BROOKHAVEN
NATIONAL LABORATORY

BNL-96957-2012 -CP

Design and Implementation of an Intelligent End-to-End Network QoS System

Sushant Sharma, Dimitrios Katramatos, Dantong Yu, Li Shi

The 21st International ACM Symposium of High-Performance Parallel and Distributed Computing
Delft, Netherlands
June 18-22, 2012

February 2012

Computational Science Center

Brookhaven National Laboratory

U.S. Department of Energy
Office of Science
Office of Advanced Scientific Computing Research

Design and Implementation of an Intelligent End-to-End Network QoS System

Sushant Sharma Dimitrios Katramatos
Dantong Yu
Brookhaven National Laboratory
Upton, NY, USA, 11973.

Li Shi
Department of ECE
Stony Brook University
Stonybrook, NY, USA, 11794.

ABSTRACT

End-to-End guaranteed network QoS is a requirement for reliable and predictable data transfers between geographically distant end hosts. Existing QoS systems, however, do not have the capability or intelligence to decide what resources to reserve and which paths to choose when there are multiple and *flexible* resource reservation requests. In this paper, we design and implement an intelligent system that can guarantee end-to-end network QoS for multiple flexible reservation requests. At the heart of this system is a polynomial time scheduling algorithm called resource reservation and path construction (RRPC). The RRPC algorithm schedules multiple flexible end-to-end data transfer requests by jointly optimizing the path construction and bandwidth reservation along these paths. We also show that constructing such schedules is an NP-hard problem. We implement our intelligent QoS system, and present the results of deploying the system on real world production networks such as ESnet and Internet2. An important aspect of our implementation is that it does not require any modification or any new software to be deployed on the routers within the network.

1. INTRODUCTION

The networking research community has dedicated significant amounts of effort in developing novel QoS mechanisms. Such mechanisms include architectures for implementing QoS (such as Diffserv [3] and Intserv [4]), as well as protocols/tools that can be used to reserve resources within QoS enabled networks (such as RSVP [2, 5] and TeraPaths [14]). Furthermore, research and education networks such as Internet2 [13], ESnet [8], and GEANT [9] connect large number of educational and government institutions, and allow these institutions to reserve available resources (e.g., bandwidth[1]). Although such networks allow reservation of resources, these

[1]In this paper, we consider bandwidth as the resource to be reserved, and use the terms "bandwidth" and "resource" interchangeably.

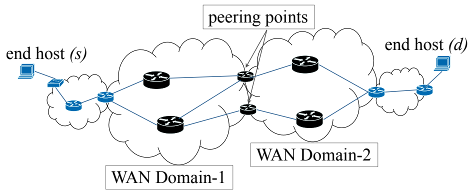


Figure 1: An example network connecting two end hosts. Path from one end host to the other goes via two Wide Area Networks and two Local Area Networks.

reservation capabilities are only a first step towards achieving a true end-to-end network QoS. Note that our focus in this paper is not on providing intelligent QoS within Internet. Instead, we focus on research and education networks such as Internet2 and ESnet that allow for bandwidth reservations.

True end-to-end QoS is achieved when resources can be reserved along the complete path between two end hosts (e.g., the two end points of a TCP connection). This complete path includes the local area networks of the end sites (such as educational institutions and government organizations), in addition to any intermediate domains (such as Internet2 or ESnet) that connect these end sites (see e.g., Fig. 1). In almost all end sites, end hosts usually do not have a direct connection to the connecting router of the intermediate domain. The route from an end host to the border router usually goes via few other intermediate routers. As a result, in order to guarantee a true end-to-end network QoS, it is important for the end sites to also have resource reservation capabilities within their local area networks (LANs). The task of reserving resources becomes even more challenging as the end site networks and the intermediate domains may not implement the same QoS enabling technologies (e.g., Intserv, Diffserv, MPLS, etc.). A reservation tool should be able to coordinate resource reservations among these heterogeneous networks. Such a reservation tool forms an important component of the system that can provide a true end-to-end intelligent network QoS. Note that we do not aim to provide an alternative to existing QoS enabling technologies. Instead, our proposed intelligent system is built on top of these technologies. We assume the existence of bandwidth reservation mechanisms within network domains along the end-to-end path.

One important limitation of existing QoS systems is the lack of intelligence/flexibility in reserving requested resources.

That is, existing systems typically reserve exactly the amount of resources that were requested. However, many times, user requests are flexible or less rigid in terms of bandwidth and time requirements. Handling flexible requests require additional intelligence to be built into the current QoS ecosystem. In general, user requests have the following characteristics:

- Requests can be flexible in nature, e.g., a user may require a set of data to be successfully transferred from one end point to another by a certain deadline. Any resource reservation that can meet that deadline would be acceptable.
- There may be multiple flexible requests that are needed to be satisfied.
- Requests can be available ahead of time, in advance. As an example, scientists at one site may know that their experiments are going to generate a certain amount of data by a certain time. This data would need to be transferred for analysis to a site located in a geographically different location in a timely manner. In such a case, it is desired to reserve an appropriate amount of network resources ahead of time.

The above characteristics create situations where there can be multiple end-to-end paths that can satisfy reservation requests. Furthermore, there could be multiple options on how much bandwidth (and time) needs to be reserved to satisfy reservation requests. This is beyond the capabilities of current QoS systems as there is no more a single or obvious option, and the QoS system needs to intelligently choose a solution among many options. Our goal in this paper is to design and implement an end-to-end QoS ecosystem that is capable of accommodating and intelligently scheduling multiple and flexible resource reservation requests.

1.1 Desired Properties

We now list a set of properties/capabilities that an intelligent end-to-end network QoS system should possess in order to accommodate multiple flexible reservation requests.

1. The QoS system should have the capability to collect and maintain knowledge about the availability of resources within all network domains (local, remote and intermediate) that connect the two end hosts.
2. Given the resource availability and a set of reservation requests, the QoS system should be able to make intelligent reservations across all domains connecting two end hosts. Such resource reservations should satisfy all reservation requests, if possible.
3. The QoS system should be flexible enough so that the researchers can implement and test different scheduling (i.e., resource reservation) algorithms with minimum effort.

Our goal in this paper is to design and implement an end-to-end network QoS system that possess all of the above properties, and should be able to accommodate and intelligently schedule multiple flexible resource reservation requests.

1.2 Contributions

The following are the main contributions of our work in this paper:

1. We present the design of a QoS system that can accommodate and intelligently schedule multiple flexible resource reservation requests between two end hosts. These hosts may not belong to the same local area network (generally they are also geographically distant as shown in Fig. 1).
2. We consider a novel problem of scheduling multiple flexible resource reservation requests. We prove that the problem is NP-hard and present an efficient heuristic, called RRPC, to solve it.
3. We implement the scheduling algorithm as well as the QoS system that we propose. Our implementation runs on real hardware. Furthermore, our system does not require updating or installing any new software on the routers within the network.
4. We have deployed our system on real world production networks. Our QoS system is stable and does not have any negative effect on the operation of regular users of the production networks. The results presented in this paper are obtained from real world networks.

