-

BNL-95022-2011-CP

End-to-End Network QoS via Scheduling of Flexible Reservation Requests

Sushant Sharma, Dimitrios Katramatos, Dantong Yu
Computational Science Center, Brookhaven National Laboratory,
Upton, NY 11973

To be presented at the International Conference for High Performance Computing, Networking, Storage, and Analysis
Seattle, Washington
November 14-17, 2011

April 2011

Computational Science Center

Brookhaven National Laboratory
P.O. Box 5000
Upton, NY 11973-5000
www.bnl.gov

DISCLAIMER

End-to-End Network QoS via Scheduling of Flexible Resource Reservation Requests

Sushant Sharma Dimitrios Katramatos Dantong Yu
Computational Science Center
Brookhaven National Laboratory, Upton, NY 11705.

ABSTRACT

Modern data-intensive applications move vast amounts of data between multiple locations around the world. To enable predictable and reliable data transfer, next generation networks allow such applications to reserve network resources for exclusive use. In this paper, we solve an important problem (called SMR^3) to accommodate multiple and concurrent network reservation requests between a pair of end-sites. Given the varying availability of bandwidth within the network, our goal is to accommodate as many reservation requests as possible while minimizing the total time needed to complete the data transfers. We first prove that SMR^3 is an NP-hard problem. Then we solve it by developing a polynomial-time heuristic, called RRA. The RRA algorithm hinges on an efficient mechanism to accommodate large number of requests by minimizing the bandwidth wastage. Finally, via numerical results, we show that RRA constructs schedules that accommodate significantly larger number of requests compared to other, seemingly efficient, heuristics.

Keywords

Scheduling, Resource Reservation, End-to-end QoS

1. INTRODUCTION

Extreme scale scientific computations within collaborative environments are highly dependent on the availability of data that they need to process. The data in such environments is usually distributed among national and international data repositories. As a result, any scientific data analysis requires frequent and time-sensitive transfers of large volumes of data from one repository to another over the network. This fact highlights the extreme importance of reliable network services. However, the default behavior of today's best effort networks is to treat all data flows equally. This causes the data flows of higher priority and/or urgency to be adversely impacted by competing flows of lower priority. In distributed data-intensive environments, such behavior is unwarranted and can degrade the effective "goodput" of the overall system. Furthermore, such behavior makes it impossible to guarantee any type of Quality of Service (QoS) which is often required for time sensitive data transfers.

As a first step to alleviate such concerns, next generation networks such as EsNet [5] and Internet2 [9] have dedicated large bandwidth links between sites distributed across distant geographical regions. These networks allow applications to reserve network resources for the exclusive use between end sites. The ability to reserve network resources is enabled via inter domain controllers (IDCs) of such networks. An example of IDC is the on-demand secure circuits and advance reservation system (OSCARS) within EsNet.

As a next step in enabling QoS guarantees, end site applications that can intelligently reserve network resources are needed. Such applications communicate with IDCs and reserve the required resources. The design of such applications can vary depending upon the kind of reservation requests the IDCs can accept, and the flexibility of the reservations needed between end sites. Terapaths [10] is one example of such network reservation tools that can communicate with the IDCs of EsNet and Internet2 to reserve network resources.

However, there are two major shortcoming of the existing network reservation tools that we address in this paper. First, the inability of existing tools to intelligently schedule reservations in the presence of multiple/concurrent reservation requests. The current mechanism to handle multiple reservation requests is to try and reserve the requests in the order in which they are generated. As we show in this paper, in the presence of multiple reservation requests, we can significantly improve the performance if we schedule the requests intelligently. We have developed an algorithm that can construct an efficient schedule for multiple/concurrent requests between a pair of end sites.

Second, we address the inability of existing tools to exploit the flexibility in reservation requests. In our experience of communicating with the end users of such systems, we have determined that the reservation requests can have some flexibility. As an example, instead of requiring a fixed amount of bandwidth, a data transfer may have an upper limit on how much bandwidth it can use. Furthermore, as opposed to having a rigid start time, it may be acceptable as long as a data transfer gets finished before a certain deadline. However, given the presence of such flexibilities, existing reservation tools still submit a fixed request to the IDCs. Existing tools usually try to reserve the maximum bandwidth that the end sites can support. The IDC of the intermediate network may not be able to support the maximum bandwidth, but it may easily finish the data transfer using a lower bandwidth before the deadline. As a result, the IDC will have to reject the request to reserve the maximum amount of bandwidth. In the presence of limited feedback from IDCs, the existing reservation tools then modify and resubmit the modified request to IDCs. It may require multiple rounds

of submissions before the request is modified in a way that is acceptable to IDC. Our work in this paper specifically address such inefficient mechanism to reserve resources. To address such inefficiencies, we allow reservation tools/applications to ask the IDC for the available resources. IDCs of the next generation networks will have the ability to provide such availability to the applications on request [11]. Once such availability information is available, it becomes possible for reservation tools to intelligently schedule the requests. The algorithm developed by us in this paper aims to construct such intelligent schedules. As a result, the IDCs can now accommodate all the submitted requests, and there is no need for modifying and re-submitting the requests one by one to IDCs.

We plan to integrate the developed algorithm into our existing network reservation tool called TeraPaths [10]. Our results show that the developed algorithm can significantly improve the overall system performance.

In Section 2, we describe our problem in detail and the precise objective that we want to achieve. Section 3 shows the proof of NP-hardness for our problem. In Section 4, we provide the description of the algorithm that we have developed, and Section 5 presents the simulation results. Section 6 discuss some related work, and Section 7 concludes the paper.

2. PROBLEM DESCRIPTION

In this section, we give the description of our problem using a motivational example.

Figure 1 shows a simple scenario where the network can reserve a bandwidth of 10 Gb/s between two end sites. However, the rate at which the data can be read from/written to the storage device on an end host is limited to 8 Gb/s. In this scenario, it is advisable to reserve a maximum of 8 Gb/s of bandwidth along the network path. A single reservation request between two end sites includes (i) the maximum bandwidth that the end sites can support (e.g., 8 Gb/s in the above example), (ii) the amount of data that needs to be transferred, and (iii) the start and end times between which the end sites needs the data transfer to begin and finish. As an example, the time varying bandwidth availability of the network between the two end sites of Fig. 1 is shown in Fig. 2. The bandwidth availability graph is a step function, with different available bandwidth between different time intervals (or steps). For Fig. 2, the available end-to-end bandwidth for time interval $[1-3]$ is 4 Gb/s, and for the interval $[3-7]$, the available bandwidth becomes 8 Gb/s (say due to termination of some other network flows within the network). Now assume that end site 1 wants to make two reservations between itself and end site 2. Both of these reservations can support the maximum bandwidth of 8 Gb/s. Assume that the start and end times for both the reservations is 1 and 7 secs respectively. Furthermore, assume that both reservations need to transfer 16 Gb of data. Once a reservation is confirmed for a request, the allocated bandwidth for that reservation remains fixed for its duration. Given the bandwidth availability information, and the reservation requests, the objective is to schedule the data transfers if possible. The schedule should be able to finish the data transfers for all reservations within the requested time window and at the same time optimize a certain objective function. Without an objective function, there could be multiple feasible schedules that will accommodate the requests. Figure 3 shows two feasible schedules for the above example. We can see that the schedule in Fig. 3(a) is optimal if we want to finish the data transfers as early as possible, and the schedule in Fig. 3(b) is optimal if we want to minimize the total data transfer time.

