

Scalable Bandwidth Optimization in Advance Reservation Networks

Stephan Schmidt and Jérôme Kunegis

DAI-Labor
Technische Universität Berlin
Franklinstr. 28, 10587 Germany
{stephan.schmidt, jerome.kunegis}@dai-labor.de

Abstract—In this paper, we present an algorithm for continuous bandwidth optimization in networks supporting advance reservations such as Grid computing environments or QoS-aware MPLS networks. The need for such reoptimization arises when resources for incoming reservation requests are allocated using fast dynamic routing with simple path selection algorithms. Although this is often necessary to satisfy time constraints for answering reservation requests, it inevitably leads to network inefficiencies due to the fact that the resulting uneven load distribution leads to bottlenecks within the network.

We propose a combination of fast online bandwidth reservation with background reoptimization which continuously frees up bandwidth for future time slots in order to allow the network to carry more traffic without adding further capacity. A combinatorial time-approximation scheme suitable for large networks will be used. In addition, we evaluate several performance metrics and show that a significant improvement in request admission rate and bandwidth utilization can be achieved under the proposed scheme.

I. INTRODUCTION

In computer network theory, resource reservations are typically distinguished into two types. The first are *immediate reservations* which are issued to the network in a just-in-time fashion. This paper deals with the second reservation type, *advance reservations*. In this case, bandwidth resources are reserved a certain time interval, referred to as the *bookahead time*, before they are actually used. Advance reservations are widely used in networks supporting quality-of-service (QoS) functionality and have become applicable when the MPLS technology allowed for more sophisticated explicit routing features. Furthermore, advance reservations are an integral part in Grid computing environments [1], [2].

In such *advance reservation* environments, the service provider allows reservation requests to be made for a fixed period of time in the future. The time between issuing such a request and the actual start of the transmission, the *reservation time*, may vary from short intervals, for example ten minutes, to relatively long time periods, e.g., weeks. Upon receiving an advance reservation request, the service provider must notify the user within a certain period of time before the scheduled start of the transmission, known as the *notification interval*, whether it has been accepted or declined. This approach yields

several benefits for the service provider. On one hand, he gains additional information about the future traffic pattern and can decide which connection requests to admit in order to maximize the profit from the admitted connections, on the other he has an ample scope for dealing with each individual request, e.g., it can be queued when allocating the inquired resources is temporarily impossible and the designated path on which the transmission is to be routed can be changed for any future point of time without having to put up with any rerouting costs, as long as the connection has not yet been set up. Furthermore, the decision whether to accept or to decline it can be delayed until the notification interval is reached; this time can meanwhile be used to free up resources for the request, which otherwise would need to be declined.

The client also benefits from issuing a reservation for a transmission in advance. Most importantly he gains information regarding whether his reservation can be admitted, whereas the admission probability increases as the request is made sufficiently early.

Advance reservation concepts have been widely incorporated in frameworks providing QoS functionality [3]–[6]. When striving to provide QoS guarantees in advance reservation networks, service providers must use the available bandwidth to its full potential in order to meet the requirements of their users regarding the relevant parameters bandwidth, latency, jitter and delay. As clients compete for scarce resources in advance reservation networks, it is often necessary to add extra bandwidth in order to satisfy increasing demands and ensure timely processing and quality of service. However, adding link capacity always comes at a cost; therefore, one strives to maximize the network throughput by removing existent network inefficiencies. In order to achieve this goal, incoming resource reservation requests as well as already established connections need to be handled in an intelligent fashion. We will term an entity accepting reservation requests on behalf of the network *bandwidth broker*. The broker strives to utilize the network resources to its maximum potential by allocating available resources in an optimal fashion, whereas optimality depends on the objective function, for example admitting the maximum number of requests, maximizing the total allocated bandwidth or maximizing the total profit gained from accepting requests.

A. Related work

In order to accommodate as many connection setup requests as possible, efficient network flow routing algorithms are necessary. These have been studied extensively for immediate reservation networks [7], [8]. As reservation requests arrive one-by-one, the desired quick connection setup leads to an online allocation problem. This problem can be tackled with sophisticated routing algorithms which however do not meet the scalability requirements in large networks for quick connection setups. This necessitates the use of simple and scalable path selection algorithms, which in turn will cause routing inefficiencies within the network leading to “stranded” capacity.

