NETWORK QoS PROVISION FOR DISTRIBUTED GRID APPLICATIONS

RASHID AL-ALI,[1] SHALEEZA SOHAIL,[2] OMER RANA,[1]
ABDELHAKIM HAFID,[3] GREGOR VON LASZEWSKI,[4]
KAIZAR AMIN,[4] SANJAY JHA[2] AND DAVID WALKER.[1]

[1] *Cardiff University, Cardiff, UK.*
[2] *University of New South Wales, Sydney, Australia.*
[3] *Université de Montréal, Montréal (Québec) Canada.*
[4] *Argonne National Laboratory, Argonne IL, USA.*

Abstract: Grid computing provides a global-scale distributed computing infrastructure for executing scientific and business applications. An important requirement is the need to make this infrastructure appear as a single logical co-ordinated resource – which in reality consists of a variety of resources aggregated across different administrative domains. The aggregation of network resources is often undertaken over a 'best effort' infrastructure as provided by the Internet – however many applications, which necessitate soft-real time constraints, such as collaborative working or remote visualisation, require more stringent traffic guarantees. Managing Quality of Service (QoS) requirements across these aggregated resources therefore becomes an important concern. Such QoS criteria must extend to computational, data and network resources, and are often expressed in a Service Level Agreement, multiples of which may co-exist over the entire collection of resources. In this paper we focus on network QoS as part of our wider work on the Grid QoS Management (G-QoSm) framework. Provisioning of such QoS support is provided via the Differentiated Services architecture, and relies on the use of a Bandwidth Broker (BB) component. Performance results of using the BB alongside other elements in G-QoSm are presented.

Keywords: Grid Computing, Quality of Service (QoS), Differentiated Services (DiffServ), Resource Management.

1. INTRODUCTION

As additional application users begin to make use of Grid infrastructure, there has been concern that current resources can only be used in a 'best effort' mode. This means that once a collection of suitable resources have been identified, subsequent use of these resources is not constrained in any way. The resource provider often does not give any guarantees on time taken to complete a particular job mapped to the resource. This can be of concern for applications which have soft real-time constraints, and it is necessary to consider QoS attributes that need to be associated with a service advertised by a resource owner. Such attributes may be used by an application user to gauge likely response times from a resource, thereby helping to predict the execution time of an application.

A key concern in Grid infrastructure is that resources are not owned by a single administrator, and may need to be aggregated across multiple domains. There is therefore a need to specify a contract or Service Level Agreement (SLA) that enables a collection of resources to be combined to execute a single problem. Such an SLA must contain descriptions of network, storage and computational resources, and forms the basis of our G-QoSm framework. We explore how network QoS parameters can be measured, and specified, within such a SLA.

In *Section 2 ~ Related Work* we discuss previous work related to QoS management in Grid computing, focusing in particular on GARA and NRS systems [Foster *et al.* 1999; Bhatti *et al.* 2003]. *Section 3 ~ Grid QoS Management Background* discusses the basis of G-QoSm and its architecture, and a QoS-enabled Grid service is defined, i.e. a service with QoS attributes defined as part of its interface. *Section 4 ~ Network Quality of Service* discusses key elements in managing network QoS, such as DiffServ and the Bandwidth Broker, and *Section 5 ~ BB_{Basic} Integration* specifies how an implementation of the Bandwidth Broker is used alongside the Java Commodity Grid (CoG) kit in G-QoSm, and illustrates using a code example how a developer can deploy the QoS-enabled Grid service. *Experimental Results* are presented in *Section 6*.

2. RELATED WORK

The concept of Quality-of-Service (QoS) was first used in the network community [Aurrecoechea *et al.*

1995], with network QoS dealing specifically with providing certain quality levels for network link characteristics between two points, expressed in terms of delay, jitter, packet loss rate and throughput (bandwidth).

To manage these network parameters, certain network elements – network routers or network traffic-control entities, such as Linux-based routers – are modified to support QoS-aware mechanisms, such as the Differentiated Services architecture [Blake *et al.* 1998; Xiao and Ni, 1999], or changes are made to the application to manage packet transmission from/to it, based on feedback from the receiver. The first of these – modifying network elements – is usually undertaken at the network level and requires the configuration of QoS-aware network elements. The alternative approach is an application-level solution, where feedback on network performance is used to control the rate at which data is transmitted from the sender. This end-point feedback based approach has limited use and is often only appropriate in delivering multimedia applications between two points on the network. The feedback from the destination leads to reducing the amount of data transmitted by the sender – so that a lower quality data stream is sent, compared to no stream at all, to the destination.

With the emerging interest in Grid computing, several research groups have been actively involved in bringing concepts and mechanisms from the network community to support network QoS for Grid applications. Most of these activities are based on the Differentiated Services architecture (DiffServ) [Blake *et al.* 1998], introduced by the Internet Engineering Task Force (IETF). Two substantial recent efforts in the Grid community aim at addressing the issue of introducing DiffServ-based QoS provision for use in Grid applications, namely: i) the General-purpose Architecture for Reservation and Allocation (GARA) [Foster *et al.* 1999], and ii) the Network Resource Scheduler (NRS) project [Bhatti *et al.* 2003].

The General-purpose Architecture for Reservation and Allocation (GARA) is the best known framework for supporting QoS in computational Grids, and provides the capability for specifying end-to-end QoS requirements; its advance reservation service treats various types of resources uniformly, such as network, computation and storage, and provides a guarantee that an application initiating a reservation will receive a specific QoS from the Resource Manager. GARA also provides an application-programming interface to *create*, *modify*, *bind* and *cancel* reservation requests. Network QoS in GARA is designed and built to work with a specific network router, the Cisco 7507, and uses Cisco's Modular QoS Command-line interface (MQC) to configure routers, i.e. a Policy Enforcement Point (PEP), to support DiffServ capability. In a multi-domain network, the GARA system must exist in every administrative domain. In making a network reservation for traffic spanning multiple administrative domains, two issues need to be resolved: 1) locating/contacting the GARA system in each domain along the traffic path, and 2) ensuring that the application/user requesting the reservation has secure access to each GARA system along the path. This introduces manageability limitations and constraints on the types of domains in which GARA can be deployed.

The Network Resource Scheduler (NRS) on the other hand adopts the Peer-to-Peer (P2P) model, as a NRS exists in every administrative domain and it is assumed there is a trust relationship between neighbouring NRSs. The NRS uses the DiffServ concept, and therefore every neighbouring NRS has a DiffServ Service Level Agreement (SLA). The application/user requesting a network QoS need only negotiate with the local NRS and establish a local SLA. During the negotiation process the local NRS replicates the request to all NRSs along the network path to conduct an admission control check and, subsequently, establish a SLA. NRS, somewhat like GARA, is designed and built to only work with Cisco routers and makes use of Cisco-ISO to configure Cisco's routers (PEP) to support DiffServ capability. Although NRS has demonstrated its effectiveness in providing DiffServ QoS, it is not clear how a Grid application developer would make use of this capability especially as the application programming interface is not clearly defined. The use of a NRS also requires the definition of specific network parameters such as Traffic Specification (TSpec), token bucket size and token bucket rate – which require advance networking knowledge.

3. GRID QoS MANAGEMENT BACKGROUND

Grid Quality of Service Management (G-QoSm) is a framework to support QoS management in computational Grids in the context of the Open Grid Service Architecture (OGSA) [Al-Ali *et al.* 2002; Al-Ali *et al.* 2003]. G-QoSm is a generic modular system that, conceptually, supports various types of resource QoS, such as computation, network and disk storage.

G-QoSm has three main operational phases: *establishment*, *activity* and *termination* [Hafid and Bochmann, 1998]. During the establishment phase, a client application states the desired service and QoS requirements. G-QoSm then undertakes a service discovery, based on the specified QoS properties, and negotiates an agreement for the client application. During the activity phase additional operations, such as QoS monitoring, adaptation, accounting, and possibly re-negotiation may take place. During the termination phase the QoS session is ended due to an expiring resource reservation, an agreement being violated or a service completion; resources are then freed for use by other clients. The framework supports these three phases using a number of specialist components, as depicted in Figure 1. In subsequent sections we describe these interactions, and highlight how service provision is undertaken.