The rest of this paper is organized as follows: In Section 2, we describe the architecture for the intelligent QoS system in detail. In Section 3, we discuss a scheduling problem and an efficient algorithm to solve it. In Section 4, we discuss the details of the deployment of our system on real world networks, and present results. Section 5 discusses the related work, and Section 6 concludes this paper.

2. SYSTEM ARCHITECTURE

2.1 Overview

Networking domains between the two end hosts usually employ heterogeneous technologies to enable QoS (dynamic circuits, MPLS tunnels, Diffserv, Intserv, etc.). As a result of this heterogeneity, providing a true end-to-end QoS requires a mechanism for these heterogeneous systems to coordinate with each other. This requirement motivates the need for the first component in our architecture called *Domain Controller* (DC). Each intermediate domain that connects the two end hosts has a DC. Such DC coordinates with other domains and exposes a set of services to enable resource reservations within its own network. The underlying mechanism to enable reservations can be different for different domains.

In addition to performing coordination with other domains, each DC further requires a mechanism to obtain and reserve the resources within its own LAN. In order to accomplish this, we introduce another component called *LAN-manager.* Each DC communicates with its corresponding LAN-manager to obtain and reserve necessary resources within its LAN.

The third and most important component of our architecture that instills the intelligence within the QoS ecosystem,

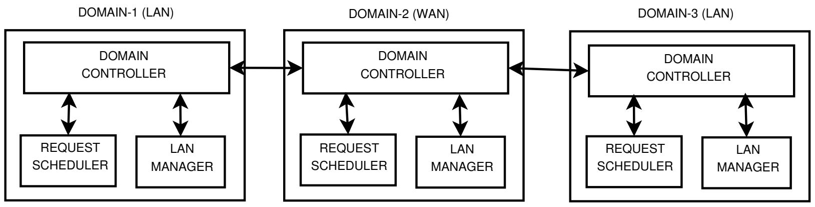


Figure 2: Architecture of an intelligent end-to-end network QoS system.

```
class payloadInfo {
    long[] amount_of_data;
    // amount of data that needs to be
    // transferred for each flexible
    // request.

    long[] sTime;
    // times when the data will become
    // available for transfer

    long[] eTime;
    // time by which data transfer should
    // complete for each request

    long[] rate_limit;
    // maximum rate at which data for
    // each request can be transmitted

    String objective;
    // objective of the data transfer

    String srcIP, dstIP;
    // source host and destination host

    String port_range;
    // the range of ports that will
    // be used for data transfer
}
```

Figure 3: Data structure for the reservation request.

is called the *Request Scheduler* (RS). The objective of RS is to accept the resource availability and reservation requests as input, and to construct a feasible reservation schedule that satisfies the reservation requests according to some objective. An RS is invoked by a DC. Figure 2 shows components of the proposed architecture for our intelligent end-to-end network QoS system. We now describe the three components of our architecture in detail.

2.2 Domain Controller (DC)

The domain controllers are the glue that holds the complete intelligent QoS ecosystem together. An application that wants to make multiple end-to-end reservations between two end hosts will contact a DC in order to submit the reservation requests. The DC being contacted can belong to either one of the two end site LANs, and is referred to as *local DC*. The DC for the LAN of the other end host is referred to as *remote DC*. The goal is to provide sufficient information to the local DC that will help it to perform or initiate appropriate configurations within the local network domain, remote network domain, and across all the intermediate domains.

Description of the reservation request. An application can use the data structure shown in Fig. 3 to submit multiple reservation requests to the local DC. The `amount_of_data` array contains the amount of data that needs to be transferred as part of each flexible request. The variable `sTime` indicates the time when the data will become ready to be transferred for each request, and the variable `eTime` indicates the deadline by which the application would like the data transfer to finish for each request. The `rate_limit` array specifies the maximum rate at which the data for each request can be transmitted. As an example the data that is being read from a particular disk will be limited by the rate at which data can be read from that disk. Therefore, even if network can support a higher data transfer rate, this field will indicate that there is no need to reserve a larger amount of bandwidth. The `objective` variable indicates the objective of data transfer for all requests. The objective value dictates the algorithm that the RS will execute in order to schedule the reservations. As an example, an intuitive objective could be to make reservations for the submitted requests in a way that minimizes the sum of the data transfer times of all the requests. This is also the objective that we consider later in Section 3 where we develop our scheduling algorithm. The `srcIP` and `dstIP` indicate two end hosts. The `port_range` variable indicates the range of port numbers (e.g., port 20 for FTP) that the data transfer application will use to perform data transfer. This port range is required by the LAN-managers to configure the routers within the local and remote LANs.

Once a DC receives the reservation request, it performs the following tasks to determine the resource availabilities:

- The first step is to determine the intermediate domains that connects the two end sites. As an example, the Brookhaven National Lab (BNL) is connected to the Lawrence Berkeley National Lab (LBL) via one domain (ESnet), whereas BNL is connected to the University of Michigan (UMICH) via two intermediate domains (ESnet and Internet2). Currently, we maintain this information in a local database at each end site. However, for large numbers of end sites and intermediate domains we plan to develop a distributed database design in future.

- In the second step, the local DC contacts the LAN-manager of its own site and obtains the amount of available resources.

- In the third step, the local DC contacts the remote DC and obtains the resource availability within the remote LAN.

- In the last step, the local DC contacts the DCs for each of the intermediate domains and collects the resource availabilities within these domains.

The precise manner in which the resource availability is depicted is described in Section 2.3, where we also describe the operation of a LAN-manager. After collecting the necessary resource availability information, the local DC passes on this information to its RS along with the associated reservation request. The DC then waits for the scheduling results from the RS. Once the scheduling results are communicated back to the DC, the DC further communicates these results to all other DCs involved (itself, remote, and intermediate) along the path. The objective of this communication is to indicate to the other DCs that the network devices within their LANs should now be configured in accordance with the generated schedule. It is the responsibility of the LAN-manager to actually configure the network devices within the LAN in accordance with the generated reservation schedule.

Once the network devices within all domains (local, remote, and intermediate) have been configured, the local DC communicate the reservation schedule back to the application that requested the reservations. The application can now schedule the data transfers between two end hosts in accordance with the communicated reservation schedules.

2.3 LAN-Manager

A LAN-manger can be subdivided into two main components based on the functions they perform. The two components are as follows:

- **Resource manager.** It receives the resource availability requests from the DC, and replies back with the resource availability within the LAN.
- **Network device manager.** It receives the reservation schedule from the DC, and configures the network devices within LAN in accordance with the reservation schedule received.

We next describe these components in detail.

2.3.1 *Generating Resource Availability*

Upon receiving the request for resource availability, the LAN-manager has to respond with the resource availability along the portion of the path/paths within its domain.