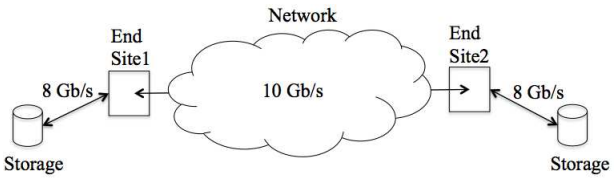


Figure 1: An example scenario.

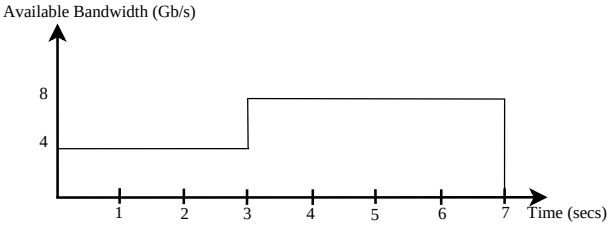
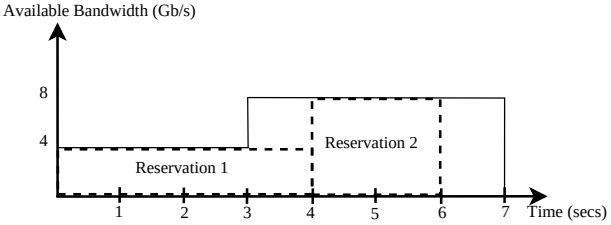
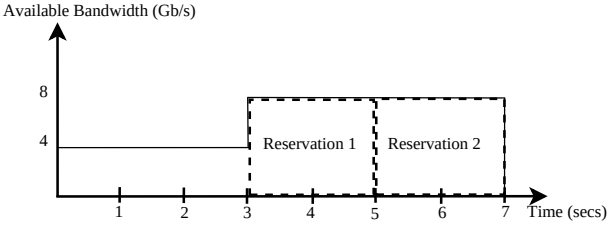


Figure 2: Bandwidth availability graph.



(a) Solution 1.



(b) Solution 2.

Figure 3: Two feasible schedules.

From the above example, it is easy to see that finding a feasible and optimal schedule becomes much more challenging if the bandwidth availability graph contains many more variations. Furthermore, the problem becomes even more challenging if we introduce the following four generalizations: (i) the requested start and end times for different reservations can be different, (ii) the amount of data that needs to be transferred for every reservation can also be different, (iii) the maximum bandwidth that a particular reservation can use may vary from one reservation to another, and (iv) there are multiple reservation requests that need to be scheduled.

In this paper, we consider all these generalizations and solve the scheduling problem with two goals: (i) accommodate as many reservation requests as possible (all the requests in best case), and (ii) minimize the sum of total times that is required to transfer data for all accommodated requests. We name our problem as SMR^3 (Scheduling Multiple Resource Reservation Requests).

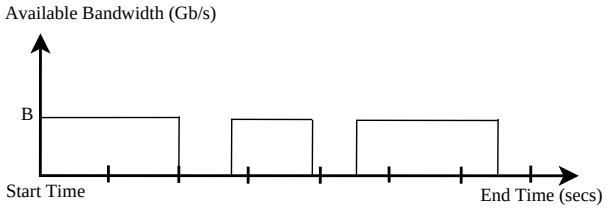


Figure 4: Piecewise bandwidth availability graph in *simple*-SMR^3.

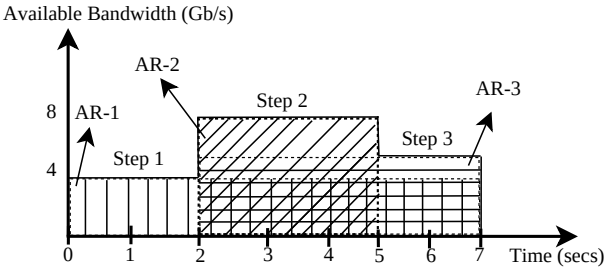


Figure 5: An example showing accommodating regions.

3. PROBLEM COMPLEXITY

In this section, we prove the NP-hardness of our SMR^3 problem. We consider a known NP-hard variation of the generalized assignment problem (GAP), and convert an instance of this GAP into a special instance of our problem in polynomial time. We first describe the special instance of our SMR^3 problem, which we call *simple*-SMR^3.

In *simple*-SMR^3, all reservation requests have same start and end times, same values for the maximum bandwidth that they can use, but different amounts of data that they need to transfer. The bandwidth availability curve in *simple*-SMR^3 is a piecewise linear function. An example of this is shown in Fig. 4. The available bandwidth during all intervals is exactly the same as the maximum bandwidth requested in the individual reservation requests. However, the time duration of every interval in which the bandwidth is available can be different. We can make two important observations from the definition of *simple*-SMR^3. First, in any solution, the time taken to complete the data transfer for a particular request will be same irrespective of the interval in which it is scheduled. This is because all intervals have the same amount of bandwidth available for every request. As a consequence, every feasible schedule will have the same total time taken to transfer data for all reservation requests. Second, for the *simple*-SMR^3 problem, the objective of finding a feasible schedule with minimum data transfer time is now reduced to finding just a feasible schedule. We now have the following lemma.

LEMMA 1. *The* simple-*SMR^3 problem is an NP-hard problem.*

PROOF. We consider the following variation of GAP that is known to be NP-hard [3]:

Given n bins, m items, the capacity of each bin, and the size of each item: the goal is to determine a feasible assignment of items to bins such that the sum of the size of items in each bin does not exceed the bin's capacity.

An instance of the above problem can be converted into the *simple*-SMR^3 problem by considering the bins as the intervals in the piecewise bandwidth availability graph. The capacity of each bin can be considered as the area under each interval where the bandwidth is available. Each item can be considered as the bandwidth reservation request, where the size of each item is equivalent to the amount of data that needs to be transferred for that request. Finally, the goal of finding a feasible assignment of items to bins can be considered equivalent to finding a feasible schedule of reservation requests within the intervals of the piecewise bandwidth availability graph. □

Based on lemma 1, we now have the following theorem.