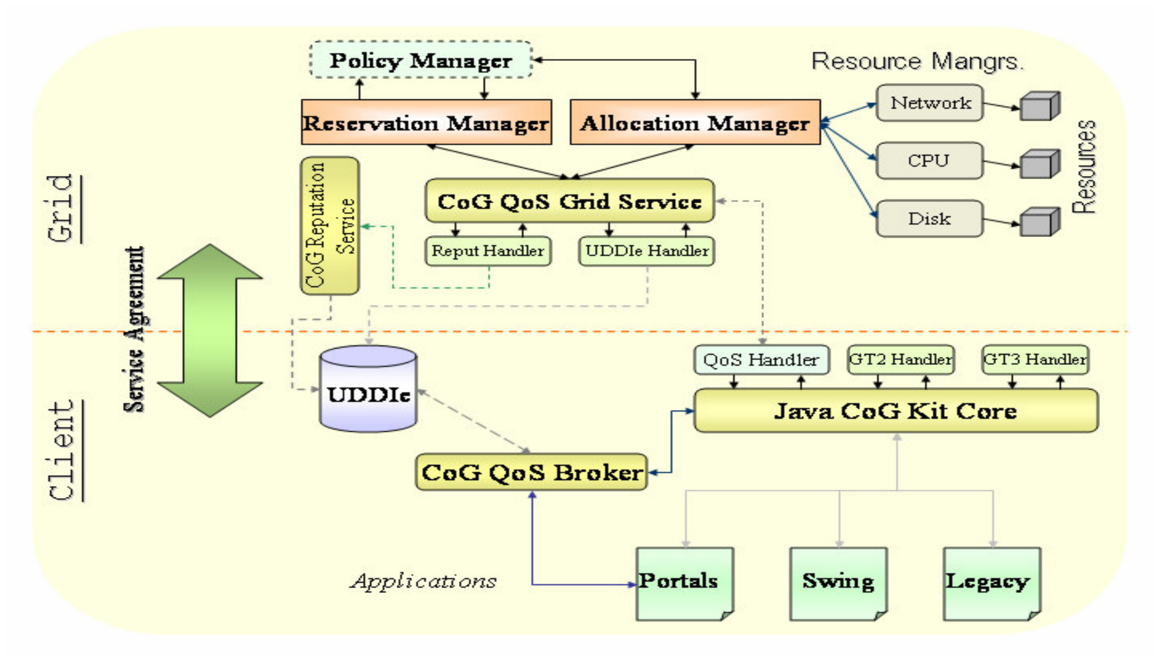

Figure 1: G-QoSm Architecture

3.1. Grid QoS-enabled Service

The basic building block of our architecture is the Grid QoS-enabled Service (GQS), an OGSA-compliant Grid service providing QoS functionality, including negotiation, reservation and resource allocation with specific quality levels. Each QoS-enabled resource is accessed through a GQS, which is published in a QoS registry service. This enables both clients and QoS brokers to discover the GQS.

The GQS interacts with the QoS Handler, Reservation Manager, Allocation Manager, and the QoS Registry Service (UDDIe) modules, as in Figure 1, to deliver QoS guarantees. The architecture, as illustrated in Figure 1, consists of a client – the lower part of the figure, and a service provider – the upper part of the figure. The client makes use of a registry service (UDDIe) which can be used alongside the Java CoG kit. A client may be a physical user accessing the registry service via a portal, or may be another service issuing a search request. If the client is another service, access to the registry is via a QoS Broker. A service provider, on the other hand, illustrated in the top right portion of Figure 1 must provide access to physical resources that are used to manage the service – including support for computation, data storage and network access. The first interaction between a client and a service provider is therefore via the discovery operation invoked on the registry service. The UDDIe Handler enables a service provider to publish properties of a service within the registry, and to subsequently alter any parameters associated with the service.

Once a request for a service has been received, either the Reservation or the Allocation Manager is invoked. To support QoS characteristics a service provider must ensure that in addition to the service being offered to external users, it also supports additional components to allow reservation and subsequent allocation of resources. In addition, the service must be annotated with additional properties that enable these QoS attributes to be encoded in its interface.

The GQS undertakes i) resource reservation, and ii) resource allocation. When a reservation request is received, the GQS undertakes an admission control to check the feasibility of granting such a request. This feasibility check is undertaken via the Reservation Manager, and if possible the requested resources are reserved, the reservation table is updated and an agreement on the reservation specification is generated and returned to the client.

One the other hand, when a resource allocation request is received, in the case of compute QoS, the GQS verifies that the user has indeed made a reservation based on the supplied Service Level Agreement. If this test passes, then the GQS submits the specification of the job to be executed to the Globus Resource Allocation Manager (GRAM) for that particular resource. Along with the job specification, the GQS supplies other parameters related to allocating a particular fraction of a compute resource. These parameters are passed from GRAM to the local compute resource manager for immediate allocation. This process is handled by the Allocation Manager provided in the GQS.

The GQS contains the following modules:

Reservation Manager: The Reservation Manager uses a data structure that supports reservations for quantifiable resources – i.e. resources associated with defined capacities, such as 10% of CPU time, or 25% of main memory. The Reservation Manager is de-coupled from the underlying resources and does not have direct interaction with them. However, it obtains resource characteristics and policies governing resource usage from the Policy Manager. The Policy Manager, in turn, is responsible for validating reservation requests by applying domain-specific rules established by the resource owners, regarding when, how, and by whom the resource can be used. In brief, when the Reservation Manager receives a reservation request from the GQS, it contacts the Policy Manager for validation of this request, and then performs an admission control to check availability of the requested resource. Upon success, it returns a positive reply to the GQS – which leads to the generation of a service agreement offer to the requesting client.

Allocation Manager: The Allocation Manager has a primary role to interact with underlying resource managers, for resource allocation and de-allocation, and to inquire about the status of the resources. It has interfaces with resource managers for each type of resource supported, namely the Dynamic Soft Real Time Scheduler (DSRT) [Chu and Nahrstedt, 1999] for computational resources, and the Bandwidth Broker (BB_{Basic}) [Sohail *et al.* 2003] for network resources. When the Allocation Manager receives a resource allocation request from the GQS, it forwards the request to the designated underlying Resource Manager for further processing. At present we do not support a resource manager for storage allocation.

QoS Registry Service: Service publishing, in this context, does not simply mean publishing a service name, URL and basic description. For example, for GQS it includes a list of QoS-enabled services it offers, allocation strategies it employs in the case of compute QoS provision, and classes of network QoS it offers; for example 'best effort', 'controlled load' or 'guaranteed'. For other Grid services, service publishing includes information on QoS properties such as performance characteristics and service execution requirements. We make use of an extended version of the Universal Description Discovery and Integration (UDDI) registry to achieve this. UDDIe [ShaikAli *et al.* 2003] is a Web services registry providing service providers and users a means to publish and search for services with QoS properties.

4. NETWORK QUALITY OF SERVICE

Currently the Internet treats all traffic equally as 'best effort' and provides no support for any other Quality of Service (QoS). The Internet Engineering Task Force (IETF) has proposed the Integrated Services (IntServ) and the Differentiated Services architectures (DiffServ) [Barden *et al.* 1994; Blake *et al.* 1998]. Both architectures support QoS and provide a type of guarantee, in terms of bandwidth, latency and other data transfer parameters, to deliver network traffic between a source and destination.

IntServ enables the user to reserve resources by maintaining per flow admission control, signalling, classification and scheduling at every router on the path from source to destination. Hence, IntServ can provide 'per flow' guarantees to users. Scalability is the main issue negating the deployment of the IntServ architecture on the Internet. Provision of 'maintenance of state' information, for huge number of flows passing through the Internet core routers, needs enormous resources. This approach is therefore virtually impossible to deploy in reality.