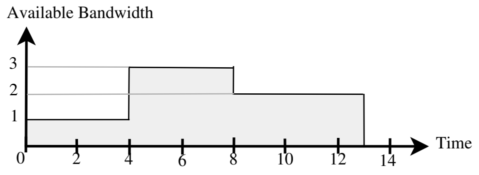
End-site LANs. For the end site LAN domains, these paths connects the end host to the border router. The border router of an end site connects the end site to an intermediate WAN domain via which the route to the other end host goes.

In order to determine the resource availability along the portion of paths within an end site LAN, the LAN-manager does the following:

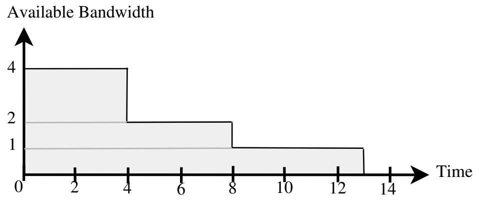
- It maintains the topology of the LAN and the resources that are available or reserved along individual hops.

```
class _segment{
    long sTime;
    long eTime;
    long bandwidth;
}
class BAG {
    _segment[] segment;
}
```

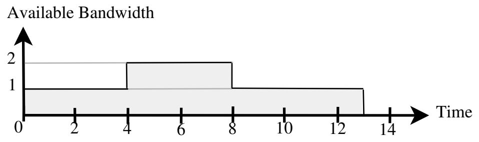
(a) Data structure for a BAG.



(b) Graphical representation for BAG-1.



(c) Graphical representation for BAG-2.



(d) Intersection of BAG-1 and BAG-2.

Figure 4: Illustration of bandwidth availability graphs for a single link between two time instances.

- It determines the paths between the end host and the border router using the stored topology information. Even in large organizations, the number of paths that packets follow from end hosts to the border router are limited to only a few (one/two in most cases).
- It uses a data structure called *bandwidth availability graph* (BAG) [21] to indicate the time varying availability of bandwidth over a certain link for a given time interval. The BAG for a link is constructed using the resource usage/availability information stored in a database. The BAG data structure is shown in Fig. 4(a), and the graphical representation of an example BAG is shown in Fig. 4(b).

To construct the BAG for a particular path, BAGs for all the hops in that path are intersected with each other. In the intersected BAG, every time instant will have the available bandwidth as the minimum of the bandwidths in all input BAGs at that instant. As an example of intersection, Fig. 4(d) shows the intersection of BAGs in Figs. 4(b) and 4(c). The BAGs for all the links in the paths between the end host and the border router are notified back to the DC that requested it.

Intermediate WANs. For an intermediate WAN domain, the path within itself connects the border routers of the two adjacent domains. The LAN-manager in this case can return the available resources using a procedure similar to what we described for an end site LAN. However, we assume that it is up to the WAN administrators on how they maintain and generate such information. Our framework in this paper does not impose any restrictions on intermediate WANs.

As our experiments in this paper are based on using Internet2 and ESnet as the intermediate WAN domains (which utilize the OSCARS system [18] for resource reservations), we explain here what kind of information is generated by the LAN-managers of these networks. These domains require applications to request bandwidth availability graph only for a certain number of paths (as opposed to the complete topology). These will be the paths that connect the two border routers. Furthermore, these domains also need the applications to specify additional criteria based on which paths should be returned (e.g., "only paths that can provide a bandwidth of ≥ 2 Gbps between time t_1 and t_2"). Without these limits and criteria, the number of potential paths can be very large, which can place unnecessary communication and processing overhead on DCs.

Another important caveat, while requesting availability information from intermediate DCs, is related to the way in which two end points are specified to these DCs. As we have noted before, the route between two end sites can traverse more than one WAN domains. Given that the network topologies for these domains are available publicly [8, 13], the LAN-managers for these domains can construct complete routes connecting the two end sites. However, no single intermediate DC can provide BAGs for the complete path interconnecting two end sites. Currently, an intermediate DC can construct BAGs only for the hops that are along the segment of the path within its domain. As a result, the local DC of an end site needs to contact the intermediate DCs one by one in order to gather the BAG for complete paths connecting two end site LANs. In the future, if and when intermediate DCs can exchange information about their resource availability, it may become possible to query a single controller to construct the BAGs along complete paths.

2.3.2 Configuring Network devices

The second task performed by the LAN-manager is to configure individual routers within the LAN in accordance with the reservation schedule given to it by the DC. Note that the actual reservation schedule is determined by the RS. The DC only passes on this schedule to the LAN-manager.

Configuring a routing device within a domain depends on the particular device and the options that the device provides for its configuration. Automated configuration is done typically via the Command Line Interface (CLI) of a device driven by software connecting to the router using the TELNET or SSH protocols. This is the way we use currently in our work to perform configuration changes. Alternatively, configuration changes can be applied through the SNMP protocol. The emerging OpenFlow [17] standard is promising way towards a unified method to configure routers of multiple different vendors. OpenFlow has been adopted by several network device vendors, including Juniper and Cisco. We plan to add OpenFlow as one of the network device managers within our LAN-manager to support OpenFlow-enabled network devices.

Table 1: An example reservation schedule.

#	Reservation St- -art Time (Hrs)	Reservation End Time (Hrs)	Reserved BW (Gb/s)
1	0900	1030	4.5
2	0430	0930	3.0
3	N/A	N/A	N/A
4	1230	1300	1.0

In our current implementation, the LAN-managers of local and remote sites utilize a combination of differentiated services (diffserv) and policy-based routing (PBR) techniques to setup the segment of a path within an end site LAN. The traffic that is allowed to enter the path is identified by an Access Control List (ACL). The diffserv configuration assigns this traffic to a high-priority class of service, typically the Expedite Forward (EF) class, and polices the bandwidth to the reserved amount. The PBR configuration forwards the traffic into an acquired layer 2 circuit within the WAN. There are some more details regarding the configuration of network devices in [14].

2.4 Request Scheduler (RS)

The Request Scheduler is the component that instills intelligence in the end-to-end network QoS system. The RS accepts a set of flexible requests (see Fig. 3) along with the resource availability (i.e., BAGs) on the relevant links between two end sites. These are the set of requests that were originally submitted by the user to the DC. The resource availability was collected by all the DCs via their LAN-managers. The local DC, to which the reservation requests were submitted, collects the availability information from all other DCs. Once the local DC has collected the resource availability as well as the reservation requests, the information is passed onto the RS. The objective of the RS is to generate a feasible network reservation schedule so that the submitted requests are satisfied according to some objective.

Table 1 shows an example reservation schedule generated by the RS for some set of requests. Request #3 was not satisfied. Note that in our current implementation, the system maintains the time in UNIX epoch form. The algorithm used by the RS to generate the reservation schedule depends on the objective in the reservation requests passed by the user. Different algorithms will result in different schedules. We now consider one such objective and design an algorithm to accomplish it in the next section.

3. JOINT OPTIMIZATION OF PATH CONSTRUCTION AND RESOURCE RESERVATION

3.1 Problem Definition

We consider a joint problem of constructing an end-to-end path and a reservation schedule for multiple flexible requests along the constructed path.