THEOREM 1. *The SMR^3 problem is an NP-hard problem.*

PROOF. The SMR^3 problem is a generalized version of the *simple*-SMR^3 problem. As the *simple*-SMR^3 problem is shown to be NP-hard, SMR^3 is also NP-hard. □

Given that the SMR^3 problem is an NP-hard problem, it is not possible to develop a polynomial time solution procedure to solve it optimally. As a result, we develop an efficient algorithm with polynomial running time that can construct effective solutions.

4. RRA: THE RESOURCE RESERVATION ALGORITHM

In this section, we develop an algorithm, called RRA (resource reservation algorithm) to solve our SMR^3 problem. We begin by providing an overview of the RRA in Section 4.1, which is followed by the detailed description in Section 4.2. In Section 4.3, we illustrate the working of the RRA algorithm via an example. In Section 4.4, we show the polynomial running time of the RRA algorithm.

4.1 Algorithm Overview

The RRA algorithm runs in iterations. During every iteration, RRA will try to reserve resources for some of those requests that can have a large effect on the objective function, i.e., the requests that can utilize the largest amount of bandwidth. At the end of an iteration, there may be few reservation requests that were not accommodated in the bandwidth availability graph. The RRA algorithm then updates the bandwidth availability graph to reflect the current reservation of requests that were accommodated. This is followed by more iterations in which RRA tries to reserve resources for the remaining requests (if any) in the updated bandwidth availability graph. An obvious question here is: how subsequent iterations can accommodate the requests if the current one cannot? The answer will become clear during the detailed description of the algorithm. If, during some iteration, *none* of the remaining requests could be accommodated, then the RRA algorithm will stop. The remaining requests that were not accommodated cannot be satisfied. The RRA algorithm runs in polynomial time and can construct effective solutions.

4.2 Detailed Description

In this section, we will provide detailed description of the RRA algorithm. Every iteration within RRA consists of three phases. We will describe these phases here.

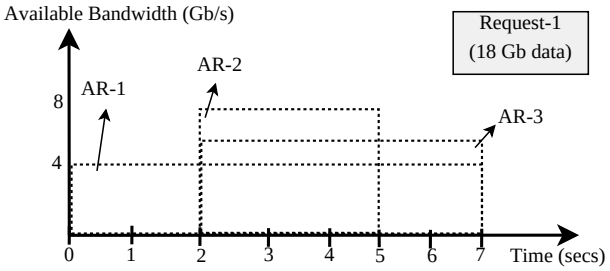
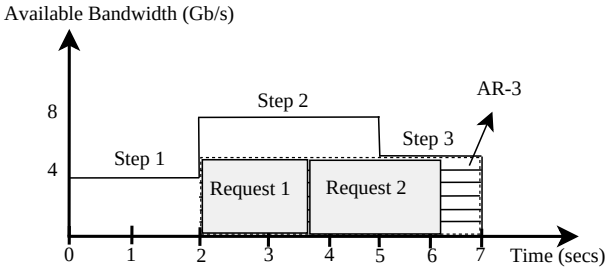
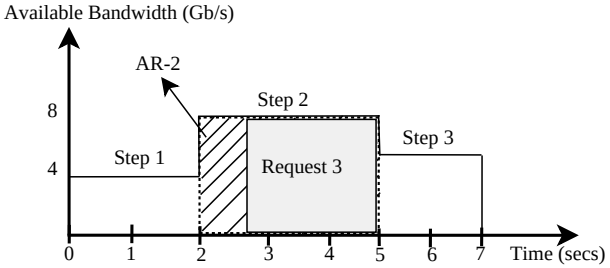


Figure 6: Assigning an AR to a request.



(a) AR assigned to multiple requests.



(b) Overlapped AR assignment.

Figure 7: An example illustrating the need for feasibility check.

4.2.1 Phase I: Identify Accommodating Regions

The RRA algorithm begins an iteration by identifying one accommodating region (AR) for every step in the bandwidth availability graph. For a particular step size, RRA identifies the corresponding AR as a rectangle within the bandwidth availability graph that (i) contains the step, (ii) has a height that corresponds to the maximum available bandwidth of that step, and (iii) has the longest length possible while maintaining the bandwidth. These ARs can be efficiently constructed using an algorithm proposed in [7]. The current pending reservation requests needs to be accommodated in these ARs. Note that an AR for a step may span multiple other steps, and a single AR may accommodate multiple reservation requests. As an example, Fig 5 shows the ARs for all three steps. The region AR-1 for step-1 spans from time [0–7], region AR-2 spans from time [2–5], and region AR-3 spans from time [2–7]. Regions AR-1 (with bandwidth 4 Gb/s) and AR-3 (with bandwidth 6 Gb/s) are spanning multiple time steps, and their height is corresponding to the bandwidth of their respective time steps. Region AR-2 (with bandwidth 8 Gb/s) spans only step-2 because the bandwidth of adjacent steps 1 and 2 is less than that of step-2.

4.2.2 Phase II: Initial Assignment

After identifying all ARs, the next step in the RRA algorithm is to assign individual requests to these ARs. In order to accomplish this, for every request ρ_i, RRA identifies the AR that can finish the request in the shortest amount of time (while satisfying the start and end time requirements of the request). In case there are multiple ARs that can finish the request in shortest time, RRA will assign the AR with smallest bandwidth to request ρ_i. This will ensure that the ARs with excessive bandwidth availability can be used by requests that can utilize such large bandwidth. Figure 6 shows an example where Request-1 with 18 Gb of data can utilize a maximum bandwidth of 6 Gb/s with earliest start time and completion deadline as 0 and 7 respectively. It also shows all three ARs that fall within the start and end time requirements of the request. Out of the three ARs, only two, i.e., AR-2 with 8 Gb/s and AR-3 with 6 Gb/s, can satisfy Request-1 in 3 secs. AR-1 with 4 Gb/s will require 4.5 secs to satisfy the request. In this case, the RRA algorithm will assign AR-3, i.e., the AR with minimum bandwidth among the two best ARs, to Request-1. If a certain request cannot be accommodated inside any AR, then that request can not be satisfied and will be removed from further consideration.

A situation is possible where none of the available ARs is assigned to more than one reservation requests, and none of the requests gets assigned to overlapping ARs. In such a situation, the current initial assignment is considered as a feasible assignment. However, it is also possible that some ARs (overlapping/non-overlapping) get assigned to multiple reservation requests (e.g., see Fig. 7). In such a scenario, the RRA algorithm needs to make sure that the current assignments are indeed feasible, i.e., whether all/few of the requests can be accommodated inside the assigned RAs.

