Rerouting in advance reservation networks

Chongyang Xie\textsuperscript{a}, Hamed Alazemib, Nasir Ghania\textsuperscript{a,}\textsuperscript{c}
\textsuperscript{a}Department of Electrical and Computer Engineering, University of New Mexico, USA
\textsuperscript{b}Computer Engineering Department, Kuwait University, Kuwait

\textbf{A R T I C L E  I N F O}

\textbf{Article history:}
Received 19 April 2011
Received in revised form 21 February 2012
Accepted 23 February 2012
Available online 24 April 2012

\textbf{Keywords:}
Advance reservation
Rerouting
Bandwidth migration
Load-balancing

\textbf{A B S T R A C T}

The advance reservation of network connections is an area of growing interest and a range of service models and algorithms have been proposed to achieve various scheduling objectives, i.e., including optimization-based strategies and heuristic schemes. Now given the time-shifted nature of future requests, rerouting strategies have also been considered to improve resource allocation and carried loads. However, most existing rerouting schemes have focused on minimizing connection disruptions and have not considered further link load information. Along these lines, this paper develops novel rerouting strategies to improve connection scheduling in advance reservation networks. Specifically, a dynamic optimization formulation is presented to handle "on-line" arrivals along with a new heuristic load-balancing strategy. The performance of these proposed solutions is then evaluated for a wide range of network topologies and also compared against some existing rerouting schemes.

\textcopyright{} 2012 Elsevier B.V. All rights reserved.

1. Introduction

Advance reservation (AR) of user connections is becoming a key requirement in next-generation networks. As compared to regular immediate reservation (IR) provisioning, AR provides operators with a powerful tool to improve the usage of their deployed infrastructures and boost revenue potential. Now many users in the "e-science" field already require "pre-scheduled" network capacities to support applications such as large-scale data transfers, workflow process management, and distributed grid/cluster computing [1,2]. Here, given the massive increase in scientific data volumes, AR is particularly imperative as even the most scalable backbones may not be able to support all requests in an on-demand manner. The topic of AR scheduling has received much focus in recent years, with a range of algorithms studied for bandwidth-provisioning and optical wavelength routing networks [3–11]. Some of these studies have also considered broader protection/survivability issues [9] as well as flexible service models, e.g., such as variable start/stop times, variable capacity requirements, etc. [10,11]. Furthermore, researchers have also applied rerouting concepts (originally developed for IR networks [12–16]) to achieve improved resource distribution for AR demands [17–20]. These methods are particularly attractive since non-active future reservations can be moved in a non-disruptive manner. However, for the most part, existing AR rerouting schemes have used graph-based heuristic strategies and have focused on minimizing the disruption of existing reservations (caused by rerouting). Although this is a desirable objective, the further use of resource optimization techniques can be very beneficial here. For example, link load information can potentially be very beneficial for achieving improved network load distributions and thereby improving the number of AR requests scheduled in the network. Along these lines, this paper presents a more comprehensive look at AR rerouting, using formalized ILP methods to improve resource distribution. The key aim here is to incorporate link load information in order to increase network resource efficiencies and increase provisioning success rates. The paper is organized as follows. First, Section (2) presents a background survey of existing AR rerouting schemes. Section 3 then presents an ILP formulation, including a novel “dynamic” rerouting scheme to handle sequential “on-line” requests. Meanwhile a counterpart load-balancing rerouting heuristic is detailed in Section 4 and its run-time complexity analyzed. Finally, detailed performance evaluation results are presented in Section 5 for a range of network topologies and scenarios and comparisons made against some existing rerouting schemes. Overall conclusions and directions for future work are then presented in Section 6.

2. Background review

Rerouting schemes were initially studied to improve connection provisioning in IR settings, see [12–14]. However, the rerouting of active connections posed serious concerns for some operators, owing to the potential for data loss during switchover events [13,14]. As a result additional studies were done to apply rerouting in the broader IR survivability context, i.e., to improve service protection...
after link failures, see [15,16]. In general, most studies here have shown sizeable benefits for rerouting in IR settings, both in terms of bandwidth efficiency and reliability/availability, see [14]. In light of the above, researchers have also evaluated rerouting schemes for AR settings, i.e., to help improve resource distributions and free up capacity for future requests. These techniques are particularly amenable for AR since future reservations are inactive and can be rerouted without causing active service disruptions. These contributions are now surveyed.