DiffServ, however, provides a broad, and flexible, range of services avoiding a 'per flow' state in core routers. The main goal of the DiffServ architecture is to provide a preferred level of service to particular types of network traffic without increasing overhead in the core routers. Essentially, DiffServ provides an aggregated end-to-end service over a number of separately administered domains. At inter-domain level there has to be a mechanism to exchange critical information among domains about aggregated flows. A Bandwidth Broker has been introduced [Teitelbaum *et al.* 1999] as a resource management entity that provides the necessary functionality for allocation of intra-domain resources, and arranging inter-domain agreements.

A Bandwidth Broker (BB) is a logical entity responsible for managing QoS resources in an administrative domain, based on a Service Level Agreement (SLA). A SLA is the contract between two domains, or between a domain and a client, which specifies, to the forwarding service, the amount of traffic the client can receive. Organizational policies can be configured by using the mechanism provided by a BB. On the inter-domain level the BB is responsible for negotiating QoS parameters and setting up bilateral agreements with neighbouring domains. On the intra-domain level a BB's responsibility includes configuration of edge routers, to enforce resource allocation and for admission control. Edge routers can be configured to (mainly) police and classify/mark packets with the corresponding DSCP (DiffServ Code Point). Policing entails ensuring the received rate does not exceed the agreed-on rate; if exceeded, depending on the adopted policy, excess packets are either discarded or re-marked for delayed discard if there is congestion.

4.1 Bandwidth Broker in DiffServ

The functional model of a Bandwidth Broker (BB) in the DiffServ domain is discussed in this section; we also elaborate on the importance of the BB in providing QoS in a DiffServ domain.

DiffServ architecture supports a simple mechanism to provide QoS to network traffic. The traffic entering a DiffServ domain is classified and conditioned at the boundary of the network and then assigned to different behaviour aggregates. The flows entering a domain are classified into one of many classes based on the value of the DSCP in the header of the packet. All packets with the same DSCP are treated in the same manner, and belong to the same behaviour aggregate (BA). The core routers forward packets, according to the treatment required, on the basis of their BA.

The main resource management entity in a DiffServ domain is the BB, which maintains policies and negotiates SLAs with clients and neighbouring domains. The interactions of BB with other components of a DiffServ domain, and the end-to-end communication process in a DiffServ domain are shown in Figure 2. This figure shows that when a flow needs to enter the DiffServ domain, or a local user wants to send traffic, the BB is requested to check related SLAs and the present traffic condition on the network. The BB decides whether or not to allow the traffic, on the basis of the previously-negotiated SLAs – to ensure new traffic does not violate current SLAs. If there is a new flow, the BB might have to negotiate a new SLA with the neighbouring domain(s) depending on the traffic requirements. Once the BB allows the traffic, the edge or leaf router needs to be reconfigured by the BB. SLA negotiation is a dynamic process that needs to take into account the ever-changing requirements of network traffic. The BB is responsible for admission control, as it has global knowledge of network topology and resource allocation.

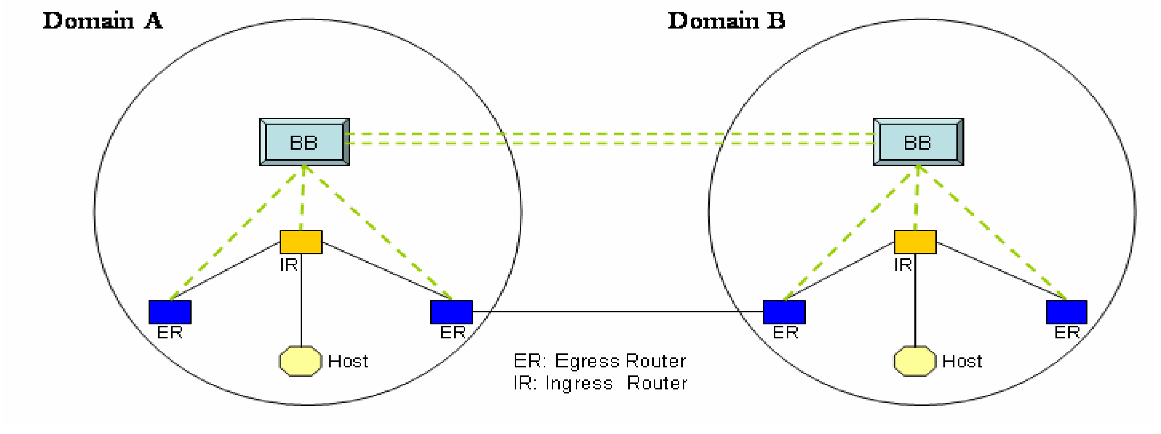

Figure 2: Role of Bandwidth Broker (BB) in DiffServ

4.2 Bandwidth Broker Architecture

A BB is a complex entity, and functionality provided by the BB is divided into four distinguishable parts: Inter-domain, Intra-domain, Database and User/application – as discussed in the following sub-sections and also shown in Figure 3.

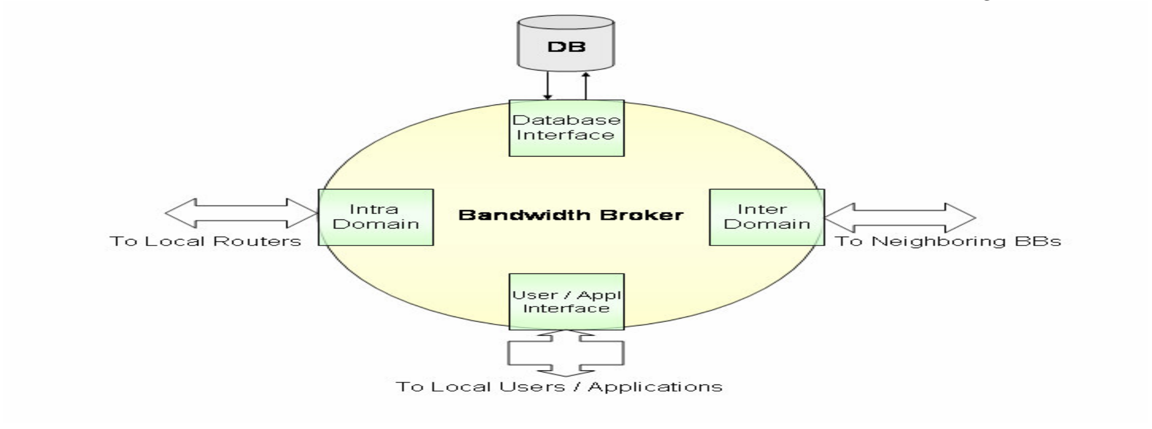

Figure 3: Bandwidth Broker Concept

- ***Inter-domain***

At inter-domain level, a BB communicates with neighbouring BBs to reserve resources in other domains. A BB needs this communication when the destination of the user's flow, i.e. which resources are requested, is in another DiffServ domain. There are a number of protocols that fulfil BBs intra-domain needs; but when it comes to inter-domain level, there is no single protocol that fits the requirements of the BB. Internet2 QBone BB Advisory Council proposed a simple inter-domain BB signalling protocol (SIBBS) [Teitelbaum *et al.* 1999]. The SIBBS protocol follows the request-response model between peer BBs, and is sender oriented. BBs have long running TCP connections with one another, with TCP providing the basic reliability and flow control for the protocol.

Whenever a BB receives a Resource Allocation Request (RAR) from another BB, it checks the sender's identity, the route, egress router for the flow, SLA related to user or flow and policies related to the flow. On acceptance of a request, if the destination of the flow is not in the BB's domain, the BB propagates RAR to its neighbouring BB which lies on the path of the flow. In this manner the RAR propagates to the BB which contains the destination host in its domain. The Resource Allocation Answer (RAA), which is the response to the RAR, is sent back from the destination BB to the source BB. In a situation where the request is to be rejected the BB sends the RAA back to the BB from which it received the request.

- ***Intra-domain***

On the Intra-domain level the BB needs to communicate with edge routers, as well as core routers to pass policy decisions – with the routers configured according to policy decisions to provide network QoS. There are multiple intra-domain protocols that can fulfil this requirement, such as Common Open Policy Services (COPS) [RAP, 2000], SNMP and Telnet; however intra-domain protocols used in the DiffServ domain are only of significance to the local network provider.