An approach to combine dynamic online connection routing in a connection-oriented network with an offline optimization module which constantly rebalances the load in the network whenever a certain imbalance threshold is exceeded has been examined in [8]. In this scenario, the network operator determines a rebalancing benefit indicating the amount of traffic that could additionally be routed if the current traffic were to be redistributed. If this threshold is exceeded, i.e. the benefits or reoptimizing the network exceeds the incurred costs of the flow rerouting, then reoptimization is performed.

In this paper, we extend the approach from [8] to advance reservation networks in order to maximize the throughput of the network. Note that in this immediate reservation scenario, rerouting is only performed if a certain rerouting benefit threshold is exceeded. This notion is obsolete in our advance reservation reoptimization scheme, since we perform *background optimization* only. This means that *future* time slots are optimized in the background in an iterative and continuous fashion, and hence no rerouting costs are incurred on currently routed connections.

The reoptimization can be modeled mathematically as a *maximum concurrent flow problem* (MCFP), one of the standard network flow problems. For this category of problems, an efficient combinatorial algorithm has been proposed in [9]. It has been substantially improved later by Fleischer [10] and Karakostas [11], where it is shown that this algorithm can be adapted to be independent of the number of commodities residing in the network.

B. Summary of Contributions and Outline

In this paper, we propose a scalable background optimization scheme for advance reservation networks. It has the following characteristics:

- No impact or cost incurred on currently routed traffic
- Substantial improvement in request admission rate
- Substantial improvement in overall network capacity utilization
- Computationally efficient

For single time slot optimization, we use a slightly modified version of the algorithm developed by Fleischer [10] to account for advance reservation network features. This algorithm is described in the following section. Subsequently, we demonstrate in Section III how incoming reservation requests are

handled and develop a heuristics for determining the time slot optimization order. In Section IV, the results of the conducted simulations are described. The paper ends with concluding remarks and an outlook on future work.

II. SINGLE SLOT OPTIMIZATION

A network $N = (V, E)$ with a set of commodities (source-sink router pairs) (s_i, t_i) and edge capacities $c(e)$ is given. Additionally, each commodity (s_i, t_i) is associated with a demand d_i . In the absence of traffic forecasts, we define the traffic demand matrix as the currently provisioned demands and assume it to be directly proportional to future demand matrices. With these input parameters, the linear programming formulation of MCFP is

$$\begin{aligned} \max \lambda & & (1) \\ \sum_{P:e \in P} x(P) \leq c(e) & \quad \forall e \in E \\ \sum_{P \in P_i} x(P) \geq \lambda d_i & \quad \forall i \\ x(P) \geq 0 & \quad \forall P \end{aligned}$$

The MCFP therefore consists of computing a flow x so that for each commodity (s_i, t_i) at least a fraction λ of its demand d_i is satisfied. Note that since we use the currently provisioned demands, $\lambda_{\text{opt}} \geq 1$.

The dual linear program (2) assigns lengths $l(e)$ to the edges and weights z_j to the commodities of the network. The length of an edge represents the marginal cost of using an additional unit of capacity of the edge, while the weight of a commodity represents the marginal cost of not satisfying another unit of demand of the commodity.

The constraints of the dual program postulate that for each commodity c_j the length of the shortest path between s_j and t_j for that commodity has to be at least z_j , and the sum of the products of all commodity weights z_j and demands d_j has to be at least 1.

$$\begin{aligned} \min \sum_{e \in E} c(e)l(e) & & (2) \\ \sum_{e \in P} l(e) \geq z_j \forall j & \quad \forall P \in \mathcal{P}_j \\ \sum_{1 \leq j \leq k} d_j z_j \geq 1 & \\ l(e) \geq 0 & \quad \forall e \in E \\ z_j \geq 0 & \quad \forall j \end{aligned}$$