4.2.3 Phase III: Ensuring Feasibility

This is the most important phase of the RRA algorithm. The goal of this phase is to determine whether the assigned ARs can accommodate the requests to which they were assigned. Furthermore, this phase also calculates the actual bandwidth and the start and end times that will be assigned to each request that can be accommodated. For this phase, RRA needs to maintain the following information for every identified AR: (i) the start time of the AR, (ii) the end time of the AR, (iii) a list of other overlapping ARs, and (iv) all the requests to which it was assigned. The RRA algorithm will then iterate through the ARs one by one. It is likely that the ARs with large bandwidth can complete the requests in less amount of time. As a result, for the benefit of the objective, the RRA algorithm will iterate through the ARs in a decreasing order of bandwidth values (ties are broken randomly). For every AR considered, the goal is to see if it can accommodate all the requests to which it was assigned. If not, then the goal should be to accommodate as many as possible. Furthermore, once a request is accommodated inside an AR, the size of other overlapping ARs should be adjusted as follows. If the accommodated request splits some overlapping AR into left and right portions, then the larger portion will be retained, and the smaller portion will be removed from that AR. Note that, if needed, the available bandwidth in the removed portion can be taken into consideration during the next iteration of the RRA algorithm.

To accommodate the requests within an AR, the RRA algorithm follows a greedy approach. It will iterate over the requests in increasing order of starting times, breaking ties randomly. The reason to iterate in increasing order of starting times is to reduce the amount of time gaps between accommodated requests within the AR.

To understand the processing of each request within the iteration, we use the following notation. We denote $\mathcal{R} = \{\rho_0, \rho_1, \cdots, \rho_{N\text{-}1}\}$ as the set of all N requests. For request ρ_i, denote the maximum bandwidth that can be used as $B(\rho_i)$ b/s, the requested start time before which the data transfer should not start as $S(\rho_i)$, the requested end time by which the data transfer should complete as $C(\rho_i)$, and the amount of data that need to be transferred as $D(\rho_i)$ bits. For request ρ_i, denote the actual start time of data transfer in a solution as $s(\rho_i)$, the actual bandwidth that was reserved for ρ_i as $b(\rho_i)$ b/s. The actual completion time for ρ_i can be calculated as $c(\rho_i) = s(\rho_i) + \frac{D(\rho_i)}{b(\rho_i)}$. For each request ρ_i under the greedy approach, the RRA algorithm will do the following:

- The starting time of request ρ_i (i.e., $s(\rho_i)$) will be calculated as the maximum of $S(\rho_i)$ (earliest possible start time) and the starting time of the AR. If this value comes out to be larger than $C(\rho_i)$ (the completion deadline for the request), then this request cannot be accommodated during this iteration, and will be considered in the later iterations.
- The bandwidth allotted to request ρ_i will be calculated as the minimum of $B(\rho_i)$ (maximum possible value that the ρ_i can support) and the maximum bandwidth available in the AR.
- Using the above values of start time and allotted bandwidth, the completion time of the request ρ_i will be calculated. If the value of completion time exceeds the end time of the AR, then the request ρ_i cannot be accommodated during this iteration of the RRA algorithm.
- Finally, if ρ_i can be accommodated within the AR, then the start time of AR will be updated to the completion time of ρ_i.

Once the algorithm has finished iterating over the requests for a particular AR, it is possible that there are few requests remaining that still need to be accommodated. There requests will be considered during the next iteration of the RRA algorithm. Within this iteration, the RRA algorithm will now move on to the AR with next highest bandwidth, and will repeat the greedy approach to fit the requests to which this AR was assigned.

4.2.4 Preparing for the next Iteration

An iteration of RRA ends when the RRA algorithm finish iterating through all ARs. At this time, there may be some remaining reservation requests that RRA was not able to accommodate. The RRA algorithm will now update the bandwidth availability graph while taking into consideration the current accommodated requests. We have developed an efficient algorithm, called C-BAG (Constructing an Updated Bandwidth Availability Graph), that constructs an updated bandwidth availability graph given the requests that are accommodated within the current graph. The details of the C-BAG algorithm are as follows:

The C-BAG Algorithm. To begin with, the C-BAG algorithm will calculate the number of steps that will be there in the updated BAG. To achieve this objective, C-BAG uses an efficient data structure called `Set`. The `Set` data structure is a collection of unique values. The C-BAG algorithm first inserts the start and end times of all steps in the current BAG into `Set`. This is followed by the insertion of all actual start and end times of the accommodated requests within the current BAG into `Set`. With an efficient hash function, the insertion can be made a constant time operation. It can be verified that the size of `Set` minus 1 will be the number of steps in the new updated BAG. The values in `Set` and the number of steps in the new BAG are needed for the next operation in the C-BAG algorithm.

After calculating the number of steps in the new BAG, the C-BAG algorithm iterates over `Set` in increasing order of stored values. Every encountered value in `Set` marks the beginning of a new step, and ending of the previous step in the updated BAG. To obtain the available bandwidth in these steps for the new BAG, the C-BAG algorithm maintains and updates two variables: (i) the amount of bandwidth that is currently in use by the accommodated requests (denoted by `BW-USE`), and (ii) the bandwidth of the step that is active at this time in the old BAG (denoted by `BW-STEP`). For every step in the new BAG (i.e., at every encountered time value in the `Set`), the amount of bandwidth for this new step is equal to `BW-STEP` minus `BW-USE`.

After constructing a new bandwidth availability graph, the RRA algorithm will run through the three described phases again. Note that in this new iteration, an AR that gets assigned to a previously non accommodated request will be different from ARs in the previous iteration. This is the reason that a request that may not be accommodated inside an AR during one iteration may get accommodated during some subsequent iteration.

RRA_Algorithm(`Requests Reqs, Steps` $\mathcal{S}$**)** /*$\mathcal{S}$ is the bandwidth availability graph */

1. Use [7] to create a set $\mathcal{R}$ of accommodating regions from $\mathcal{S}$
2. For every request $q \in$ `Reqs`
3. Identify ARs that can satisfy q in smallest amount of time
4. Among the identified ARs, assign the one with minimum bandwidth to q
5. If there is no identified AR for q, then q cannot be satisfied and is removed from `Reqs`
6. Consider each AR $r \in \mathcal{R}$ in decreasing order of bandwidth values
7. Consider all requests $q \in$ `Reqs` to which region r was assigned in the increasing order of earliest possible start times
8. If q cannot fit in region r, then q cannot be satisfied in this iteration
9. If q can fit inside region r, then
10. Mark the actual start time of q as the current start time of region r
11. Assign the bandwidth of r as the actual bandwidth allotted to q
12. Using the allotted bandwidth and start time, calculate the finish time of q
13. Update the start time of region r to the finish time of q
14. For all overlapping regions $\hat{r}$ partitioned by q
15. Remove the smaller portion of $\hat{r}$.
16. If the number of requests assigned in the previous step is not zero
17. Use the C-BAG algorithm to update the bandwidth availability graph $\mathcal{S}$ while considering all the accommodated requests in `Reqs`
18. Remove the accommodated requests from `Reqs`
19. Goto step 1
20. The requests that were not accommodated cannot be satisfied

Figure 8: Pseudocode of the RRA algorithm.