Objective. The objective is to jointly construct an end-to-end path between two end hosts and a reservation schedule along the links of that path. The constructed path and reservation schedule should maximize the number of satisfied requests while minimizing the total data transfer time.

Input. Any algorithm that solves the problem will take the following as input:

- The addresses for two end hosts.
- A set of input requests. Every input request for the algorithm consists of (i) the amount of data that needs to be transferred, (ii) the earliest start time when the data transfer can start for this request, and (iii) the deadline by which the data transfer for this request should complete.
- A set of links between two end hosts along with their BAGs. The BAGs can be different for different links.

A sample input network is shown in Fig. 1 (on page 1). BAGs for individual links are not shown.

Output. The output for a set of reservation requests consists of the following:

- The path that is chosen between the two end hosts.
- A set of start times at which the application is guaranteed to have reserved bandwidth. Every start time corresponds to a submitted request. An unsatisfied request will not have any start time.
- A set of durations for which the bandwidths will be reserved for individual requests. Each duration corresponds to a submitted request. Again, unsatisfied requests will not have any durations.
- A set of bandwidths that will be reserved for individual requests for the above mentioned durations. There is one value of bandwidth corresponding to each submitted request.

A number of questions arise from the above problem description.

First, why is flow splitting not allowed? The reason is twofold. One, it is well known that flow splitting can cause the data packets to arrive out of order at the destination due to variation in RTT along different paths. This can cause unexpected behavior in terms of bandwidth utilization [20]. Two, maintaining reservation information for single request across multiple paths can be very expensive for intermediate routers. One can explore this option as future work at the expense of memory and processing power of routers.

Second, why a single path is chosen to satisfy all the requests? The reason for this is more technical than theoretical. The circuits that can be reserved through intermediate WANs such as Internet2 and ESnet behave as virtual wires interconnecting the end site border routers with a single hop. These virtual wires appear to end site routers as Virtual Local Area Networks (VLANs), each with a specific numeric tag assigned. The number of VLAN tags is limited in number (currently 4K). Given that several users submit their own sets of requests and each path needs at least one VLAN tag, we decided to limit a set of requests to use only one VLAN tag. Note that it is easy to manipulate the system by submitting one request per set. This way each satisfied request gets a different path at the expense of VLAN tags.

Third, why the reserved bandwidth for an individual request remains constant throughout the reserved time period? Again, the reason is more technical than theoretical. Changing the bandwidth for an individual circuit requires updating configurations of all the routers along the path. This is an extremely expensive and time consuming operation. During this change, traffic may need to be assigned back to the default best effort path and it may not be possible to provide QoS guarantees.

3.2 Problem Complexity

We now present a sketch of proof which shows that our problem is an NP-hard problem.

THEOREM 1. *Joint optimization of path construction and resource reservation for multiple flexible requests between two end hosts is an NP-hard problem.*

PROOF. We first show that our problem is a generalized version of the so-called SMR^3 problem [21]. The SMR^3 problem considers that a path and the BAG for this path between two end hosts is given. It then tries to accommodate multiple flexible resource reservation requests (as shown in Fig. 3) along that path. The objective of SMR^3 is to construct a reservation schedule that will accommodate as many reservation requests as possible while minimizing the time required to perform a data transfer. Therefore, SMR^3 is a special instance of our problem where we have exactly one path between two end hosts.

Furthermore, in [21], it was shown that an NP-hard variation of the Generalized Assignment Problem [6] can be converted to SMR^3 in polynomial time, thereby proving that SMR^3 is also an NP-hard problem. Since an NP-hard problem (i.e., SMR^3) is a special instance of our problem, our problem is at least NP-hard. □

Given the NP-hardness of our problem, we cannot develop an optimal polynomial time solution procedure unless $P = NP$. As a result, we will develop an efficient polynomial time heuristic that constructs effective solutions for our problem.

3.3 Resource Reservation and Path Construction (RRPC) Algorithm

In this section, we are going to develop a polynomial time algorithm, called RRPC, to solve the joint problem of path construction and resource reservation. The objective is to maximize the number of satisfied requests while minimizing the total data transfer time. We first provide an overview of the RRPC algorithm, followed by a detailed description and complexity analysis.

3.3.1 Basic Idea

Our RRPC algorithm follows a procedure similar to Dijkstra's shortest path algorithm. However, instead on constructing a shortest hop (or shortest distance) route between the two end hosts, the RRPC algorithms aims to construct a route that can provide the best solution to our problem.

The RRPC algorithms begins by considering the source end host as the current node. Connecting the current node with each neighboring node will provide partial paths (one path corresponding to each neighboring node). The RRPC algorithm then calculates a solution along each of these partial paths that have their end nodes as the neighboring nodes. All the current solutions along with their corresponding partial paths are stored. The end node with the best current solution becomes the current node, and the above procedure is repeated again. The procedure stops when the destination end host becomes the current node.

In addition to the detailed description, there are two fundamental questions that we have not described in the above overview of the algorithm.

1. How to construct the solution for a given (partial) path?
2. How to determine if one solution is better than the other? That is, how can two solutions be compared?

We answer the above questions before we present the detailed description of our algorithm.

3.3.2 Creating a Solution for a Partial Path

A solution along a partial path implies a set of time varying bandwidth reservations along all the hops in that path. As per our objective, these reservations should be made in a way that can accommodate as many reservation requests as possible while minimizing the total data transfer time.

The first step towards creating such a solution is to construct a BAG for the given path. This can be achieved by intersecting the BAGs for all the hops in the path. These individual BAGs are part of the input to our algorithm. This intersection will give us a single BAG that represents the time varying availability of bandwidth along the given path. We have developed a linear time algorithm, called BAGSECT (BAG-interSECTion), to perform the intersection of two BAGs. The BAGSECT algorithm can be described as follows:

BAGSECT. We consider that the input BAGs to the algorithm are well formed. That is, the segments within the BAGs are stored in the increasing order of start time values, and the individual segments do not overlap in time domain. The algorithm begins by considering the start times of the first segment from two BAGs; bag_1 and bag_2. As these start times will be same[2], the start time of the first segment in the merged BAG (denoted by mBAG) will also be same. The bandwidth in the first segment of mBAG will be equal to the minimum of the bandwidths in the first two segments of bag_1 and bag_2. The algorithm will then move on to the next segment of both bag_1 and bag_2, as well as to the next segment of mBAG.

The start time of the current segment of mBAG will be made equal to the minimum of the start times of current segments of bag_1 and bag_2. Among the two current segments of bag_1 and bag_2, the BAGSECT algorithm will next consider the bandwidth of the segment with earlier start time. This bandwidth is denoted by bw_{early}. The algorithm then uses bw_{prev} to denote the bandwidth of the previous segment of the segment with later start time. The bandwidth of the current segment of mBAG will be made equal to the minimum of bw_{early} and bw_{prev}. The algorithm then increments the current segment for the mBAG and the current segment for the BAG corresponding to bw_{early}. It then repeats the procedure in the previous paragraph.