COPS is used to send policy decisions from the policy decision point (PDP) to the policy enforcement point (PEP). PEP has the ability to handle Internet Protocol (IP) traffic and implements policy-based admission control for data flows, whereas PDP has a complete view of the network, and configures its PEPs according to network policies. BB normally has the functionality of PDP, with all the edge routers configured as PEPs. COPS is a client/server protocol, where the server (PDP) has a TCP connection with all its clients (PEP) [Durham *et al.* 2000]. PEP maintains a Policy Information Base (PIB), described in Chan *et al.* [2003].

For support of policy specification, a new client type COPS for provisioning (COPS-PR) is introduced in [Chan *et al.* 2001], which is independent of the type of policy being supported – based on either QoS constraints or security concerns. COPS-PR has support for a real-time event-driven communication mechanism. PEP has only one connection to a PDP in one area of policy control, which supports large atomic transactions of data and efficient error reporting. It has state-sharing synchronization and only exchanges differential updates.

On 'boot-up' the PEP establishes a connection with a PDP and sends all device-relevant information. The PDP replies with all provisioned policies relevant to the device. If there is any change in policies at the PDP it sends an update message. Alternatively, if there is a change at the PEP end it sends the changes to PDP, which can in turn reply with new relevant policy provisioning elements. COPS has some features that makes it suitable for use with BB, and has a 'keep-alive' mechanism to ensure that a PDP remains active. COPS has an option for the PDP to redirect a client, if it does not support that client type or for load balancing purposes. As QoS-enabled services are highly vulnerable to denial and theft of services, COPS uses IP Security (IPSec) and integrity objects to provide the required security.

- ***Database***

The BB has a database interface to gather information for decision-making purposes. To provide QoS in the network, the BB must have a comprehensive picture of the complete network, and needs information on policy, Service Level Agreements (SLAs), network state information and current resource allocation status [Teitelbaum *et al.* 1999]. Routers can also be configured to provide monitoring data, to enhance the security of the network and improve resource usage. Router's configuration data and information about the BB's own components is also maintained for the purposes of fault tolerance. Many database management systems are available that can fulfil BB's database requirements, such as MySQL and Oracle.

- ***User/application***

There is a need for a protocol and interface for a network operator and application/user to interact with the BB. The network operator may use this interface to monitor or update the performance related features of the BB, while the application/user requires the protocol and interface to request, or query, the BB.

4.3 BB$_{Basic}$ Implementation

Bandwidth Broker$_{Basic}$ (BB$_{Basic}$) implementation provides almost all the features mentioned in section 4.2. It is implemented in Java and follows a client/server model. BB$_{Basic}$ can configure Linux-based routers, whereas the Linux routers need to have DiffServ support enabled; built into the Linux kernel from version 2.4 onwards. Java handles remote client/server functionality through TCP sockets. A BB$_{Basic}$ server can handle multiple connections from the routers and clients simultaneously. Once a request is accepted by BB$_{Basic}$, the tearing down process for the request is automated. The implementation provides a querying facility for users and network administrators, about resources, SLAs and requests. A detailed implementation description, and exact implementation mechanism for BB$_{Basic}$ is available in Pham and Nguyen [2003]. Some implementation details of BB$_{Basic}$ are briefly explained in the following sub-sections.

4.3.1 Inter-domain

The implementation of inter-domain protocol is embedded in (BB$_{Basic}$) which is designed on the specifications of SIBBS, addressed as SIBBS$_{Basic}$ here. The design specification of the SIBBS protocol [QBone, 2002] does not explicitly state the mechanism the BB uses to gather information about neighbouring BBs and details about their administrative domains. SIBBS$_{Basic}$ collects this information from its database, which contains a comprehensive network map, enabling BB$_{Basic}$ to identify the neighbour which should be contacted to complete a user's Resource Allocation Request (RAR). Whenever the resources requested include those from other domains, BB$_{Basic}$ gathers information from neighbouring BBs and contacts them via SIBBS$_{Basic}$. The neighbouring BB checks its resources, and, if the request is accepted, propagates the request to the next BB in the direction of the destination of flow. The process continues until the request reaches the BB containing the destination host in its domain, and replies propagate back in the reverse manner. After sending the Resource Allocation Answer (RAA), in case of request acceptance, BB$_{Basic}$ configures its edge routers via intra-domain protocol to allocate network resources for the accepted flow.

4.3.2 Intra-domain

COPS-PR is used as the intra-domain communication protocol for Bandwidth Broker$_{Basic}$ (BB$_{Basic}$). COPS-PR [Halim and Darmadi, 2000] is an independent implementation linked to BB$_{Basic}$. The COPS-PR and BB$_{Basic}$ was tested on Linux routers, and results [Halim and Darmadi, 2000; Pham and Nguyen, 2003] indicate that BB$_{Basic}$ manages the network resources effectively by reconfiguring the relevant routers with COPS-PR when required. BB$_{Basic}$ functions as a PDP which connects to its own domains routers (PEP), to configure them according to a pre-defined domain policy. Whenever BB$_{Basic}$ accepts a request, related core and edge routers, if required, are contacted via COPS-PR. The core router needs reconfiguration when it is a 'first hop' router for the flow; reconfiguration is required for marking and shaping the flow's packets. Marking of packets is required to classify the packet, and shaping is required to keep the flow under agreed limits. The edge router is contacted by BB when the destination or source of the requested flow is in a different DiffServ domain, to enable the edge router to filter/shape/schedule/mark the flow packets according to the Service Level Agreement (SLA).

4.3.3 Database

A MySQL database is used to store information critical for the BB. At the top level, the stored information in the database is divided into three distinct parts; user, BB and network. The user part of the database consists of a users' SLA, password and resource requests information. The BB part contains relevant information about peer BBs, and the SLAs with these BBs. The last part of the database contains all the information on the network which is essential to determine the routers which need reconfiguration when the BB accepts a request. Network information is also necessary to find the neighbouring BBs to contact, for resources acquired from multiple domains.

4.3.4 User/application

A useful feature of BB_{Basic} is its multiple interfaces that enable applications/users to access its network resource management ability. Providing multiple interfaces allow applications/users to choose the most suitable mechanism to interact with BB_{Basic}. Three distinct interfaces, for applications, users and resource managers are provided, namely a Java client, Web-based client and an XML/SOAP client. Detailed description of these interfaces and information about their usage is available in Pham and Nguyen [2003].

5. BB_{Basic} INTEGRATION

The BB_{Basic} is integrated into the G-QoSm framework as the Network Resource Manager. The integration of any resource manager into the G-QoSm takes the form of designing and building an interface module specific to that resource manager. This interface module, sometimes called a 'wrapper', resides in the Allocation Manager module within the G-QoSm architecture, and acts as a gateway to and from the resource manager; it translates requests from the Grid QoS-enabled Service (GQS) into requests understood by the corresponding resource manager. Requests can include:

- resource status and availability
- resource allocation, and
- resources release

Most resource managers implement functionality already provided by the GQS, such as resource reservation and policies, which means an overlap of some functionality. This overlap is interesting in the way it allows some flexibility; for example, the BB_{Basic} has Service Level Agreements (SLA) at the network level, denoted as $SLA_{network}$ to distinguish it from SLAs for computational or data storage resources. We can define two SLAs at the network level as $SLA_{network1}$ and $SLA_{network2}$, and can then define two resource capacities – i.e. pool of resource – one for 'Guaranteed' clients and the other for 'Best Effort' clients. We can then map the concept of the logical definition of the resource pool as viewed by G-QoSm, to the physical resources defined and managed by a resource manager, such as the $SLA_{network}$ in this case. Such mapping allows some degree of flexibility and is consistent with the adaptation strategy of the G-QoSm model outlined in Al-Ali *et al*. [2004b]; this flexibility lies in the ability of the GQS to manipulate the logical resource pool and conduct admission control checks, while not actually contacting the physical resources until this is necessary. Figure 4 shows a high level diagram of the BB_{Basic} integration.