In our algorithm, we initialize the lengths as $l(e) = \delta/c(e)$, while $\delta = (\frac{m}{1-\epsilon})^{-1/\epsilon}$, and start out with the zero flow $x \equiv 0$. Proceeding in phases, we iterate over the k commodities while the solution is dually feasible ($D(l) < 1$). For each commodity, we determine the shortest path between the source and the sink node of the commodity using the length function l . Subsequently, we determine its capacity and augment the

minimum of this capacity and the remaining demand along this path. Finally, the primal (x) and dual (l) variables are updated. Note that the dual length variable l is updated by increasing the length of an edge exponentially with regard to the congestion of the edge. The factor $(1 + \frac{\epsilon c_{\min}}{c(e)})$ is hereby chosen in such a fashion that the length of the bottleneck edge increases by a factor of $(1 + \epsilon)$ during each step. The algorithm terminates when the dual objective function value $D(l) := \sum_{e \in E} c(e)l(e)$ is at least 1.

In connection-oriented networks where information has to be stored at the routers on a per-flow basis, distributing requests on more than one path over several slots will cause administrative overhead. However, this cannot be totally avoided since this would entail the additional constraint limiting the maximum number of paths per commodities to $k \in \mathcal{N}$. This brings about a new problem type, a so-called *k-splittable flow problem*, which has been analyzed by Baier, Köhler and Skutella in [12]. To our knowledge, no time-approximation scheme is known for this class of flow problems.

However, by handling the total traffic demand independent of the number of the requests for a router pair as a single commodity, we can eliminate scalability problems by simply routing requests along the same path until its designated bandwidth is exceeded; the remaining flow of the request causing the overflow is routed on the next path and so on. This leads to at most $\sum_{c \in C} |P_c - 1|$ fragmented requests. This is a fixed value independent of the total number of connections, hence the scalability of our framework is ensured.

The final algorithm for optimizing a single time slot is depicted in Figure 1. Fleischer [10] has shown that it constitutes a fully polynomial time-approximation scheme for the maximum concurrent flow problem, and that an ϵ -approximate solution can be obtained in a running time of $O^*(\epsilon^{-2}m(k+m))$.

III. ONLINE RESOURCE ALLOCATION AND OPTIMIZATION SLOT SELECTION

The bandwidth broker of our framework responsible for allocating resources receives so-called advance reservations. The term *advance reservation request* denotes a request for a connection setup that specifies the following parameters:

- The source router s
- The target router t
- The starting time t_{start} of the connection
- The end time t_{end} of the connection
- The bandwidth b to be allocated

We define an advance reservation request as $r = (s, t, t_{\text{start}}, t_{\text{end}}, b)$. We allude to the fact that the terms *reservation* and *request* are sometimes confused. Generally speaking, the former refers to the actual process of allocating a path within the network, while the latter term merely denotes the request for such an allocation, regardless whether it has yet been accepted. The interval for which reservations can be made is called the *book-ahead time*, and the time between the issuing of the request and the actual start of the transmission is called the *reservation time*. It is customary in network practice

```

Input:
network  $N$ 
capacities  $c(e)$ 
source-sink router pairs  $(s_i, t_i)$ ,  $1 \leq i \leq k$ 
accepted requests  $R_j$  for each commodity  $j$ 
desired accuracy  $\epsilon$ 
Output: primal (infeasible) and dual solutions  $x$  and  $l$ 

Initialize  $l(e) = \delta/c(e) \forall e \in E$ ,  $d_j \leftarrow \sum_{r \in R_j} b(r)$ ,  $x \equiv 0$ 
while  $D(l) < 1$ 
  for  $j = 1$  to  $k$  do
     $d'_j \leftarrow d_j$ 
  end for
  while  $D(l) < 1$  and  $d'_j > 0$ 
     $P \leftarrow$  shortest path between  $s_j$  and  $t_j$  using  $l$ 
     $c_{\min} \leftarrow \min\{d'_j, \min_{e \in P} c(e)\}$ 
     $\text{count}(P) = \text{count}(P) + 1$ 
     $d'_j \leftarrow d'_j - c_{\min}$ 
     $\forall e \in P: l(e) \leftarrow l(e)(1 + \frac{\epsilon c_{\min}}{c(e)})$ 
  end while
  if  $t = 2^{\frac{1}{\epsilon}} \log_{1+\epsilon} \frac{m}{1+\epsilon}$ 
    for  $j = 1$  to  $k$  do
       $d_j \leftarrow 2d_j$ 
    end for
  end if
end while

if  $\lambda = t - 1/\log_{1+\epsilon} \frac{1}{\delta} < 1$  break
for  $j = 1$  to  $k$  do
   $\forall P \in \mathcal{P}_j:$ 
    while  $x(P) < (\text{count}(P) / \sum_j \text{count}(P_j))d_j$  do
      route  $r \in R_j$  on  $P$ ,  $R := R \setminus \{r\}$ 
    end while
end for