The algorithm stops its iterations when it has iterated over all the segments from both BAGs. It can be easily shown that BAGSECT is an optimal algorithm. However, we do not include such proof here. Pseudocode of the BAGSECT algorithm is omitted due to pare limitations.

It is possible that the intersected BAG has contiguous segments with equal bandwidth values. If needed, such segments can be merged by going over the BAG in one pass.

Once the final intersected BAG of all the hops along the path is obtained, the next (or the final) step is to accommodate as many requests as possible in this BAG while minimizing the total data transfer time. As shown in [21], this problem is also an NP-hard problem. In [21], we have developed an efficient polynomial time heuristic called resource reservation algorithm (RRA) to accommodate multiple requests within a BAG with same objective as that of RRPC. As a result, instead of developing a new heuristic, we plan to use RRA in our system implementation to solve this sub-problem. Note that the design of our RRPC algorithm can accommodate any alternate sub-procedures to create a solution for partial path, and is not limited to using RRA. Any sub-procedure other than RRA may not be suitable for the joint path construction and resource reservation problem considered here.

3.3.3 Comparing Two Solutions

Our objective is to generate a solution that accommodates largest number of reservation requests while minimizing the data transfer time. However, by looking at the structure of possible solutions, we can observe that there can be multiple different solutions that satisfy such an objective. As an example, Fig. 5 shows two optimal solutions that accommodate three reservation requests and complete the data transfer in same amount of time. However, the first solution enables the transfer of 250 GB data, while the second solution enables the transfer of 300 GB data. From a theoretical perspective, both of these solutions are optimal. However, in real world systems, one would definitely prefer the second solution. We can now see that there are three fundamental building blocks of each solution: (i) the number of requests that are satisfied, (ii) the time taken to complete the requests, and (ii) the amount of data transferred. For a

[2]Although not possible in our settings, in the case start times are not same, then we can make them same by adding a segment with zero bandwidth to the beginning of the BAG with later start time.

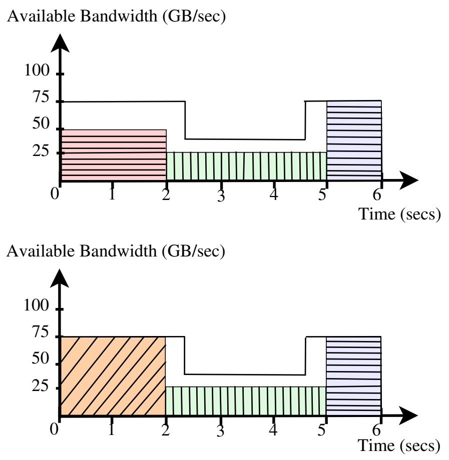


Figure 5: Two solution with same number of requests and same amount of data transfer times.

particular solution, we define the ratio of "total data transferred in the satisfied requests" to "the time taken to transfer that data" as the effective bandwidth utilization (EBU). We consider a solution to be superior to another solution if it satisfies larger number of requests. If the number of satisfied requests is the same, then the solution with shorter transfer time will be considered superior. Finally, if the transfer times are also the same, then the solution with larger EBU is considered to be superior. If both solutions accommodate the same number of requests, have the same transfer time, and the same EBU, then both solutions are considered to be equally effective. Note that it is possible that a different criteria for solution comparison may be appropriate in some other scenario. For example, someone may prefer EBU over number of satisfied requests. In that case, the design of our RRPC algorithm is flexible enough to incorporate the new user defined criteria.

3.3.4 *Detailed Description of the RRPC Algorithm*

The RRPC algorithm runs in iterations. To begin with, every node in the network stores a path, a BAG, and a solution that accommodates zero requests (i.e., the worst possible solution). The stored path at a node represents the current best path from the source node to this node, and is set to null in the beginning. The BAG stored at a node represents the intersection of BAGs along the hops of the stored path, and is also set to null in the beginning. The stored solution represents the solution obtained if the reservations are scheduled along the stored path (i.e., solution along the partial path for nodes that are not destination nodes). The algorithm then starts the first iteration by considering the source node as the current node, and performs the following steps:

1. It considers the path stored at the current node, and considers all the paths that extend the stored path to the neighboring nodes.
2. For every new extended path considered, the algorithm calculates the effective BAG along that path, and calculates the solution. The new solution obtained is compared with the one that is already stored at the corresponding neighboring node. If the new solution is better, then the new solution replaces the stored solution and the stored path in the corresponding neighboring node.
3. The current node is marked as *visited*.
4. Among all the nodes that are not visited, the one that has the best solution becomes the current node.
5. If the current node is the destination node, then the RRPC algorithm stops. The path and the solution stored in the destination node is the final solution.
6. If the current node is not the destination node, then all the steps (from Step 1 to Step 6) are repeated for this current node.

There are some observations that we can make regarding the RRPC algorithm.

First, whenever we increase the length of a path by one node, the new solution on this longer path cannot be better. This can be easily proven by using the fact that intersection of two BAGs can never produce a superior BAG. That is, at every time instant, the intersected BAG will have the available bandwidth that is not larger than the available bandwidth in the two intersecting BAGs. As a result, if the solutions being constructed for partial path are optimal, then our RRPC algorithm will construct an optimal solution. This statement may not seem obvious. However, the proof can follow the same token as the proof of optimality of Dijkstra's algorithm on the graphs with non negative weights. Also note, that creating a solution for a partial path is an NP-hard problem. Therefore, it is not possible to have an optimal algorithm with polynomial running time unless $P = NP$.

Second, instead of saving the complete path from source node to a particular node, that particular node may only save its predecessor.

3.3.5 *Runtime Complexity*

The RRPC algorithm never traverses a node already marked as "visited", and each node is marked as "visited" only once. This results in every hop in the network being considered only once. As a result, the number of hops being traversed in worst case is $O(|\mathcal{V}|^2)$, where $\mathcal{V}$ is the set of all vertices/nodes in the network. Next, whenever a hop is considered, RRPC (i) intersect two BAGs using the BAGSECT algorithm, and (ii) constructs a solution using RRA.

For (i), the running time of BAGSECT algorithm is $O(M_{b_i} + M_{b_j})$, where M_{b_i} and M_{b_j} are the number of segments in bags b_i and b_j respectively.

For (ii), the running time of RRA is $O(N^3 + N^2M)$ [21], where N is the number of flexible requests that needs to be accommodated in a bandwidth availability graph with M steps.

In order to identify the next node to be visited, the RRA algorithm has to select the best node corresponding to the $O(|\mathcal{V}|)$ stored solutions. The selection can be accomplished in $O(|\mathcal{V}|)$ time. As the number of nodes to be visited is also limited to $O(|\mathcal{V}|)$, the total time for identifying next node during all iterations will be $O(|\mathcal{V}|^2)$ in worst case. Note that

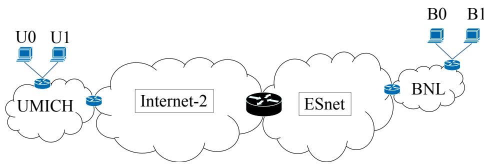


Figure 6: Network connectivity.

