Agrawal, “Establishment of QoS Enabled Multimedia Collaboration Grid over Native IPv6 Fabric” Dissertation Report, 53p, BITS Pilani, India, December 2005