Initial AR rerouting studies were done for survivable connection requests. For example, the work in [17] purposes a scheme to improve AR connection protection against link failures with unknown durations, i.e., as down-times may be difficult to predict beforehand. Specifically, this solution incorporates the current reservation states on the failed link and only reroutes a subset of the pending (i.e., admitted but inactive) reservations. The goal here is to avoid having to reroute all reservations on the failed link, particularly since link failures may not affect all AR schedules (pending recovery times). Overall simulations show notable reduction in the termination rates of pending reservations with this selective rerouting scheme, i.e., almost on par with competing strategies that reroute all failed link reservations. Meanwhile, [18] proposes an alternate strategy in which destination nodes initiate rerouting to reduce the signaling delays associated with source-initiated rerouting after link failures. Specifically, this scheme uses one-way RSVP-TE messaging to reroute paths and reduce overall service disruption times. Simulations show that the solution achieves higher rerouting success as well as lower service disruption times, i.e., whilst still maintaining utilization and blocking rates on par with those from two-way signaling schemes.

Other studies have also looked at AR rerouting for regular working connections, i.e., non-protection settings. Here, most solutions have implemented a “two-stage” arrivals-based approach in which rerouting is attempted after failure of a regular setup request (see also Fig. 1). For example, [19] proposes two rerouting solutions, one which tries to minimize the hop counts of the new incoming reservation and the other which tries to minimize the number of rerouted reservations themselves. These schemes utilize modified Dijkstra’s shortest path algorithms to compute a tentative path between the source-destination pair and then select admitted reservations along the path to reroute. Overall results show that both strategies can achieve sizeable gains over non-rerouting strategies using minimum hop-count rerouting metrics, i.e., up to 23% lower request blocking rates. However, the difference between the two rerouting algorithms is marginal, with the minimum hop scheme yielding slightly lower blocking rates. Meanwhile, [20] tables another algorithm that uses k-shortest path schemes. This is done by selecting a link that has the minimum link capacity from all the links that constitute the k pre-computed source-destination routes and then rerouting reservations along this link to vacate capacity. Specifically, the scheme selects reservations with the minimum bandwidth requirements, then reroutes them to links that are link-disjoint from the k paths. Simulation results for the NSFNET topology show that the proposed scheme yields almost 50% higher success rates versus non-rerouting approaches. The solution also improves bandwidth utilization by about 10%.

Finally, a “departure-based” AR rerouting scheme is also proposed in [21] to re-organize bandwidth allocations along the paths of departing reservations. Namely, this approach tries to re-optimize resource allocations after request departures, with the aim of re-distributing bandwidth utilization to achieve the maximum available bandwidth for next incoming request. Overall results show a notable increase in bandwidth utilization resulting from the re-optimization of vacated bandwidth of departing connections. In addition, the effectiveness of rerouting also increases with higher average node degrees, i.e., generally more effective in topologies with richer connectivity. Finally, [22] proposes a combined rerouting solution which attempts fast “on-line” admission of incoming requests as well as background path re-computation for existing reservations. This scheme uses hop-count based (re)routing, i.e., resource minimization, and unlike the above strategies, triggers rerouting even if there is sufficient capacity for an incoming AR request. Namely, background re-computation tries to re-organize the routes for a batch of reservations by shortening their aggregate hop counts. Overall findings show that the proposed scheme yields up to 60% lower blocking as compared to non-rerouting alternatives.

In summary, the above-detailed studies have yielded a good set of findings for AR rerouting. However, recent studies for regular (i.e., non-rerouting) AR scheduling have also shown the benefits of incorporating dynamic link load information into the provisioning process, see [23,24]. The general reasoning here is that more “resource-balanced” networks will be better able to support more incoming requests. Hence it is postulated that the use of such information in AR rerouting settings also has the potential to provide further performance gains. This forms the key motivation for the work herein.