C-BAG_Algorithm(`Requests Reqs, Steps` $\mathcal{S}$**)**

1. Create a new `Set`
2. Add the actual start and actual end time of the satisfied requests to the `Set`
3. Add the start and end times of all the steps in $\mathcal{S}$ to the `Set`
4. The number of steps in new bandwidth availability graph will be sizeof(`Set`) -1
5. Sort the values within the `Set` in increasing order
6. Initialize $i = 0$
7. For each time value t in the sorted `Set`
8. if $i <$ sizeof(`Set`)
9. Mark t as the beginning of the i-th step in the new bandwidth availability graph $\hat{\mathcal{S}}$
10. If$(i > 0)$
11. Mark t as the end time of the $(i-1)$-th step
12. If t is the start time of some step in $s \in \mathcal{S}$
13. Denote the current available bandwidth as B
14. For all the accommodated requests that start at t
15. Denote sum_1 as the sum of used bandwidth values in these requests
16. For all the accommodated requests that end at t
17. Denote sum_2 as the sum of used bandwidth values in these requests
18. Denote $U = sum_1 - sum_2$ as the bandwidth in use
19. if $i <$ sizeof(`Set`)
20. Mark the bandwidth of i-th step in $\hat{\mathcal{S}}$ as $B - U$
21. Increment i by one
22. Update $\mathcal{S} = \hat{\mathcal{S}}$

Figure 9: Pseudocode of the C-BAG algorithm.

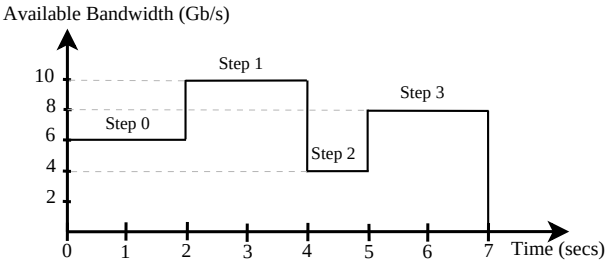
The RRA algorithm stops if it cannot accommodate any requests during an iteration, or if there are no more remaining requests that need to be accommodated. Figure 8 shows the pseudo code of the RRA algorithm, and Fig. 9 shows the pseudo code of the C-BAG algorithm.

4.3 An Example

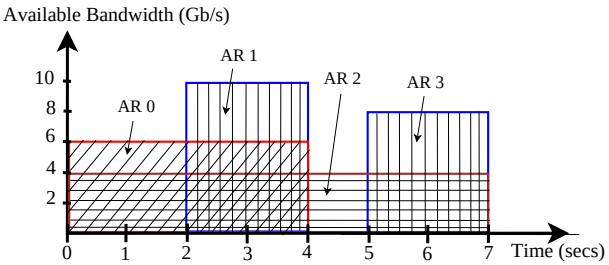
In this section, we will walk through an example to illustrate the workings of the RRA algorithm. Figure 10(a) shows an initial bandwidth availability graph with four steps. Step-0 spans over the interval [0-2] with 6 Gb/s bandwidth, step-1 spans [2-4] with 10 Gb/s bandwidth, step-2 spans [4-5] with 4 Gb/s bandwidth, and step-3 spans [5-7] with 8 Gb/s bandwidth. Table 1 shows the reservation requests that are to be accommodated in the bandwidth availability graph of Fig. 10(a). The units used in this example are for the illustration purpose only. As a example, the time duration can be in minutes or hours instead of seconds that we have used here. The first iteration of the RRA algorithm will operate on the initial bandwidth availability graph of Fig. 10(a).

The first phase in this iteration is to build a set of accommodating regions for every step in the graph. Figure 10(b) shows the ARs for the steps in Fig. 10(a).

The second phase, i.e., the initialization phase involves iterating through the requests and assigning ARs to them. The ARs assigned to individual requests during the second phase are shown in the second column of Table 2. Request #0 can take 1.667 seconds to finish within ARs 0, 1 and 3. However, it will be assigned AR-0 because the bandwidth of AR-0 is closest to the maximum of 6 Gb/s bandwidth that the request can use. Similarly, requests #1, #2, and #3 will be assigned ARs 0, 1, and 0 respectively.



(a) Bandwidth availability graph.



(b) Accommodating regions.

Figure 10: An example.

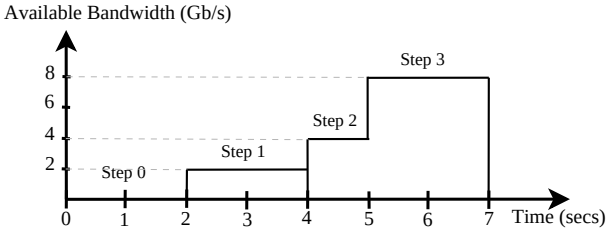
Table 1: A set of requests.

#	Data (Gb)	Max BW (Gb/s)	Earliest Start (secs)	Completion Deadline (secs)
0	10	6	2	7
1	8	6	1	7
2	16	8	2	5
3	12	6	0	7

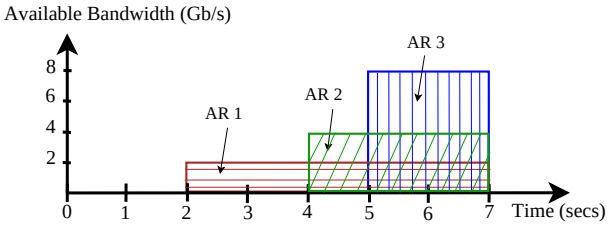
Table 2: ARs during Iteration1.

#	Initial AR	Accommodated
0	0	NO
1	0	NO
2	1	YES
3	0	YES

For the third phase of this iteration, the RRA algorithm will first select the AR with the largest available bandwidth, which is AR-1 (see Fig. 10(b)). As there is only one request (i.e., #2) to which this AR was assigned initially, request #2 will be accommodated in this AR with starting time of 2, ending time of 4, and bandwidth of 8 Gb/s. At this point, the boundaries of the overlapping ARs will be adjusted. That is, the end time of AR-0 will be updated to 2, and the start time of AR-2 will be updated to 4. As there is no other request to which AR-0 was assigned, the RRA algorithm will move onto AR-3 that has the next highest bandwidth. However, AR-3 is not assigned to any request. So, the RRA will move over to the next AR, i.e., AR-0. Now, there are three requests to which AR-0 was assigned. The RRA algorithm will iterate over these requests in the increasing order of their earliest start time possible. That is, it will first check request #3, which will be accommodated with start time of 0, end time of 2, and allotted bandwidth of 6 Gb/s. Since the end time of AR-0 was updated to 2, the remaining two requests cannot



(a) Bandwidth availability graph.