```

Fig. 1. MCFP time-approximation scheme for slot-based optimization in advance reservation networks.

to set an upper bound for the book-ahead time so the network status information pertaining to the reservations made can be stored and retrieved efficiently. In order to gain the acceptance of its intended users, the operator of a network incorporating advance reservations needs ensure that longer reservation times entail higher admission probabilities.

In addition to the increasing his admission chances, the user's prime motivation to issue an advance instead of an immediate reservation is that he expects a reply from the bandwidth broker whether the reservation request can be admitted. If the broker does not answer in due time or as it sees fit, the concept of planning reliability is foiled. Therefore, we employ a *notification interval* concept, meaning that once a user issues a request to the bandwidth broker, the latter must respond with an *accepted* or *declined* message within this notification interval. In case the reservation time falls short of the notification interval, this reply must be made

at once. Within our framework, we will set this notification interval to a fixed value for all requests. However, this may be extended in a straightforward manner down to the level of individual requests. For this case, the user issuing a request specifies the response time himself for the bandwidth broker, acknowledging the fact that allowing a longer response time entails a higher probability that a positive response will be given.

A. Time Slots

One of the main advantages of advance reservation environments is that the bandwidth broker does possess information regarding the future traffic pattern by means of the advance reservation requests. Therefore, we can strive to not distribute the *current* network load more evenly but also the load requested to be allocated for any future point in time. Naturally however, a reoptimization cycle can only be performed for time spans where the demand matrix is constant, hence our goal is to put a bound on the number of changes in the demand matrix. We will refer to such a time frame with a constant demand matrix as a *time slot*.

It is less beneficial to perform reoptimization for slots with a shorter length than others since the shorter a slot is, the lower the probability that a newly arriving request will fall into it. Yet, arriving or pending requests that do fall into this slot have a smaller chance of being admitted than others with the same bandwidth, which thwarts the fairness principle. Hence, we employ time slots of a *fixed length*. The respective value is specified by the network operator and is chosen in a fashion that takes the specific features of the network topology into account. To this end, the average length of the reservations to be allocated is the prominent factor; it should correlate with the length of the chosen time slot, i.e., the smaller this mean value is, likewise the smaller the time slot length needs to be.

B. Handling Incoming Requests

We employ a min-hop routing algorithm to allocate paths to newly arriving requests. Upon receiving an advance reservation request $r = (s, t, t_{\text{start}}, t_{\text{end}}, b)$, the bandwidth broker first of all acknowledges its receipt. Subsequently, it tries to allocate resources for the request on a per-slot-basis, proceeding as follows:

- For each time slot t_i with $t_{\text{start}} \leq t_i \leq t_{\text{end}}$, the bandwidth broker computes the min-hop path from s to t with a capacity $c(P) \geq b$. For this purpose, if $N = (V, E)$ is the mathematical representation of the network, all edges are removed from the network N whose residual capacity is smaller than b , thus generating an auxiliary network $N' := N \setminus \{e \in E | c(e) - x(e) < b(e)\}$.
- Dijkstra's algorithm is run on N' with starting node s , giving a shortest (s, t) -path with a minimum capacity of b in case such a path exists.
- The broker repeats the above algorithm for all time slots $t_{\text{start}} \dots t_{\text{end}}$ relevant for the given request. If paths $P_{t_{\text{start}}} \dots P_{t_{\text{end}}}$ with a minimum capacity of b do exist for all slots, the request can be allocated. In this case, the

broker reserves a bandwidth of b for all time slots on the respective path computed for that slot.