3. Optimization formulations

The basic AR problem has been shown to be NP-complete [4], and earlier studies have developed a range of ILP models to achieve different optimization objectives, e.g., such as maximizing the number of admitted requests [10], minimizing resource utilization [25], etc. Now for the most part, all of these ILP schemes have assumed idealized settings in which operators have full a priori demand knowledge. Hence broadly speaking, these strategies can be construed as performing “full rerouting” across all input batch requests, and are therefore termed as global ILP (GILP) solutions herein. Nevertheless, in most real-world networks, AR demands tend to arrive in a sequential “on-line” manner, i.e., non a priori [1]. Hence the direct use of GILP solutions will be rather difficult here. Moreover, even in cases where all (or some) of the demands are known beforehand, globalized models will pose high computational complexities/intractabilities. Although this is a general limitation of most ILP schemes, these scalability concerns are further compounded in AR settings since related formulations discretize the time axis (and thereby introduce a whole new dimension of variables).

To resolve these challenges, a novel dynamic ILP (DILP) solution is proposed here for re-optimizing, i.e., rerouting, existing reservations in sequential “on-line” AR scenarios. This scheme basically
restricts optimization to a subset of time-overlapped demands at the arrival time of an incoming AR request, i.e., thereby precluding the need for full a priori knowledge. In addition, the proposed DILP solution also introduces a new objective function to achieve better load distribution. Now even though this “partial” ILP-based rerouting strategy may not match the performance of idealized GILP schemes with full a priori state, it will still provide a good bound on the achievable performance levels in realistic “on-line” settings.

The proposed DILP scheme is now detailed. Before doing so, however, a precursor GILP model is outlined to introduce the desired objective function (and also provide a baseline ILP scheme for comparison and benchmarking). Note that this framework only addresses regular bandwidth-provisioning networks, and added considerations for optical wavelength-routing networks are left for future study. In addition, all AR requests are assumed to be characterized using “static” parameters, i.e., fixed start/stop times and bandwidth requirements. Indeed, further studies can relax these considerations to incorporate more flexibility in both the time and bandwidth dimensions [4].

3.1. GILP model

A GILP model is first presented and then used to derive the subsequent DILP scheme. Consider the requisite notation here. The overall network is modeled as a graph topology, \( G(V,E) \), where \( V \) is the set of nodes and \( E \) the set of links. All links have fixed capacity, \( C \), and connectivity is assumed to be bi-directional, i.e., two uni-directional links in opposite directions between neighboring nodes. Akin to other ILP formulations for AR, time is discretized into time slots of fixed duration, thereby mandating all requests to start and stop at integral multiples thereof. Hence time can be denoted by the integral value of the time slot. Furthermore, assuming a static AR parameter model, the \( n \)th user request is denoted by the 5-tuple \( r^n = (s^n, d^n, t^n_0, t^n_i, b^n) \) comprising of the source node \( s^n \), the destination node \( d^n \), the start time slot \( t^n_0 \), the stop time slot \( t^n_f \), and the requested bandwidth \( b^n \leq C \). In addition several other variables are also defined as follows:

- \( R \) is the set of all AR requests.
- \( T \) is the current time slot in which the GILP is run.
- \( T_m \) = max \( \{t^n_0\} \) is the maximum stop time slot across all requests \( r^n \in R \).
- \( p^{n \in k} \) is a binary flag which denotes link occupation in time slot \( k \), i.e., \( p^{n \in k} = 1 \) if \( r^n \) does not use link \( e \in E \) at time slot \( k \).
- \( v \rightarrow e \) if \( v \in V \) is the ingress node of link \( e \in E \), \( e \rightarrow v \) if \( v \in V \) is the ingress node of link \( e \in E \).