Table 2: Individual requests between three pairs of end sites. These requests were submitted one by one.

Req-uest#	Soure	Desti--nation	Volu--me (GB)	Start Time (Hrs)	Dead--line (Hrs)	BW Limit (Mb/s)
1	B1	U1	200	14:41	15:31	800
2	U1	B0	100	14:46	15:36	400
3	B0	U0	160	14:51	15:41	600

we can do better than $O(|\mathcal{V}|^2)$, however, it does not affect the overall running time as we see next.

The total running time of the RRPC algorithm can now be written as $O(|\mathcal{V}|^2 \cdot (N^3 + N^2M + M_{b_i} + M_{b_j}) + |\mathcal{V}|^2)$. Whenever two BAGs, b_i and b_j, with M_{b_i} and M_{b_j} steps are intersected, the resulting BAG cannot have more than $M_{b_i} + M_{b_j}$ steps. As a result, in the worst case, the size of each BAG at any step in the RRPC algorithm cannot exceed $\sum_{b_i \in \mathcal{B}} M_{b_i}$, where $\mathcal{B}$ is the set of BAGs along all hops in the network.

As a result, the final expression for the worst case running time of the RRPC algorithm is

$$O\left(|\mathcal{V}|^2 \cdot \left[N^3 + N^2 \cdot \sum_{b_i \in \mathcal{B}} M_{b_i}\right]\right).$$

We find that in real world experiments, the worst case running time is never reached, and the RRPC algorithm runs extremely efficiently.

4. EXPERIMENTS

4.1 Network Setup

We ran our experiments between two geographically distant sites connected via two intermediate WAN domains. Figure 6 shows the connectivity of our network. The twp end sites are Brookhaven National Laboratory (BNL) and University of Michigan (UMICH). BNL is connected to ESnet, and UMICH is connected to Internet-2. ESnet and Internet-2 are connected to each other via several peering points.

We have omitted the network topology of ESnet and Internet-2 as they are rather large and can be obtained online [8, 13]. We have deliberately omitted the LAN topology of end sites as they are productions networks (as opposed to testbeds), and their topologies cannot be disclosed publicly.

In Section 4.2, we will show the working of our QoS system, and in Section 4.3, we will show the working of our RRPC algorithm.

Table 3: Reservation schedules for individual requests in Table 2.

Request #	Soure	Desti--nation	Start Time (Hrs)	End Time (Hrs)	Reserved BW (Mb/s)
1	B1	U1	14:41	15:21	800
2	U1	B0	14:46	15:26	400
3	B0	U0	14:51	15:34	600

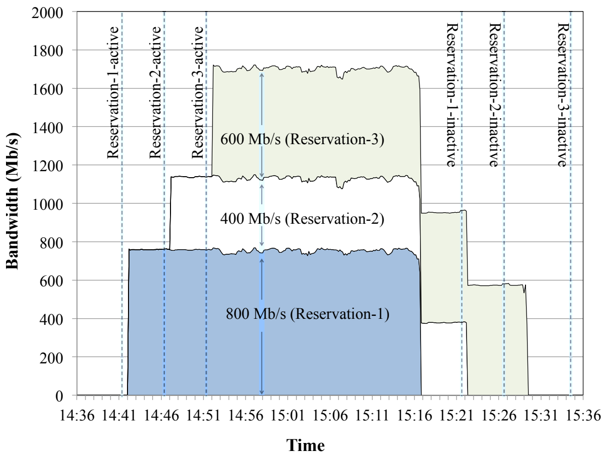


Figure 7: Stacked area graph showing the bandwidth usage for three data transfers.

4.2 Single Request

Our goal in this section is to demonstrate the operation of bandwidth reservations within our QoS system. In this section, we will limit the number of requests submitted to the DC of an end site to only one at a time. We will then show the effect of bandwidth reservations on the data transfers involving large files.

We consider two end sites, BNL and UMICH, between which the data transfers will happen. We then consider three pairs of end hosts between which the individual files will be transferred. The pairs and the submitted requests are shown in Table 2. The first two columns in Table 2 show the source and destination of the submitted request. The third column shows the volume of data that needs to be transferred. The fourth column shows the start time, i.e., the time when the data will become available for transfer. The fifth column shows the time by which the request would like the data transfer to finish. Note that in our implementation, we use UNIX epochs for storing time values. The last column shows the maximum bandwidth that the request can use.

Source B1 first submits its first request to the DC of BNL. Source U1 then submits its request to the DC of UMICH. Source Bo then submits its second request to the DC of BNL. As there was enough bandwidth available within the end sites and the intermediate WAN domains, all the three requests were satisfied. We are allowed to reserve a maximum of 2 Gb/s between BNL and UMICH at any given time. Table 3 shows the amount of bandwidths and the durations for which these bandwidths were reserved for each request. We can now make few observations and explain some of our choices.

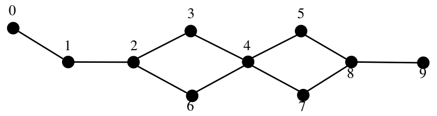


Figure 8: Network topology.

First, note that in the Table 2, we omitted the port range that was submitted along with each request. For the above transfers, each data transfer request is between three different source and destination pairs. As such, we reserved the bandwidth for all the ports, i.e., the requests were submitted for ports (1 − 65535). Routers can distinguish between individual flows based on their source and destination ip-addresses. A second reason for reserving all the ports is the data transfer tool that we have used, which is GridFTP [10]. GridFTP uses a wide range of ports for performing data transfers (as opposed to single standard port for scp or ftp).

Second, why have we chosen GridFTP as the data transfer tool? In our experiments, we were unable to scale scp or ftp to use the large amount of bandwidth that we were reserving for individual data transfers over long distances. We believe the reason is the inability of a single TCP flow to scale to large bandwidth values when the RTT becomes large. As shown in our experiments, gridFTP can perform data transfers using multiple TCP flows in parallel, and thus can utilize large bandwidths much more efficiently. This brings out another important discussion on the ability of data transfer applications or transport protocols to utilize the reserved bandwidth. However, we consider this discussion out of scope for this paper.

Third, from Table 3, one can observe that the amount of time for which the bandwidth is reserved is slightly more than what is required (approximately 20% larger). The reason we are doing this is the overhead of the gridFTP data transfer tool. We observed that GridFTP needs some additional bandwidth to transfer its meta data related to multiple parallel TCP streams.