(b) Accommodating regions.

Figure 11: Updated steps and regions after the first iteration.

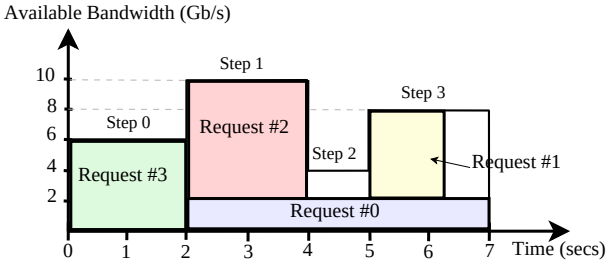


Figure 12: Final solution for Fig. 10(a) and Table 1.

be accommodated within this AR. The RRA will next move over to the last region, i.e., AR-2. As this AR was not assigned to any request, the first iteration is completed.

After the first iteration, there are still two requests remaining that were not accommodated. So, the RRA will update the bandwidth availability graph taking into consideration the two accommodated requests. The updated bandwidth availability graph is shown in Fig. 11(a), and the corresponding ARs are shown in Fig. 11(b). Following the same approach as the first iteration, the RRA will accommodate request #1 in AR-3 during this iteration. In the third iteration, the RRA algorithm will finally accommodate request #0. The final accommodated requests are shown in Fig. 12.

4.4 Complexity of the RRA Algorithm

In this section, we will show that the RRA algorithm developed in this paper has polynomial time complexity. We denote the number of requests that need to be accommodated as N, and the number of steps in the initial bandwidth availability graph as M. Every iteration in RRA runs in three phase. We will analyze these phases one by one.

Phase I. For the first phase, the accommodating regions can be identified by using an algorithm [7] that runs in linear time with respect to the number of steps in the bandwidth availability graph. However, due to the accommodation of requests, the number of steps in the graph can increase with every subsequent iteration. In the worst case, every accommodated request can increase the number of steps by two. As a result, the running time of the first phase in worst case can be $O(N + M)$.

Phase II. For the second phase, the RRA algorithm identifies the best region corresponding to every request. The number of requests can be $O(N)$, and the number of regions in worst case can be $O(N + M)$ (as calculated for the first phase). Therefore, the worst case running time of this phase comes out to be $O(N \cdot (N + M))$.

Phase III. For the third phase, the RRA algorithm iterates over all ARs, i.e., $O(N + M)$ ARs in worst case. For every AR, it checks for the feasibility of accommodating $O(N)$ requests within that AR. As a result, the running time of the third phase is $O((N + M) \cdot N)$.

After the third phase, the RRA algorithm constructs a new bandwidth availability graph using the C-BAG algorithm. In the worst case, the number of steps in any bandwidth availability graph can be $O(M + N)$. The number of accommodated requests can be $O(n)$. As a result, the cost of sorting all values in `Set` is $O((M + N) \cdot \log(M + N))$. The bandwidth for the individual steps in the new BAG can be calculated in one pass over the sorted values in `Set`, i.e., in $O(M + N)$ time. As a result, the overall running time of C-BAG is $O((M + N) \log(M + N))$.

The RRA algorithm stops if an iteration cannot accommodate any reservation request. This means that at least one request is accommodated during every iteration of the RRA algorithm. As a result, the number of iterations are limited to $O(N)$. This gives us the final runtime complexity of the RRA algorithm as $O(N \cdot (N + M + (N \cdot (N + M)) + ((N + M) \cdot N) + (N + M) \cdot \log(N + M))$, which can be reduced to $O(N^3 + N^2 M)$.

5. NUMERICAL RESULTS

In this section, we present simulation results to show the efficiency of our RRA algorithm.

5.1 Operation of RRA

To begin with, we construct a schedule of requests using the RRA algorithm on a sample of randomly generated reservation requests and bandwidth availability graph.

Input. Table 3 shows details of the 30 steps in the bandwidth availability graph. Column 1 of Table 3 shows the step number, column 2 shows the start time of the step, column 3 shows the end time of the step, and the last column shows the amount of bandwidth available in the step. Table 4 shows a set of 15 requests that needs to be accommodated inside the bandwidth availability graph of Table 3. The first column of Table 4 shows the request number. Column 2 shows the time in seconds before which data transfer for this request cannot start. Column 3 shows the time limit by which the data transfer for this request should finish. Column 4 contains the maximum bandwidth that this request can use and the last column shows the amount of data that needs to be transferred for this request. The final row of Table 3 shows the total amount of data that needs to be transferred for all the requests as 11329.39 Gb.

Output. After providing the steps and requests as an input to the RRA algorithm, the schedule of the accommodated requests is

Table 3: A bandwidth availability graph.

Step #	Start Time (secs)	End Time (secs)	Available Bandwidth (Gb)
0	0	29	8.98
1	29	181	9.75
2	181	305	6.28
3	305	309	8.39
4	309	394	2.30
5	394	490	9.81
6	490	592	8.04
7	592	663	3.99
8	663	691	4.07
9	691	815	4.27
10	815	907	7.69
11	907	1106	3.59
12	1106	1211	8.50
13	1211	1411	3.01
14	1411	1566	6.01
15	1566	1577	8.43
16	1577	1746	9.47
17	1746	1875	9.15
18	1875	1944	9.48
19	1944	2028	6.53
20	2028	2059	2.38
21	2059	2084	5.42
22	2084	2251	6.50
23	2251	2293	4.91
24	2293	2351	8.49
25	2351	2530	4.54
26	2530	2606	6.79
27	2606	2626	9.74
28	2626	2728	3.21
29	2728	2842	9.22

Table 4: A set of requests.

Request#	Earliest Start Time (secs)	Latest End Time (secs)	Maximum Bandwidth (Gb/s)	Data (Gb)
0	364.31	1917.31	2.20	1139.13
1	1028.44	1099.47	5.15	121.84
2	988.20	1823.2	3.22	896.82
3	1707.58	1869.64	8.54	461.26
4	988.64	1078.64	6.91	207.31
5	1156.31	2263.43	8.15	3007.78
6	714.70	1612.70	7.70	2304.95
7	1288.25	1570.84	5.73	539.72
8	1871.83	2719.83	5.47	1545.57
9	1811.35	1926.49	5.64	216.59
10	2657.56	2720.56	5.72	120.09
11	2797.77	2841.03	8.79	126.72
12	2651.57	2681.57	7.30	72.98
13	2363.70	2585.31	4.00	295.43
14	2464.11	2549.11	9.64	273.20
Total Data				11329.39

shown in Table 5. Column 1 of Table 5 shows the request number. Columns 2 show the actual start time when this request will start transmitting the data. Column 3 shows the time when the data transmission for this request will finish. Column 4 shows the amount of bandwidth that will be reserved for this request, and the last column shows the difference between columns 3 and 2, i.e., the total time for which this request will transmit data. We can see that requests #5 and #6 were not accommodated in the final schedule. As a result, these requests are considered as unsatisfied. We can also see that requests #2 and #7 have overlapping schedules, i.e., they will be active at the same time for some duration. The last row of Table 5 shows the total time needed to complete the data transfer for all the requests as 1722.49 secs..