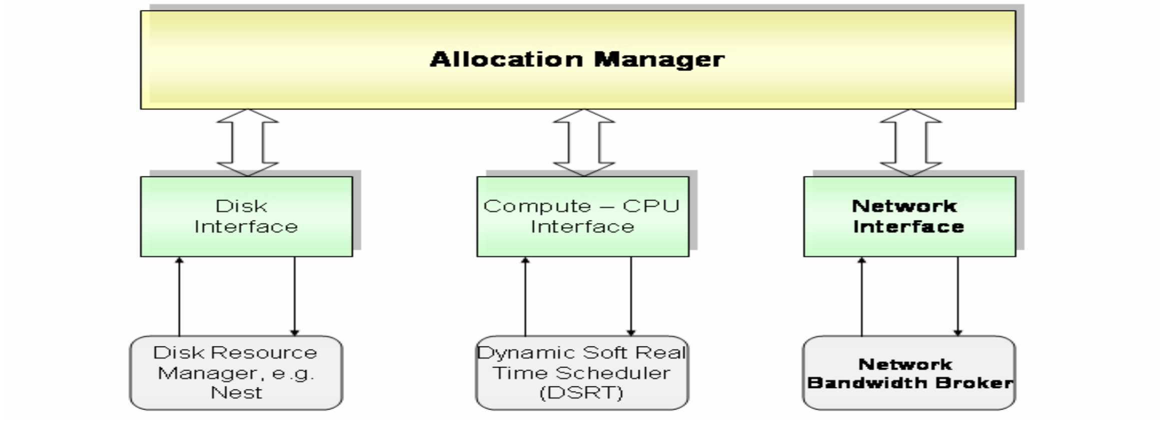

Figure 4: Integration of Various Resource Managers to G-QoSm

5.1 Network Interface

The Network Interface is the actual module that does the translation of requests from the GQS to and from the BB_{Basic}. The GQS may include four types of requests:

1. *Querying Resources*

   Resource querying can be classified into: a) Querying $SLA_{network}$, and b) Querying status of a Resource Allocation Request (RAR) within a particular $SLA_{network}$. Querying $SLA_{network}$ allows the GQS to inquire about the capacity of a specific network element currently being used, and the remaining capacity available for use. The second type of query allows the GQS to inquire about the status of a particular established RAR, and view associated information such as start/end time, amount of resource granted, type of network service, and source/destination IP addresses.

2. *Allocating Resources*

   This involves issuing a RAR associated with a pre-defined $SLA_{network}$. This request takes a number of parameters, such as the amount of network bandwidth required, the type of network service, the associated $SLA_{network}$, start/end time and source/destination IP address.

3. *Releasing Resources*

   The release, or de-allocation, of resources only works for pre-established RARs. Here, the RAR can be deleted – i.e. removal of network QoS privileges – with the parameter required for this request the RAR Identification number (ID).

4. *Modifying Requests*

   This type of request concerns modifying $SLA_{network}$ and RARs. For example, $SLA_{network}$ can be modified by increasing/decreasing its resource capacity, the effective domain it belongs to, or the type of network service being provisioned. Similarly, RARs can be modified to change their bandwidth capacity, and start/end time.

5.2 Requesting Network Resources

With this BB_{Basic} integration into the G-QoSm architecture, Grid applications will be able to request network resources with QoS constraints – as G-QoSm supports not only network resources but also computation resources, i.e. CPU. The protocol involved in requesting network resources is similar to that for computation resource as outlined in Al-Ali *et al.* [2004a]. G-QoSm extends the Java CoG Kit Core architecture and makes use of its application programming interface (API), which simplifies Grid applications interacting with underlying services.

Java CoG Kit Core

The Java CoG Kit is used to provide the interaction between a GQS and existing Grid middleware systems. The Java CoG Kit [von Laszewski *et al.* 2001] is a Java-based modular middleware used to provide access to various Grid implementations such as Globus Toolkit v2 (GT2) and v3 (GT3). One of the modules of the Java CoG Kit, called 'cog-core', provides the core functionality for such technology and architecture-independent interoperability. Cog-core provides APIs offering abstract Grid functionality, such as remote job execution, and file transfers, without any consideration for the underlying Grid implementation. For example, consider a Grid application developed using the APIs provided by cog-core. Since cog-core offers abstract functionality, irrespective of the back-end architecture, the same application can be executed on a variety of platforms. Hence, to run the application on a GT2 service, the user needs to merely mention a provider attribute to be GT2. The same application can be later executed on a GT3 service, without any modification to its implementation, by simply changing the provider attribute from GT2 to GT3.

Cog-core contains the required functionality for mapping the abstract application requirements into back-end specific details controlled by the corresponding provider attribute. To provide seamless interaction between Grid applications and the QoS-aware Grid resources, we augment the functionality of cog-core by incorporating QoS-related parameters. All the necessary logic and implementation overhead for QoS management is thus embedded into cog-core, by changing the provider attribute to 'QoS'. In the remaining part of this section we describe the two basic constructs of the cog-core library and its enhancement to support QoS-enabled services.

Task: Cog-core defines a Task as an atomic unit of execution. It abstracts generic Grid functionality such as user authentication, remote job execution, file transfer requests and information query. It has a unique identity, a security context, a specification, a service contact and a provider attribute. The task identity helps to uniquely represent the task across the Grid. The security context represents the abstract security credentials of the task. Most back-end Grid implementations will have their own notion of a security context, and the security context in cog-core offers a common construct, that can be extended by different implementations, to satisfy the corresponding back-end requirement. The specification represents the actual attributes or parameters required for the execution of the Grid-centric task. The generalized specification can be extended for common Grid tasks such as remote job execution, file transfer and information query. The service contact associated with a task symbolizes the Grid resource (service) required to execute it, and the provider attribute specifies the desired back-end Grid implementation for the task.

Handlers: The Task Handler provides a simple interface to interact with a generic Grid task. It categorizes the tasks and provides the appropriate functionality for it, based on the provider attribute of the submitted task. Cog-core contains a separate handler for each of the operations it supports. These handlers then map the generic Grid parameters of the task into the back-end implementation specific Grid functionality. To augment the cog-core functionality into the QoS domain we provide a QoS handler that holds the QoS-related implementation and logic. The QoS task handler manages the negotiation, task execution and data redirection between the application and the GQS.

For a Grid application to request a network resource with QoS provision, a few simple steps need to be considered. The application developer has to specify the QoS parameters that must be considered during QoS negotiation, including start/end time, resource type, in this case network, and specifications, such as bandwidth. Once the task object has been specified, the QoS Handler is delegated, on behalf of the client or application, to negotiate QoS requests. In this case the QoS Handler is seen as the client from the GQS point of view. This is a useful approach especially when the application requires more than one Grid resource. All the application needs is to instantiate the required number of QoS Handler objects, submit the Task object to the handlers, and let the handlers negotiate QoS requests with the GQS to return an agreement.

The Java code segment in Figure 5 demonstrates how an application can generate a network QoS negotiation request to a QoS handler.

```
public class QosNegotiation {
   static Logger logger =
Logger.getLogger(QosRequest1.class.getName());
   private Task task;

   public QosNegotiation() {
      prepareQosNegotiationTask ();
      submitQosTask();

         // get SLA identifier, stored in 'agreementToken'
         String status = (String)
this.task.getAttribute('agreementToken');

         if (status != null) {
            System.out.println('Request has SUCCEEDED and
the agreementID is:'+ status);
         } else
            System.out.println('Your request has FAILED!');
   }

/*** QoS: Prepare Negotiation Task ***/
private void prepareQosNegotiationTask() {
  // create a QoS service and setup QoS attributes for network
resource
  Task task = new QosTaskImpl(``myTask",
QoS.NEGOTIATION);
  this.task.setAttribute(``startTime", startTime);
  this.task.setAttribute(``endTime", endTime);

this.task.setAttribute(``networkBandwidth",networkBandw
idth);
  this.task.setAttribute(``sourceIP",soruceIP);
  this.task.setAttribute(``destIP",destIP);

  // create a Globus version of the security context
  SecurityContextImpl securityContext =
     new GlobusSecurityContextImpl();
  // selects the default credentials
  securityContext.setCredential(null);
  // associate the security context with the task
  task.setSecurityContext(securityContext);

  // create a contact for the Grid resource
  Contact contact = new Contact(``myGridNode");

  // create a service contact
  ServiceContact service = new
  ServiceContactImpl(qosServiceURL);
  // associate the service contact with the contact
  contact.setServiceContact(``GQSurl",service);

  // associate the contact with the task
  task.setContact(contact);
}


/*** QoS: Task Submission to QoS Handler ***/
private void submitQosTask(Task task) {
  TaskHandler handler = new QoSTaskHandlerImpl();
  // submit the task to the handler
  handler.submit(task);
}
```

Figure 5: Java Code Segment showing how a Grid Application can Request a Network Resource with QoS Provision

6. EXPERIMENTAL RESULTS

To evaluate the effectiveness of network resource reservations based on the integration of the BB_{Basic} and G-QoSm framework, we have built a local network test-bed. This section discusses the experiments and the corresponding results.