Figure 7 shows a stacked area graph for the amount of bandwidth used by all three data transfers. The x-axis shows the time, and the y-axis shows the amount of bandwidth. There are three bands within Fig. 7. The bottom band shows the instantaneous bandwidth used by the data transfer corresponding to request #1. It is using a bandwidth of 800 Mb/s, and transferring a 200 GB file. We can see that the amount of used bandwidth does not remain constant due to the use of TCP as the underlying transport protocol. We can also observe that due to the use of multiple TCP streams in GridFTP, it can start utilizing the available bandwidth almost instantaneously. Furthermore, it seems that the bandwidth usage is a little below 800 Mb/s. This is because of the overhead of metadata used within GridFTP. Similarly, the middle band corresponds to request #2, uses 400 Mb/s, and transfers a 100 GB file. The top band corresponds to request #3, uses 600 Mb/s, and transfers a 160 GB file. We can see that all the three transfers are using the correct bandwidth that was reserved for them.

4.3 Multiple Requests (RRPC Algorithm)

Now that we have shown the successful reservation and utilization of bandwidth for individual flows, our goal in this section is to show the operation of the RRPC algorithm. We consider an example network shown in Fig. 8, and seven reservation requests shown in Table 4. The source and destination for these requests could be different. All the requests will be submitted at the same time to the DC. Figures 10(a)-10(k) shows the BAGs of all the links in the network shown in Fig. 8. Note that different links can have different BAGS because there could be multiple users with different reservations at different times.

Figure 9 shows the path constructed by the RRPC algorithm, and Fig. 10(l) shows the BAG along this path. Figure 10(l) also shows how the requests fit into the BAG of the constructed path. Table 5 shows the schedule constructed for the submitted requests by RRPC. It was not possible to satisfy requests #2 and #5. We can now make end-to-end reservations according to the generated output schedule.

Table 4: Requests beween nodes 0 and 9.

#	Start Time (secs)	Deadline (secs)	Volume (Gb)	BW Limit (Gb/s)
0	0	4	12	8
1	4	10	12	6
2	3	7	18	6
3	2	5	16	8
4	9	14	16	10
5	8	11	12	5
6	4	9	15	5

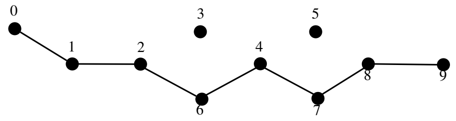


Figure 9: Path constructed using the RRPC algorithm.

Table 5: Output of the RRPC algorithm.

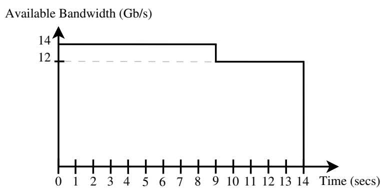
#	Reservation Start Time (secs)	End Time (secs)	Reserved BW (Gb/s)
0	0	2	6
1	4	10	2
2	N/A	N/A	N/A
3	3	5	8
4	10	12.67	6
5	N/A	N/A	N/A
6	5	8	5

5. RELATED WORK

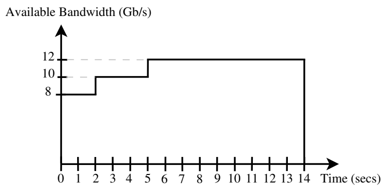
There has been a large amount of experimental as well as theoretical research performed in the area of network QoS. Here we discuss some of the work which is most relevant to this paper.

In [14], Katramatos *et al.* presented a tool called Terapaths that can be used to make end-to-end bandwidth reservations across multiple connected domains. Terapaths, however, can only reserve bandwidth for fixed non-flexible requests; one request at a time. In contrast, one of the important goals of the work presented in this paper is to be able to intelligently schedule multiple and flexible requests.

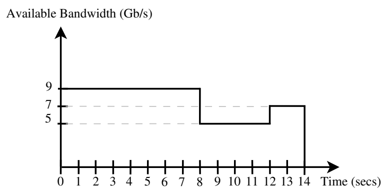
In [1], Ayyangar *et al.* proposed extensions to the RSVP-TE protocol [2] that describes how end-to-end label switched paths (LSPs) can be established from individual LSP seg-



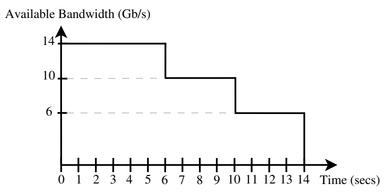
(a) BAG for link $0-1$.



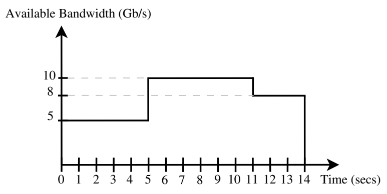
(b) BAG for link $1-2$.



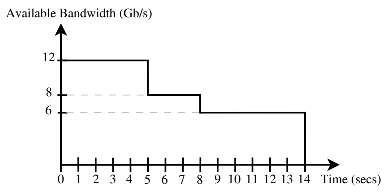
(c) BAG for link $2-3$.



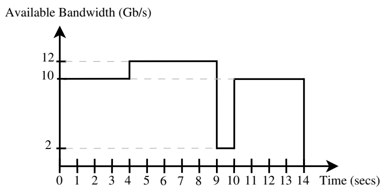
(d) BAG for link $2-6$.



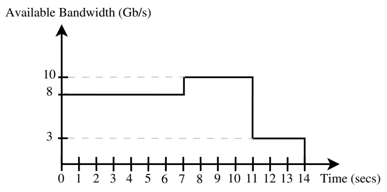
(e) BAG for link $3-4$.



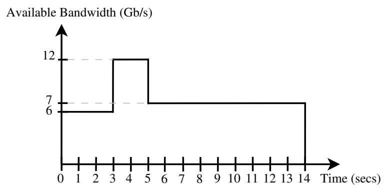
(f) BAG for link $4-5$.



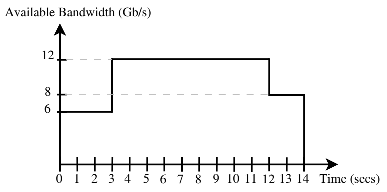
(g) BAG for link $4-7$.



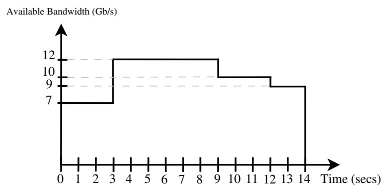
(h) BAG for link $5-8$.



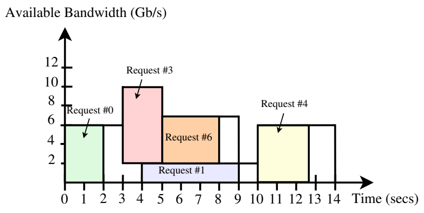
(i) BAG for link $6-4$.



(j) BAG for link $7-8$.



(k) BAG for link $8-9$.



(l) Final BAG on the constructed path shown in Fig. 9 along with the accommodated requests.

Figure 10: BAGs for the links within the network.

ments. However, such an extension is incapable of and does not aim to determine the appropriate amount of bandwidth to reserve for multiple flexible requests. In other words, their proposed protocol does not have the intelligent QoS component that we develop in this paper.