Table 5: Schedule for the requests.

Request#	Actual Start Time (secs)	Actual End Time (secs)	Alloted Bandwidth (Gb/s)	Time Taken (secs)
0	364.31	881.97	2.20	517.66
1	1046.33	1080.24	3.59	33.91
2	1411.00	1689.33	3.22	278.33
3	1707.58	1761.60	8.54	54.02
4	988.64	1046.33	3.59	57.69
5	N/A	N/A	N/A	N/A
6	N/A	N/A	N/A	N/A
7	1288.25	1482.13	2.78	193.88
8	2059.00	2399.13	4.54	340.13
9	1811.35	1849.73	5.64	38.38
10	2674.28	2711.65	3.21	37.37
11	2797.77	2812.19	8.79	14.42
12	2651.57	2674.28	3.21	22.71
13	2399.13	2473.00	4.00	73.87
14	2473.00	2533.12	4.54	60.12
Total Time				1722.49

5.2 Comparison with other heuristics

We next consider a network setup where the bandwidth availability of the network, i.e., the height each step, and the duration of each step between two end sites vary randomly. The height vary between zero and 10 Gb/s, and the duration vary between zero and 100 seconds. Each reservation request is given a random value for the earliest start time, the completion deadline, and the maximum usable bandwidth (which can take on a value between zero and 10 Gb/s). The earliest start times and the completion deadlines are restricted to the times for which bandwidth availability is known. Furthermore, for every request, the amount of data that needs to be transferred is limited by some fraction of a so-called *max-data-value*. This max-data-value for a request is the amount of data that can be feasibly transferred between the earliest start time and the completion deadline given the maximum usable bandwidth for the request.

Given the procedure to construct steps and requests, we first generate a pair of 300 random steps and 150 random requests that need to be accommodated within these steps. These serve as the input to the RRA algorithm. The results of RRA for this pair of steps and requests serve as *one* data point for the results. We then continue to generate random pairs of steps and requests, and continue generating schedules using the RRA algorithm. All these subsequent schedules gives us more data points for the results.

As a comparison, we also consider the schedules constructed from following two seemingly efficient heuristics.

First come first serve (FCFS). In this heuristic, the requests are considered for reservation within the bandwidth availability graph in the order in which requests are generated. Each request is accommodated in the AR where it can be completed in the shortest amount of time.

Largest bandwidth first (LBF). In this heuristic, the requests are considered for reservation within the bandwidth availability graph

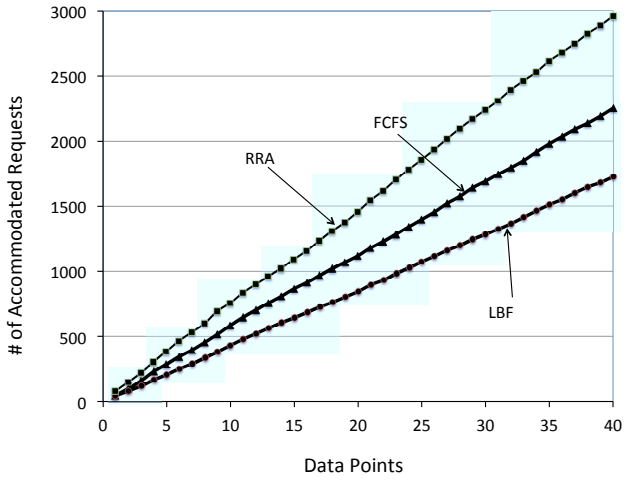


Figure 13: Comparison of the number of accommodated requests.

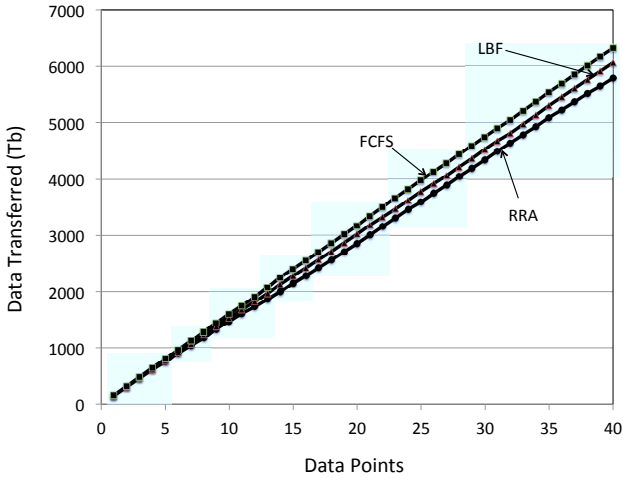
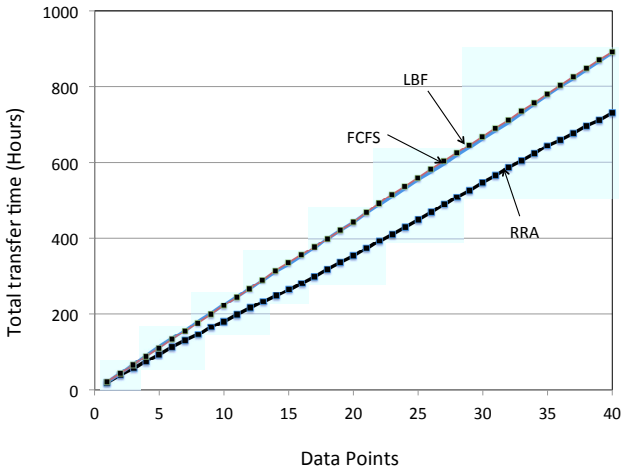
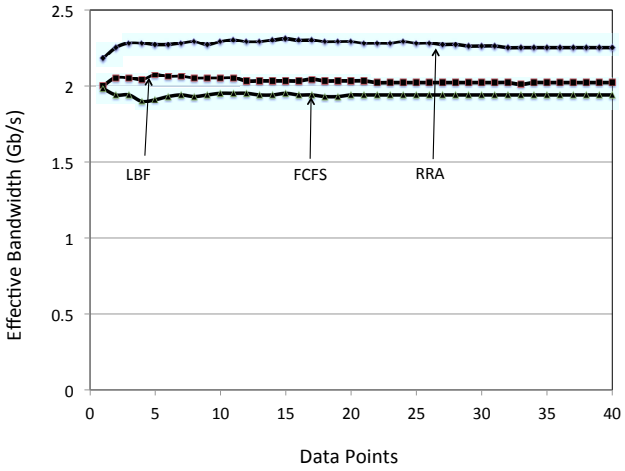


Figure 14: Comparison of total data transferred under three schemes.