Network Test-bed

The network test-bed makes use of a local area network, consisting of computing nodes and routing elements. Figures 6 and 7 depict the network setup, with the computing nodes representing the source and sink points – i.e. network traffic senders and receivers. For the routing elements we used the `iproute2` package in the Linux operating system, which has DiffServ capability; Linux machines, with `iproute2`, act as network routing elements. Figure 6 demonstrates the intra-domain architecture, while Figure 7 shows an inter-domain architecture.

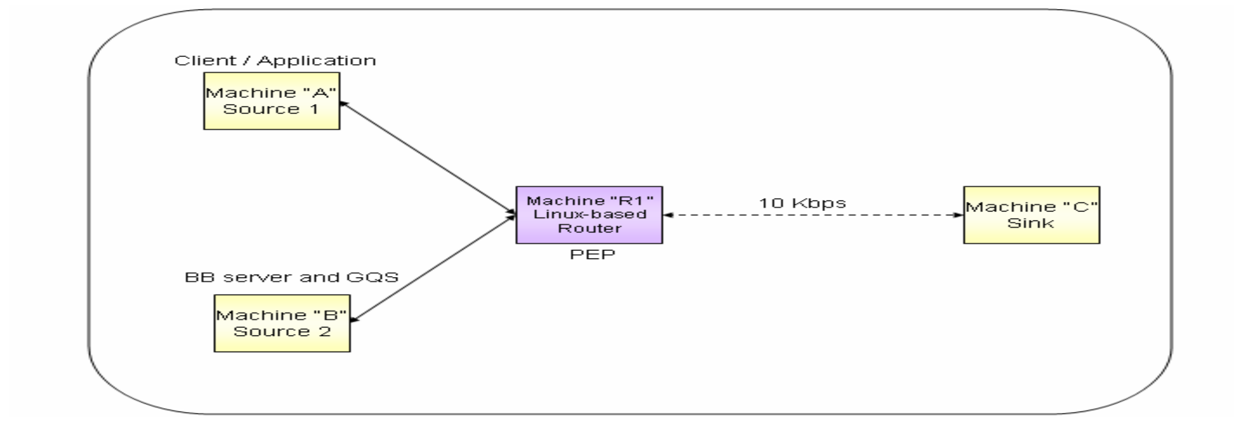

Figure 6: Network Setup for Intra-domain Architecture

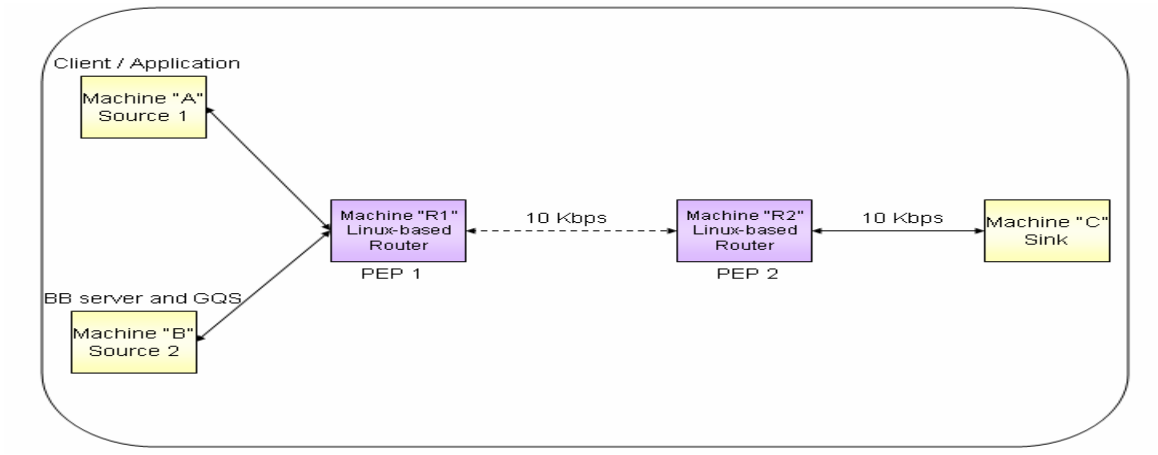

Figure 7: Network Setup for Inter-domain Architecture

BB$_{Basic}$ comes with three separate modules; i) a BB server, which needs to be installed and operational in each administrative domain, to act as a Policy Decision Point (PDP), ii) a Policy Enforcement Point (PEP) module, which needs to be installed and operational in each Linux routing element, and iii) a MySQL database, which needs to be built and populated with the relevant data describing the network and policies; for example, the database should be populated with data describing the network topology and link capacities, as well as the pre-defined SLAs, and their service type, whether expedited forwarding (EF) or 'best effort' (BE), and which SLA can be utilized in which domain. In addition the G-QoSm software acts as the QoS management entity that provides the client/application with an interface to request/modify/cancel network resource reservations.

Demonstration of Network QoS for Grid Applications

The main objective of these experiments is to show that a Grid application can initiate a network reservation request, through the G-QoSm service, that is forwarded to BB$_{Basic}$ – which supports admission control and configures routing elements accordingly.

Network traffic generator tools are used at the source to simulate applications requiring data transfer, and similarly, network traffic collector's tools are used at the sink to collect the traffic received and measure the network traffic rate – i.e. network bandwidth. The network traffic generator tools are: *Real-time UDP Data Emitter* (RUDE) and *Collector for RUDE* (CRUDE) [RUDE & CRUDE 2004]. Finally, the measurement of the traffic performance is plotted.

In the following experiment scenarios we used a UDP constant traffic rate generator to generate network traffic that simulates Grid application demand, and similarly we used a UDP traffic generator to generate competing traffic – i.e. congestion traffic. It is important to use UDP for traffic congestion, as opposed to TCP, because UDP does not employ the 'slow-start' mechanism during congestion which allows us to maintain congestion behaviour.