MPLS traffic engineering (TE) software [15] by CISCO enables traffic engineering only on MPLS enabled networks. It uses constraint-based-routing and uses RSVP to establish MPLS tunnels across the backbone. In contrast to MPLS-TE, our architecture can work with multiple WANs employing heterogenous QoS enabling technologies. As an example, in our experimental results, Diffserv and PBR were used at end site LANs whereas ESnet and Internet2 employed MPLS within WANs. Furthermore, MPLS traffic engineering cannot accommodate the type of deadline driven flexible requests that we consider in our work.

In [17], McKeown *et al.* proposed an standard called OpenFlow that aims to provide an interface to update routing tables within routers and switches. Once a router is OpenFlow enabled, one can use OpenFlow drivers to update routing tables. In [7], Curtis *et al.* proposed another architecture, called DevoFlow, as a modification to the OpenFlow. DevoFlow was shown to perform better than OpenFlow on high-performance networks. OpenFlow and DevoFlow, however, are not available on most commercial routers. As an example, all the routers used in our experiments do not support either of these. As a result, we developed the router configuration drivers by using the API provided by the router vendors. Once OpenFlow or DevoFlow are adopted by major router vendors, we can easily add them as one of the component within our LAN-manager to configure routers.

In [22], Sherwood *et al.* used OpenFlow as the underlying mechanism, and introduced FlowVisor that can be used to reserve the desired resources (e.g., bandwidth) within real world networks. In contrast to our work, the bandwidth reservations in their deployed system had to be made manually via request submissions to a network administrator. Furthermore, the FlowVisor does not aim to intelligently handle multiple and flexible resource reservation requests as we do in our work.

In [23], Wilson *et al.* developed a deadline aware delivery control protocol called D^3. In contrast to our work, D^3

requires explicit modifications to the routers within the network. Furthermore, D^3 works within data center networks and proposes modifications to the underlying transport layer protocol. In contrast, our QoS architecture works across WANs and can accommodate any underlying transport layer protocol.

NetStitcher [16] is another system that proposes to use left-over bandwidth at different times to transfer data between data-centers at multiple locations. However, the performance of NetSticher depends on an imperfect knowledge of traffic conditions. Our proposed intelligent QoS architecture maintains this required bandwidth knowledge, and can perform efficient data transfers between different sites. As a result, NetSticher could use our proposed system to better manage and schedule the left-over bandwidths.

In [21], Sharma *et al.* developed an algorithm to schedule multiple and flexible resource reservation requests. However, this algorithm works only when the source and destination are connected via exactly one path, and cannot work for a general network topology. Furthermore, in [21], authors present simulation results and do not deploy their algorithm on real networks.

There are few other theoretical works (see e.g., [11, 12, 19]) that are similar to [21] and consider the scheduling of flexible resource reservation requests. For the general overview of QoS based routing, we refer readers to a comprehensive survey in [24].

6. CONCLUSION

We have presented the design and implementation of an intelligent QoS system. The system can guarantee end-to-end network QoS for multiple and flexible resource reservation requests. A major component of the developed system is a polynomial time scheduling algorithm called RRPC. The RRPC algorithm performs a joint optimization of path construction and bandwidth reservation for multiple and flexible resource reservation requests. We showed such an optimization problem to be NP-hard. We implemented the intelligent QoS system on real hardware serving real world production networks. Our intelligent QoS system do not require any modifications or new software installation on the routers within the network.

7. REFERENCES

[1] A. Ayyangar, K. Kompella, J.P. Vasseur, and A. Farrel, "Label Switched Path Stiching with GMPLS TE," *RFC 5150,* 2008.

[2] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP tunnels," *RFC 3209,* 2001.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated service," *RFC 2475,* 1998.

[4] R. Braden, D. Clark, and S. Shenker, "Integrated services in the Internet architecture: An overview," *RFC 1633,* 1994.

[5] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP)," *RFC 2205,* 1997.

[6] C. Chekuri and S. Khanna, "A PTAS for the multiple knapsack problem," In *Proc. ACM-SIAM SODA,* pp. 213–222, Philadelphia, PA, USA, 2000.

[7] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," In *Proc. ACM SIGCOMM,* Toronto, Canada, August 15–19, 2011.

[8] "ESnet Topology," http://www.es.net/network/

[9] "The GEANT2 network," http://www.geant2.net/

[10] "GridFTP Documentation," http://globus.org/tool-kit/docs/3.2/g-ridftp/

[11] J. Gu, D. Katramatos, X. Liu, V. Natarajan, A. Shoshani, A. Sim, D. Yu, S. Bradley, and S. McKee, "StorNet: Co-scheduling of end-to-end bandwidth reservation on storage and network systems for high-performance data transfers," in *Proc. IEEE INFOCOM, Workshop on High-Speed Networks,* Shanghai, China, April 10–15, 2011.

[12] R.A. Guerin and A. Orda, "Networks with advance reservations: The routing perspective," in *Proc. IEEE INFOCOM,* pp. 118–127, Tel-Aviv, Israel, March 26–30, 2000

[13] "Internet 2–Topology," http://www.internet2.edu/o-bservatory/archive/data-collections.html#topology

[14] D. Katramatos, D. Yu, K. Shroff, S. McKee, and T. Robertazzi, "TeraPaths: End-to-end network resource scheduling in high-impact network domains," *Intl. Journal On Advances in Internet Technology,* vol 3, no. 1–2, pp. 104–117, 2010.

[15] "MPLS traffic engineering," http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/TE_1208S.html

[16] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with NetStitcher," In *Proc. ACM SIGCOMM,* Toronto, Canada, August 15–19, 2011.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* vol 38, no. 2, pp. 69–74, March 2008.

[18] "OSCARS: On-Demand Secure Circuits and Advance Reservation System," http://code.google.co-m/p/oscars-idc/

[19] K. Rajah, S. Ranka, and Y. Xia, "Advance reservation and scheduling for bulk transfers in research networks," to appear *IEEE Transactions on Parallel and Distributed Systems.*

[20] S. Sharma, D. Gillies, and W. Feng, "On the goodput of TCP NewReno in mobile networks," In *Proc. International Conference on Computer Communications and Networks (ICCCN),* Zurich, Switzerland, August 2–5, 2010.

[21] S. Sharma, D. Katramatos, and D. Yu, "End-to-end network QoS via scheduling of flexible resource reservation requests," To appear *ACM/IEEE Supercomputing Conference (SC),* Seattle, WA, November, 12–18, 2011.

[22] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?," In *Proc. USENIX OSDI,* Vancouver, Canada, October 4–6, 2010.

[23] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, "Better never than late: Meeting deadlines in datacenter networks," In *Proc. ACM SIGCOMM,* Toronto, Canada, August 15–19, 2011.

[24] O. Younis and S. Fahmy, "Constraint-based routing in the internet: Basic principles and recent research," *IEEE Communications Surveys & Tutorials,* vol. 5, no. 1, pp. 2–13, 2003.