- In the other case, we have encountered a slot where no (s, t) -path with the required bandwidth exists while iterating over the time slots. Therefore, it is momentarily impossible to accept the reservation request. The bandwidth broker queues the request in case the notification interval has not yet expired, otherwise the issuing user must be notified that the request had to be declined.

If a reservation request needs to be queued, it is stored in a global repository of pending requests. Apart from this, the request is also associated with each time slot $t_{\text{start}} \leq t_i \leq t_{\text{end}}$ along with its respective state information (*accepted*, *declined* or *pending*). In the following sections, it can be observed that this information is required during the optimization.

First, the question arises on which grounds the optimizer chooses the time slot to optimize next. Naturally, it is feasible to just repeatedly iterate over the time slots in chronological order. However, it is self-evident that this straightforward approach will not yield the best results, i.e. optimizing a slot for which a large number of requests is pending is usually to be preferred over optimizing one with no pending requests at all. The background optimization module, which we will henceforth term *optimizer*, uses the following heuristics to assign priorities to time slots; the time slot with the highest priority will be optimized next:

- Slots for which no new requests have arrived since the last optimization have the lowest priority.
- Slots without any pending requests have the second lowest priority.
- The next criterion is the number of optimization cycles already performed on this time slot; slots that have undergone a higher number of total optimization cycles have a lower priority.
- If none of the above criteria is applicable, the priority is defined as the number of pending requests times the reciprocal value of the average link utilization.

The plausibility of the first criterion is obvious, since the network state for the given slot is still the same as after the last optimization cycle, therefore an additional optimization cycle will not change the allocated paths.

The second criterion ensures that slots that have reservation requests pending for admission are optimized before those for which no such requests are in the queue. However, if there are no pending requests for a slot at all as we are about to conduct optimization, it still makes sense to optimize the slot since the load redistribution will still free up resources. These resources can in turn be occupied by newly arriving requests.

The third criterion is employed in order to prevent a slot from being passed over time and again. If this were to happen, in a worst-case scenario all requests with a reservation duration of several slots that are pending for this ignored slot could not be admitted although meanwhile an abundance of resources has been freed up in the repeatedly optimized adjacent slots.

The motivation for employing the last criterion lies within the fact that for slots with a lower average utilization but a

relatively high number of pending reservation requests, the chance that there are pending requests, which can be routed after the optimization, will be significantly higher. However, it is infeasible to show that this heuristics will be optimal with regard to the goal of freeing up the most bandwidth over all slots since we have no knowledge of the future traffic pattern; optimality can solely be guaranteed within the scope of a single time slot. The in-depth study of slot selection heuristics is left for future research (cf. Section V).

IV. SIMULATION AND RESULTS

A. Network Topology

For our experimental studies we used the network shown in Figure 2. The depicted network consists of 15 nodes. All links in the network are uni-directional and have a total available bandwidth of 100 Mbps. During our simulations, we generate demand between three source-sink router pairs, namely (1, 4), (5, 9) and (10, 11). These router pairs therefore represent the commodities of our network.

The test network was chosen in such a fashion that it features two bottlenecks arranged in a hierarchical order, namely the link 2-3 and the link sequence 6-7-8; this in turn results in a staged blocking probability for the commodities, from highest to lowest in the order (10, 11), (5, 9) and (1, 4).

This design enables us to examine to what extent the background optimization is able to free up resources for the commodities (10, 11) and (5, 9), which are receiving subproportional bandwidth with respect to their total demand due to min-hop routing.

In our testbed, there are two different brokers registered with the system manager of the framework. Both brokers use the standard min-hop routing algorithm described in Section III-B for routing the incoming requests.

The first broker (*MH + BO*) implements the background optimization approach presented in the previous chapter. Thus, reservation requests that cannot be admitted at the time of their arrival are queued for later reconsideration until their notification interval expires. Moreover, it employs an optimizer responsible for conducting the background optimization.