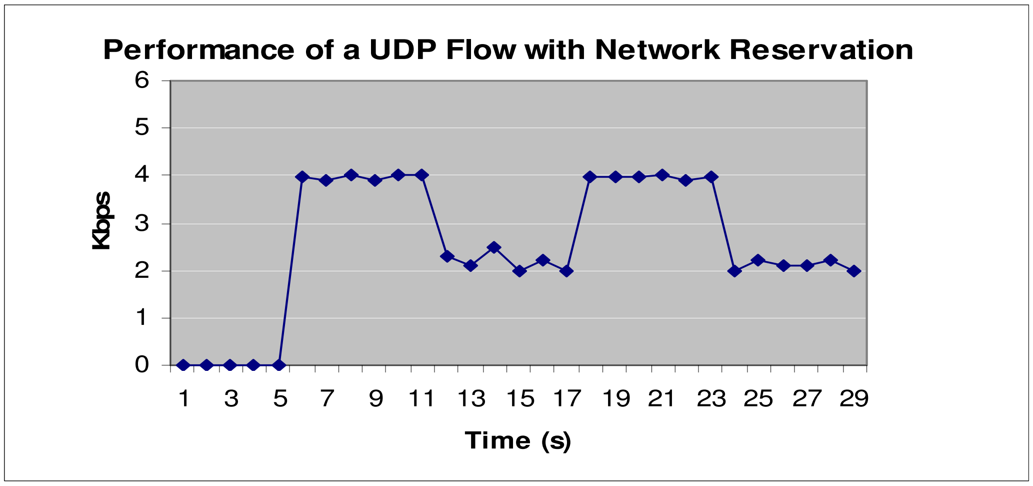

Figure 8: Network QoS under Congestion

Figure 8 shows the performance of network QoS for UDP traffic simulating a Grid application under different situations. This experiment was conducted in the intra-domain architecture shown in Figure 6. The link between the router element and the sink was configured for a 10 Kbps stream so as to be able to easily congest the link. The UDP traffic under consideration was started at time t_5 to t_{29}. From t_5 to t_{10} the UDP traffic was sending at 4 Kbps on an unloaded communication link from source to sink and without reservation – i.e. 'best effort'. From t_{11} to t_{16}, with the UDP flow still transmitting at 4 Kbps, a number of random competing traffic was started to generate congestion, and the result observed shows that the UDP traffic could not maintain the 4 Kbps rate due to congestion. A network QoS reservation was made from t_{17} to t_{23} for the UDP traffic for 4 Kbps, with the competing traffic still generating congestion.

The performance result of the QoS reservation shows that the UDP traffic manages to maintain the promised reservation rate even though the congestion is still operating. Finally, from t_{24} to t_{29}, the reservation has ended and the UDP traffic was not able to keep its 4 Kbps due to congestion.

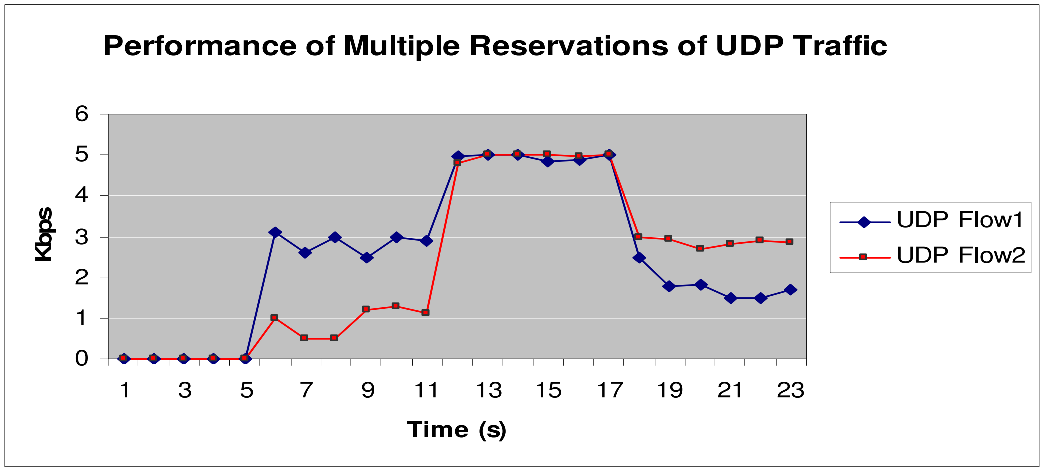

Figure 9: Multiple Network QoS Flows under Congestion

Figure 9 demonstrates multiple network QoS reservations under congestion. The setup is similar to the previous one, with the link between the router and the sink configured to 10 Kbps. In this case two UDP flows are under consideration. From t_5 to t_{10} the two UDP flows were simultaneously transmitting at rate 5 Kbps while the traffic random generation is on. Reservations were started from t_{11} to t_{16} and the two flows maintained the promised resources. Thus, the DiffServ forwarding mechanism at the routing element is performing the correct traffic forwarding.

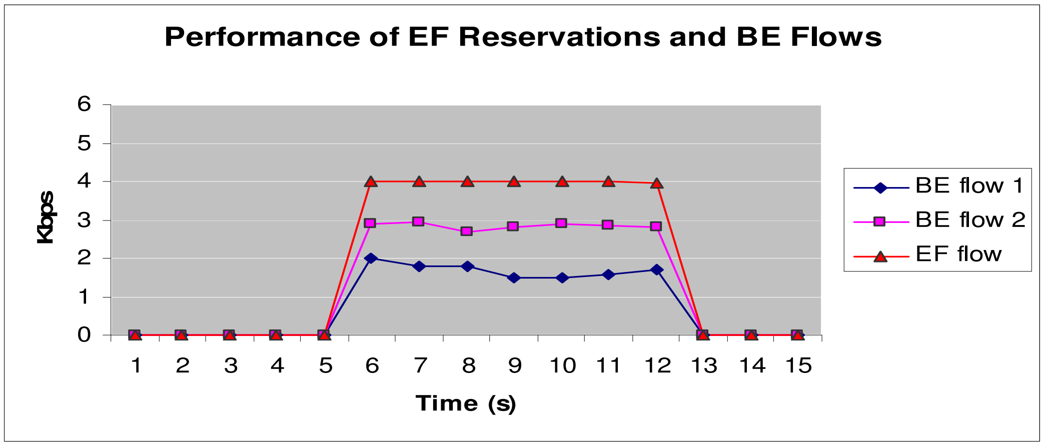

Figure 10: Guaranteed and Best Effort Network QoS

Figure 10, shows performance results of transmitting multiple traffic flows belonging to two different classes i) Expedited Forwarding (EF), which could be mapped to ‘Guaranteed’ service in G-QoSm, and ii) ‘Best Effort’ (BE) traffic. The network link from the routing element to the sink is configured to support 10 Kbps; two SLA$_{network}$ were generated; one for the EF traffic of total capacity of 5 Kbps, and the other for BE traffic of total capacity of 5 Kbps. Further, using the Linux traffic control (tc) script we configured the communication link so that ‘borrowing’ is not allowed, meaning that each type of traffic, EF and BE, must stay within the boundaries of the defined resource. The traffic performance was realized from time t_5 to t_{13}, when a network reservation was made for EF flow – i.e. ‘Guaranteed’ service – of 4 Kbps. At the same time two BE flows were attempting to transmit at a rate of 5 Kbps each. The EF flow maintained the reserved rate of 4 Kbps, while the two BE flows are less, although attempting to transmit at 5 Kbps each, because the BE network source was configured for a maximum capacity of 5 Kbps with no borrowing. Therefore, the routing element shaped, and policed, the two BE flows to fit within the configured BE network resource.

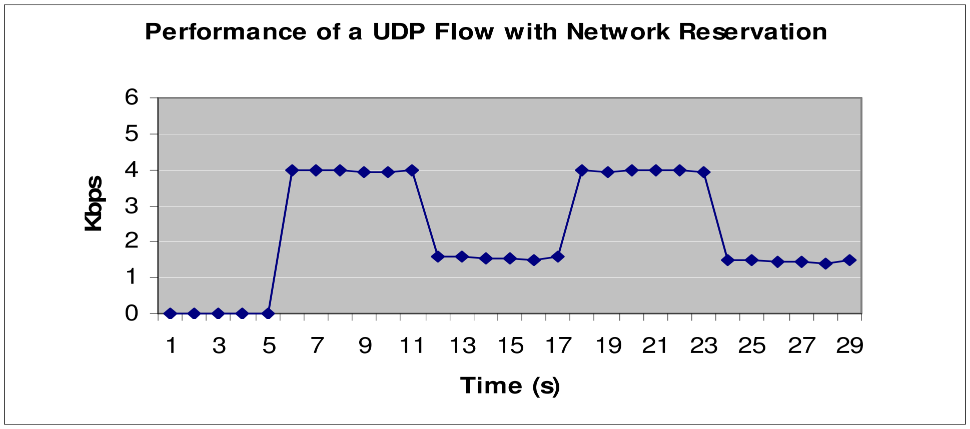

Figure 11: Network QoS under Congestion – Intra-domain Architecture

To further verify the inter-domain communication between the BB_{Basic} and PEPs, we conducted similar experiments to the ones conducted for the intra-domain described in Figure 8. Figure 11 shows the results, of the inter-domain case, that are very similar to the intra-domain, implying that the BB_{Basic} is able to configure local PEPs as well as remote PEPs. This simple concept of inter-domain communication can be replicated over a large number of administrative domains within Grid infrastructures; making the proposed architecture scalable.