Using the above notation, the overall GILP objective function is defined as:

\[
\text{minimize} \sum_{r^n \in R} \sum_{e \in E} \sum_{k \in [1,T_m]} b^n p^{n \in k}
\]

subject to the following constraints:

\[
\sum_{r^n \in R} p^{n \in k} = 1, \quad r^n \in R, \quad t^n_0 \leq k \leq t^n_f, \quad (2)
\]

\[
\sum_{r^n \in R} p^{n \in k} = 0, \quad r^n \in R, \quad t^n_0 \leq k \leq t^n_f, \quad (3)
\]

\[
\sum_{r^n \in R} p^{n \in k} = 1, \quad r^n \in R, \quad t^n_0 \leq k \leq t^n_f, \quad (4)
\]

\[
\sum_{r^n \in R} p^{n \in k} = 0, \quad r^n \in R, \quad t^n_0 \leq k \leq t^n_f, \quad (5)
\]

\[
\sum_{r^n \in R} p^{n \in k} = \sum_{v \in V} p^{n \in k} \quad r^n \in R, \quad t^n_0 \leq k \leq t^n_f, \quad \nu \notin \{s^n, d^n\}, \quad (6)
\]

\[
\sum_{v \in V} b^n p^{n \in k} \leq C, \quad e \in E, \quad T_m \leq k \leq T_m, \quad (7)
\]

\[
p^{n \in k+1} = p^{n \in k}, \quad r^n \in R, \quad e \in E, \quad t^n_0 \leq k \leq t^n_f, \quad (8)
\]

In particular, 1 tries to minimize the total resource “usage” across all scheduled demands. Namely this value is formulated as a sum of path length-bandwidth products. Meanwhile the subsequent constraints handle flow conservation, link capacity limitation, and route consistency respectively. Specifically, Eq. (2) (Eq. (3)) constrains outbound (inbound) traffic at the source node, whereas Eq. (4) (Eq. (5)) constrains inbound (outbound) traffic at the destination node. Meanwhile Eq. (6) conserves the transit traffic at intermediate nodes, i.e., non-source or non-destination nodes. Next, Eq. (7) limits the total provisioned bandwidth on a link to under its maximum capacity, C. Finally Eq. (8) ensures that a reservation adheres to its assigned route during its requested time interval.

Overall, the total variable count for the GILP formulation is quite significant. For example, consider a small 6-node mesh network fielding 20 AR requests with maximum holding times of 3 timeslots. The resulting number of variables here is upper-bounded by \( 6 \times 6 \times 20 \times (20 \times 3) = 43,200 \), i.e., without even considering the generalized case of time-overlapped requests. As a result the GILP model for this scenario will likely be intractable (unsolvable) on most modern servers.

3.2. DILP Model

The DILP scheme meanwhile implements optimization (rerouting) in “on-line” settings by triggering ILP computation upon request arrivals. The goal here is to re-optimize existing reservations in conjunction with the incoming request, and hence maintain a level of “optimality”. Now the overall DILP scheme re-uses the same objective function as the GILP model, but limits its application to a subset of inactive AR reservations whose intervals overlap with the incoming request. This choice helps preclude disruptions for active connections and drastically lowers the number of required ILP variables as well. This scheme is now presented.

The required notation for the proposed DILP scheme builds upon that used for the GILP in Section 3.1. Specifically, the following additional variables are defined:

- \( r' \) is the incoming AR request which triggers DILP.
- \( T_i \) is current time slot in which the DILP is triggered, i.e., when \( r' \) arrives
- \( R_{i}^{a} \) is the set of active reservations at time slot \( k \), i.e., \( r' \in R_k \) if and only if \( t'_i \leq k \leq t'_f \). These reservations will not be rerouted, i.e., no service disruptions.
- \( R_{i}^{b} \) is the set of admitted but inactive reservations at time slot \( k \), i.e., \( r' \in R_k \) if and only if \( t'_i > k \). These reservations are eligible for rerouting by DILP.
- \( T_{i} = \max_{r' \in R_{i}^{b} \cup (r')} t'_i \) is the maximum look-ahead time. Hence the DILP can limit optimization to the shorter interval \([T_i, T_{i}]\) as opposed to \([T, T_{a}]\) in GILP.