For the second broker (*MH*), we disabled background optimization. It therefore declines non-routable reservations immediately and does not reroute any flow. We send all requests

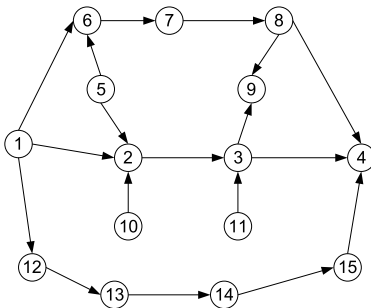


Fig. 2. Network topology used for testing

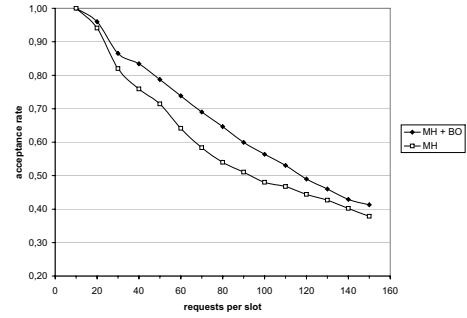


Fig. 3. Acceptance rates of min-hop routing with background optimization and sole min-hop routing for varied request arrival rates

generated in the course of our simulations to both brokers. In doing so, we are able to precisely quantify the effect of optimization with respect to the examined performance metrics.

In our experiments, we generated traffic using a constant demand matrix for the three commodities residing in the network with equal demand for all commodities, i.e., $d_{(1,4)} = d_{(5,9)} = d_{(10,11)}$. That means, on average we always generated the same number of reservation requests for all commodities within a fix time interval. The character of the offered load furthermore depends on a number of different parameters. These are given below with the reference setting in parentheses.

- the average number of requests arriving each time slot (50)
- the average reservation time of the requests in time slots (5 time slots)
- the average reservation duration in time slots (5 time slots)

Apart from the offered load, the framework-specific settings also influence the expected optimization benefit. In particular, these settings include

- the size of the notification interval (1 time slot)
- the temporal dimension, i.e., the length, of the time slots (5 s)
- the algorithm error margin value ϵ (0.01)

Recall that since the employed algorithm is $(1 - 3\epsilon)$ -approximate, this corresponds to an reoptimization solution within 3 % of the optimum.

Since *MH + BO* queues reservation requests which cannot be admitted via min-hop routing, there are always some requests that have neither been accepted nor declined at the time a test run comes to an end. Since it is not feasible to predict whether these pending requests will later be admitted, we continued the respective test until the final status of all these pending requests for *MH + BO* was known.

B. Summary

For the lowest setting, *MH + BO* and *MH* are both able to allocate resources for all conducted reservations, since there is sufficient bandwidth available to satisfy all requests on min-hop paths. With regard to the total acceptance rate, *MH + BO*

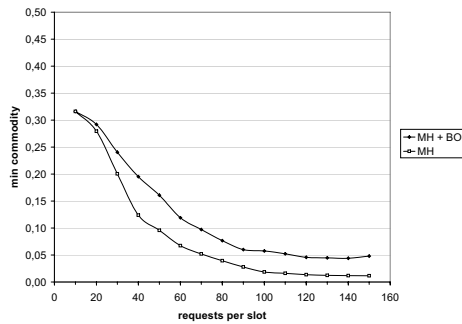


Fig. 4. Minimum acceptance rates over all commodities of min-hop routing with background optimization and sole min-hop routing for varied request arrival rates

outperforms MH as the load is continually increased. In the range of 70 to 80 requests per slot, MH + BO exhibits a serious advantage over MH with a more than 10 % higher acceptance rate (see Figure 3). As the number of reservation requests per slot grows very large, the acceptance rates converge at a lower level. This behavior can be attributed to the circumstance that the high average load on the network links does not allow for rerouting significant amounts of flow to free up stranded capacity.

For the minimum commodity acceptance rate in Figure 4, it can be observed that with increasing load this value decreases faster for MH than it does for MH + BO. For a mean 40 requests per slot, MH + BO can admit almost twice as many requests for commodity (10, 11) than MH. With further increasing load, MH can hardly admit any requests for this commodity; meanwhile MH + BO is able to constantly admit a small contingent up to four times larger than MH. We ascribe this to the interim optimizations that occur before the network is loaded to its full capacity which lessen the load on the bottleneck links and thus allow the unprivileged commodities to route more flow.