7. CONCLUSION

The provision of network Quality of Service (QoS) to support Grid applications is presented – essentially based on the IETF DiffServ model. A Bandwidth Broker is identified as the key architectural component necessary to request, monitor and support QoS management.

Network QoS, in the context of Grid applications, is discussed in a wider context – which must also involve support for managing computational power and storage, with the Grid QoS Management (G-QoSm) framework implemented in support.

This paper focuses on combining G-QoSm and a Bandwidth Broker (BB), and subsequent evaluation using a network established with Linux-based routers.

A key limitation in any network QoS work is the ability to manage and control traffic flow at non-end-point ("interior") routers. This is especially true in deploying Grid applications, where such routers may not be owned by one individual or institution. Enforcing such intermediate routing elements to conform to a defined policy is a difficult task to achieve in reality. This is often the reason why end-point-based feedback mechanisms are employed in multi-media applications.

Our approach, based on the DiffServ model, also requires intermediate routers to adopt the DiffServ-expedited forwarding model. Hence, the approach presented here is restricted to routers that support this model – providing a traditional 'best-effort' service at all other routers.

We know of no other work, in the QoS-related field, which can bypass this need to manage intermediate routers.

8. REFERENCES

Al-Ali, R; Rana, O; Walker, D; Jha, S and Sohail, S. (2002). G-QoSm: Grid Service Discovery using QoS Properties. *Computing and Informatics Journal*, 21(4):363-382, 2002.

Al-Ali, R; Amin, K; von Laszewski, G; Rana, O and Walker, D. (2003). An OGSA-Based Quality of Service Framework. *Proceedings of the Second International Workshop on Grid and Cooperative Computing (GCC 2003)*, Springer Verlag, Shanghai, China, December 2003.

Al-Ali, R; Amin, K; von Laszewski, G; Hategan, M; Rana, O; Walker, D and Zaluzec, N. (2004a) QoS Support for High-Performance Scientific Applications. *Proceedings of the IEEE/ACM 4th International Symposium on Cluster Computing and the Grid (CCGrid 2004) IEEE Computer Society Press*. Chicago IL, USA, April 2004.

Al-Ali, R; Hafid, A; Rana, O and Walker, D. (2004b). An Approach for QoS Adaptation in Service-Oriented Grids. *Concurrency and Computation: Practice and Experience Journal*, 16(5):401-412, 2004.

Aurrecoechea, C; Campbell, A and Hauw, L. (1995). A Survey of Quality of Service Architectures. Technical Report MPG-95-18, Lancaster University, 1995.

Barden, R; Clark, D and Shenker, S. (1994). *Integrated Services in the Internet Architecture: an Overview*. Internet request for Comments RFC1633, IETF, June 1994.

Bhatti, S; Sørensen, S; Clarke, P and Crowcroft, J. (2003). Network QoS for Grid Systems. *International Journal of High Performance Computing Applications, Special Issue on Grid Computing: Infrastructure and Applications*, 17(3), 2003.

Blake, S; Blake, D; Carlson, M; Davies, E; Wang, Z and Weiss, W. (1998). An Architecture for Differentiated Service. Internet RFC 2475, 1998.

Chan, K; Sahita, R; Hahn, S and McCloghrie, K. (2003). *Differentiated services quality of service policy information base* . Internet request for comments RFC3317, IETF, March 2003.

Chan, K; Seligson, J; Durham, D; Gai, S; McCloghrie, K; Herzog, S; Reichmeyer, F Yavatkar, R and Smith, A. (2001). *Cops usage of policy provisioning (COPS-PR)*. Internet request for comments RFC3084, IETF, Mar 2001.

Chu, H and Nahrstedt, K. (1999). CPU Service Classes for Multimedia Applications. In *IEEE International Conference on Multimedia Computing and Systems '99*, Florence, Italy, 1999.

Durham, D; Boyle, J; Cohen, R; Herzog, S; Rajan, R and Sastry, A. (2000). *The COPS (common open policy service) protocol*. Internet request for comments RFC2748, IETF, Jan 2000.

Foster, I; Kesselman, C; Lee, C; Lindell, R; Nahrstedt, K and Roy, A. (1999). A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation. In *Proceedings of the IEEE/IFIP International Workshop on Quality of Service (IWQOS'99)*, pages 27-36, London, UK, 1999.

Hafid, A and Bochmann, G. (1998). Quality of Service Adaptation in Distributed Multimedia Applications. *ACM Springer-Verlag Multimedia Systems Journal*, 6(5):299-315, 1998.

Halim, H and Darmadi, M. (2000). Implementation of Bandwidth Broker using COPS-PR. *Honours thesis report, School of Computer Science and Engineering*, UNSW, Nov 2000.

Pham, K and Nguyen, R. (2003) *Implementation of Bandwidth Broker in Java*. Undergraduate thesis report, School of Electrical Engineering and Telecommunications, UNSW, Jun 2003.

QBone Signaling Design Team. (2002). Final Report: http://qos.internet2.edu/wg/documentsinformational/20020709-chimento- etal-qbonesignaling/

RAP, (2000). *Resource allocation protocol (rap)* At http://www.ietf.org/charters/ manet-charter.html, 2000.

RUDE & CRUDE, (2004). Web Site: http://rude.sourceforge.net/. July 2004.

Sohail, S; Pham, K; Nguyen, R and Jha, S. (2003). Bandwidth Broker Implementation- Circa-Complete and Integrable. *Proceedings of 7th International Symposium on Digital Signal Processing and Communication Systems*, Coolangata, Australia, 2003.

ShaikhAli, A; Rana, O; Al-Ali, R and Walker, D. (2003). *UDDIe: An Extended Registry for Web Services.* Proceedings of the Service Oriented Computing: Models, Architectures and Applications, SAINT-2003 IEEE Computer Society Press. Orlando Florida, USA, January 2003.

Teitelbaum, B; Hares, S; Dunn, L; Neilson, R; Vishy Narayan, R and Reichmeyer, F. (1999) Internet2 QBone: Building a testbed for differentiated services, *IEEE Network*, 13(5):8–16, September/October 1999.

von Laszewski, G; Foster, I; Gawor, J and Lane, P. (2001). A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9), 2001.

Xiao, X and Ni, L. (1999). Internet QoS: A Big Picture. *IEEE Network*, 13(2):8-18, 1999.

Glossary

API	Application Programming Interface
BA	Behaviour Aggregate
BB	Bandwidth Broker
BE	Best Effort
COPS	Common Open Policy Services
COPS-PR	COPS for provisioning
DiffServ	DIFFerentiated SERVices
DSCP	DiffServ Code Point
DSRT	Dynamic Soft Real Time Scheduler
EF	Expedited Forwarding
ER	Egress Router
GARA	General-purpose Architecture for Reservation and Allocation
GQS	Grid QoS-enabled Service
G-QoSm	Grid QoS Management
GRAM	Globus Resource Allocation Manager
GTx	Globus Toolkit version *x*
IETF	Internet Engineering Task Force
IntServ	INTegrated SERVices
IP	Internet Protocol
IPSec	IP Security
IR	Ingress Router
Java CoG Kit	Java Commodity Grid Kit
NRS	Network Resource Scheduler
OGSA	Open Grid Service Architecture
P2P	Peer-to-Peer
PEP	Policy Enforcement Point
PDP	Policy Decision Point
PIB	Policy Information Base
QoS	Quality of Service
RAA	Resource Allocation Answer
RAP	Resource allocation protocol
RAR	Resource Allocation Request
SIBBS	Simple Inter-domain BB Signalling protocol
SLA	Service Level Agreement
tc	Linux traffic control
TCP	Transmission Control Protocol
TSpec	Traffic Specification
UDDI	Universal Description Discovery and Integration
UDDIe	extended UDDI
UDP	User Datagram Protocol