Note that the above defined look-ahead time, \( T_i \), will greatly reduce the number of variables needed in the ILP. Hence using this model, a revised objective function is defined as:

\[
\text{minimize} \sum_{r' \in R_{i}^{b} \cup (r')} \sum_{e \in E} \sum_{k \in [1,T_m]} b^n p^{n \in k}
\]

subject to the following constraints:

\[
\sum_{r' \in R_{i}^{b} \cup (r')} p^{n \in k} = 1, \quad r' \in R_{i}^{b} \cup (r'), \quad t'_i \leq k \leq t'_f, \quad (9)
\]

\[
\sum_{e \in E} p^{n \in k} = 0, \quad r' \in R_{i}^{b} \cup (r'), \quad t'_i \leq k \leq t'_f, \quad (10)
\]

\[
\sum_{r' \in R_{i}^{b} \cup (r')} p^{n \in k} = 1, \quad r' \in R_{i}^{b} \cup (r'), \quad t'_i \leq k \leq t'_f, \quad (11)
\]

\[
\sum_{e \in E} p^{n \in k} = 0, \quad r' \in R_{i}^{b} \cup (r'), \quad t'_i \leq k \leq t'_f, \quad (12)
\]
\[
\sum_{e \in \mathcal{E}} p_{n|e}^{\text{res}} = \sum_{e \in \mathcal{E}} p_{n|e}^{\text{req}} r_i^e \in R_{p}^e \cup \{r_i^e\}, \quad t_i^s < k \leq t_i^s, \quad v \neq (s^e, d^e) \quad (14)
\]

\[
\sum_{e \in \mathcal{E}} p_{n|e}^{\text{res}} = \sum_{e \in \mathcal{E}} p_{n|e}^{\text{req}} r_i^e \in R_{p}^e \cup \{r_i^e\}, \quad e \in \mathcal{E}, \quad t_i^e < k \leq t_i^e \quad (15)
\]

\[
p_{n|e}^{\text{res}} = p_{n|e}^{\text{req}} r_i^e \in R_{p}^e \cup \{r_i^e\}, \quad e \in \mathcal{E}, \quad t_i^e < k < t_i^e \quad (16)
\]

Overall, the above objective function in Eq. (10) is largely the same as that used for the GILP scheme, i.e., Eq. (1), with the exception that only inactive reservations in the sliding interval \([T_i, T_f]\) are considered. In addition, the respective constraints in Eqs. (6)–(8) mirror those for the GILP model in Eqs. (2)–(8), with appropriate modifications for localizing the timescale of the global optimization, i.e., from full time span \([T, T_m]\) to dynamic (i.e., partial) time span \([T_i, T_f]\).

Now reconsider the 6-node mesh network scenario discussed in Section 3.1, which fields 20 AR requests with maximum holding time of 3 time slots. Assuming a maximum of 3 inactive reservations in the interval \([T_i, T_f]\) for each incoming request, the total number of variables is bounded by \(6 \times 6 \times 3 \times (3 \times 5) = 1,620\). Indeed, this value is over an order magnitude lower than the variable count for the GILP scheme, and hence the modified DLILP will be much more tractable for current computing systems. However, larger sized networks (with many 10s of nodes) will still pose scalability concerns.

4. Load-balancing rerouting heuristic

ILP-based schemes generally pose very high computational complexities, particularly for larger networks with longer connection hold times (i.e., increased numbers of timeslots). To address these limitations, a novel “on-line” rerouting heuristic is now developed using graph-theoretic load-balancing techniques, termed as load-balancing rerouting (LB-R). The objective of this scheme is to lower request blocking by improving network resource distributions, i.e., following the same motivation as the ILP scheme. In particular, the proposed LB-R scheme uses dynamic AR link load information to perform connection (re) scheduling. Furthermore, the solution follows the same “two-step” strategy used in other AR rerouting schemes (as surveyed in Section 2), see also Fig. 1. Namely, the first non-rerouting stage attempts regular AR provisioning by distributing incoming requests across network links. Meanwhile the second rerouting stage (Stage 2) is triggered only upon failure of a regular setup request (Stage 1). Namely, this second stage tries to compute new routes (that can provide some partial level of capacity) for failed requests and then identifies a subset of admitted reservations for rerouting, i.e., termed as the rerouting candidate set (RCS). One of the key objectives in here is to improve rerouting effectiveness by carefully selecting the RCS to avoid excessive resource usages on detoured routes. Consider the details.