(a) Total transfer time.



(b) Effective bandwidth.

Figure 15: Comparison of total transfer time and effective bandwidth under three schemes.

in the decreasing order of maximum bandwidth value that the requests can use. A request under consideration is accommodated in the AR where it can be completed in the shortest amount of time. It may seem that accommodating requests that can use the largest bandwidth may reduce the total data transfer time for the accommodated requests. However, as the results will show, this is not the case.

Figure 13 shows the comparison of the number of accommodated requests under RRA, FCFS, and LBF. X-axis shows the data points, and Y-axis show the cumulative number of requests that were accommodated under a particular scheme upto every data point. On an average, the number of requests that were accommodated under RRA are 75% higher than the LBF scheme, and 33% higher than the FCFS scheme.

Intuitively, one may think that the RRA algorithm may have chosen the requests that have small amount of data to transfer, thereby increasing the number of accepted requests. However, Fig. 14 shows that the total amount of data transferred under all the three schemes is approximately same. Under RRA, the data transferred is 4% less than that under LBF, and 8% less than that under FCFS.

Next, one may ask that if all the schemes were able to transfer similar amounts of data, then what is the advantage of RRA? Figure 15(a) shows that the amount of time taken to transfer total data under RRA is significantly lower than total time under the other two schemes. Both FCFS and LBF takes 21% more time than the time taken under RRA. Furthermore, Fig. 15(b) shows that the effective bandwidth utilization (the ratio of data transferred and the time taken to transfer the data) under RRA is largest among all the three schemes.

To alleviate a practical concern, we calculated the running time used by the RRA algorithm for the complete simulation on a 2.8 GHz Intel Core i5. The time comes out out to be just under 900 milisecs, which is orders of magnitude smaller than the gains in the data transfer time due to the RRA algorithm (see Fig. 15(a)).

Discussion. The results show that RRA is clearly a better algorithm than the FCFS scheme. The reason is that RRA benefits from

the additional knowledge about the input that it gets by considering multiple requests for reservation at the same time. Whereas, the FCFS algorithm blindly accommodate the requests as they arrive without taking into consideration any other requests.

The LBF scheme, like RRA, also have the additional knowledge about the input. However, while accommodating requests, LBF still accommodate requests one by one in isolation. On the other hand, instead of accommodating requests in isolation, the RRA algorithm distributes the requests among accommodating regions, and then tries to satisfy them. This makes RRA perform much better than LBF.

To summarize, our results show that the RRA algorithm can construct schedules that accommodate large number of reservation requests while transferring similar amount of data compared to LBF and FCFS schemes. Furthermore, the time taken to transfer the data is much smaller under RRA when compared with time under LBF and FCFS schemes.

6. RELATED WORK

There has been significant amount of existing research in the area of network QoS. This research can be broadly divided into two main categories: (i) QoS architectures/routing mechanisms and (ii) resource reservation protocols. QoS architectures and routing mechanisms provides procedures to create network paths that can provide some kind of QoS between end sites. A comprehensive survey of QoS/constraint based routing can be found in [13]. A framework for QoS-based routing can also be found in [4]. Details of QoS architectures such as DiffServ and IntServ architectures can be found in [1] and [2] respectively. In next generation networks, such as EsNet [5] and Internet2 [9], these QoS mechanisms are implemented by inter domain controllers, also known as IDCs. Resource reservation protocols (see e.g., [14, 10]), on the other hand, develop mechanisms for applications to convey QoS requests to the IDCs. The focus of our work is not to develop another QoS mechanism. Instead, we assume that such a QoS mechanism already exists within the IDC. Our focus is on how the applications at the end sites can exploit such QoS mechanisms to their advantage. That is, given the QoS characteristics of the network, we develop an algorithm to help a resource reservation protocol in constructing an optimal schedule of reservation requests.

There are some efforts that consider the possibility that reservation requests can be available in advance [8], or that the reservation requests can be flexible [7]. Usually, for flexible requests, the applications at the end sites submit a fixed reservation request to the network domain controllers. The application then gets a response from IDC on whether the request can be satisfied or not [7]. If the request cannot be satisfied, it is then modified by the applications, and re-submitted for reservation. It may require a few iterations before a request is modified in way that can be accepted. Furthermore, in the presence of multiple requests, existing mechanisms naively submit the reservation requests one by one, usually in the order of their arrival.

In contrast to such mechanisms, when multiple flexible reservation requests are available, (i) we exploit the notion of flexibility in all known reservation requests, and (ii) we do not submit reservation requests one by one to the domain controller. Instead, we develop and follow a completely novel approach. In our approach, applications ask the IDCs for the available resources. The domain controllers in the next generation networks can provide such information to applications on request [5, 9]. As a result, an application can now gather the information about available resources and the details about the multiple flexible requests. Given such information, we then develop an algorithm that can be used to construct a schedule of reservations that the network domain controller should be able to accommodate. In contract to existing mechanisms, our approach avoids the multiple iterations where applications keeps on modifying and re-submitting requests to the domain controllers in the hope of getting accepted. Our results show the significant performance improvements of our approach over the existing iterative and sequential approaches of request submission.

7. CONCLUSION

In this paper, we solved an important problem, called SMR^3, to accommodate multiple and concurrent network reservation requests between a pair of end-sites. Given the varying availability of bandwidth within the network, our goal was to accommodate as many reservation requests as possible while minimizing the total time needed to complete the data transfers. We proved that the SMR^3 is an NP-hard problem, and then developed a polynomial-time heuristic, called RRA, to solve the problem. The RRA algorithm hinges on an efficient mechanism to accommodate large number of requests by minimizing the bandwidth wastage. Finally, via numerical results, we showed that RRA constructs schedules that accommodate significantly larger number of requests compared to other, seemingly efficient, heuristics.

8. REFERENCES

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

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

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

[4] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the Internet," *RFC 2386*, 1998.

[5] "Energy sciences network," URL:http://www.es.net/

[6] M.R. Garey and D.S. Johnson, *Computers and intractability, a guide to the theory of NP-completeness*, W.H. Freeman & Co., New York, 1990.

[7] 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.

[8] 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

[9] "Internet 2," URL: http://www.internet2.edu/

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

[11] *Private communication with the developers/researchers associated with EsNet (www.es.net).*

[12] H.D. Sherali and W.P. Adams, *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, Kluwer Academic Publishers, Boston, 1999.

[13] 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.

[14] L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP) – version 1 functional specification," *RFC 2205*, 1997.