V. CONCLUSIONS AND OUTLOOK

We have established in this paper that after routing incoming advance reservation requests on min-hop paths, rerouting flow from more to less congested paths significantly increases the request acceptance rate of the network. However, the magnitude of the gain achieved is dependent on the characteristics of the network traffic. The obtained results indicate that the network resources are allocated in a more fair fashion to the network participants as a consequence of continuous optimization, i.e., the minimum acceptance rate for all source-sink router pairs creating demand in the network is also increased in the process. Thus, we observe that the devised advance reservation framework achieves its objectives. Depending on the characteristics of the offered load, acceptance rate gains in the magnitude of 10 % and more are feasible. Moreover, we observed that the applicability of optimization is independent of the traffic pattern with regard to average durations and arrival rates of reservation requests in the network but merely depends on the total offered load. The relevant results unfor-

tunately needed to be omitted from this paper due to space limitations. We have furthermore noticed that the marginal profits of optimization decreases as more resources in terms of computing time are provided; the same holds for longer mean reservation times of requests issued to the network. Finally, the attained optimization benefits can possibly be further maximized by setting the error margin of the algorithm to a value adequate to the encountered network traffic.

It is of great interest to further study the performance of background optimization in advance reservation environments. To this end, further experimental studies covering a wide range of network topologies are necessary. Instead of the randomly generated request sequences utilized in our experiments, these studies should preferably make use of real-world traffic data in order to reach conclusions about the performance in realistic network operation environments. Regarding the offered load, different traffic demand matrices should be employed and the capability of the optimizing algorithm to efficiently handle quickly changing demand matrices needs to be examined.

REFERENCES

- [1] B. B. Chen and P. Vicat-Blanc Primet, "A flexible bandwidth reservation framework for bulk data transfers in grid networks," LIP ENS Lyon, INRIA RESO - inria-00078069, Tech. Rep., June 2006.
- [2] L. Wu, J. Xing, C. Wu, and J. Cui, "An adaptive advance reservation mechanism for grid computing," in *PDCAT '05: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 400–403.
- [3] S. Kim and P. K. Varshney, "An adaptive advance reservation algorithm for QoS sensitive multimedia networks," in *Digital Wireless Communications VI. Edited by Rao, Raghuvveer M.; Dianat, Sohail A.; Zoltowski, Michael D. Proceedings of the SPIE, Volume 5440, pp. 339-346 (2004).*, ser. Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, R. M. Rao, S. A. Dianat, and M. D. Zoltowski, Eds., vol. 5440, Aug. 2004, pp. 339–346.
- [4] S. Ganguly, B. Nath, and N. Goyal, "Optimal bandwidth reservation schedule in cellular networks," *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 1591–1602, 2003.
- [5] R. Hutchens and S. Singh, "Bandwidth reservation strategies for mobility support of wireless connections with qos guarantees," *Aust. Comput. Sci. Commun.*, vol. 24, no. 1, pp. 119–128, 2002.
- [6] S. Krasser, H. Owen, J. Grimminger, H. Huth, and J. Sokol, "Distributed bandwidth reservation by probing for available bandwidth," *ICON 2003. The 11th IEEE International Conference on Networks 2003*, pp. 443–448, 2003.
- [7] S. Kar, M. Kodialam, and T. Lakshman, "Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications," in *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 12, Dec. 2000, pp. 2566–2579.
- [8] R. Bhatia, M. Kodialam, and T. Lakshman, "Fast network re-optimization schemes for MPLS and optical networks," *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, vol. 2707, pp. 249–265, 2003.
- [9] J. Könemann and N. Garg, "Faster and simpler algorithms for multi-commodity flow and other fractional packing problems," in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998, pp. 300–309.
- [10] L. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM Journal on Discrete Mathematics*, vol. 13, pp. 505–520, 1998.
- [11] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete Algorithms*, pp. 166–173, 2002.
- [12] G. Baier, E. Köhler, and M. Skutella, "The k-splittable flow problem," *Algorithmica*, vol. 42, no. 3-4, pp. 231–248, 2005.