The heuristic AR solution re-uses much of the notation developed in Section 3 for ILP modeling. Namely network nodes/links are represented via the graph \(G(V, E)\) and the nth AR request is again given by the 5-tuple \(r_i^n = (s^e, d^e, t_i^e, t_i^e, b_i^n)\). In addition, a bandwidth availability function is also defined for each link \(e \in E\), i.e., \(b_i(t)\), as shown in Fig. 2. This function represents all current/future link resource levels and is derived by tracking all of the admitted reservations using the link over their respective requested time intervals. Finally, the RCS is denoted by the set \(S\). Using this notation, the detailed LB-R algorithm is now presented.

4.1. Non-rerouting load-balancing stage

The complete pseudo-code listing for the two-stage LB-R scheme is given in Fig. 3. Here the first non-rerouting stage is detailed in lines 2–4, and attempts to compute a “least-congested” path, \(P\), between the source node, \(s^e\), and destination, \(d^e\), with sufficient capacity, \(b^e\). Specifically, this is done by running a modified Dijkstra’s shortest-path algorithm over the link bandwidth-availability states in \(G(V, E)\), and only considering feasible links with sufficient capacity in the requested interval, i.e., for each link \(e \in E\), \(b_i(t) < t_i^e < t_i^e\). Furthermore, in order to compute the “least-congested” path, a dynamic load-based strategy is introduced here. Specifically, this scheme assigns link costs as a function of the minimum available bandwidth, i.e., bottleneck bandwidth, during the incoming request’s interval. Namely for a given link \(e\), this cost is given by:

\[
w_e = 1/b^e_{\text{min}},
\]

where

\[
b^e_{\text{min}} = \min_{t_i^e < t_i^e} \{ b_i(t) \}
\]

is the minimum bottleneck capacity of the link during the request interval \([t_i^e, t_i^e]\), see also Fig. 2. Note that this approach simply selects the “widest-path” based upon the bottleneck bandwidth value, and is formulated as an extension of similar strategies used in IR networks, i.e., \(1/b^e_{\text{min}}\), where \(b^e_{\text{min}}\) is the instantaneous available bandwidth and \(b\) is the total bandwidth on link \(e\), see \([21]\). Overall, if this path computation returns a feasible path, \(P\), the first stage is successfully completed (and subsequent setup signaling initiated to reserve the bandwidth for the incoming request \(r_i^n\)). Alternatively, the second rerouting stage is initiated, as detailed next.

4.2. Rerouting load-balancing stage

The second stage of the LB-R algorithm is also shown in Fig. 3 (from line 6 onwards) and implements the main rerouting functionality upon failure of a regular setup attempt. Specifically, this stage performs the following three steps:

1. Compute a candidate route \(P_c\) with a fraction \(\rho\) of requested bandwidth \(b^e\) in the desired interval, i.e., \(\rho b^e\) in \([t_i^e, t_i^e]\).
2. Construct a RCS \(S\) by selecting a subset of inactive AR reservations along \(P_c\) for rerouting.
3. Reroute all reservations in the computed RCS.

Now in general it is desirable to minimize the amount of capacity perturbed during rerouting. To exert better control over this process, a bandwidth reduction factor, \(\rho\), is defined here, i.e., \(0 \leq \rho \leq 1\). Specifically, if a given route can provide up to \(\rho b^e\) units of bandwidth, \(\rho\) is defined here, i.e., \(0 \leq \rho \leq 1\). Specifically, if a given route can provide up to \(\rho b^e\) units of bandwidth, \(\rho\) is defined here, i.e., \(0 \leq \rho \leq 1\). Specifically, if a given route can provide up to \(\rho b^e\) units of bandwidth, \(\rho\) is defined here, i.e., \(0 \leq \rho \leq 1\).
of capacity, then only \((1 - \rho)b_i\) units of capacity need to be rerouted from the links along this route during the interval \([t_i, t_i']\). Hence choosing a lower value of \(\rho\) implies a higher level of rerouting.

Now the actual computation of the candidate route is simply done by re-running the modified Dijkstra’s scheme from the first stage with a reduced request size of \(q_{b_i}\) (i.e., line 6, Fig. 3). If this computation is successful, the algorithm proceeds to the second rerouting step to compute the RCS (i.e., line 8, Fig. 3), otherwise the request is dropped (i.e., line 20, Fig. 3). In general, lower (higher) values of \(\rho\) will yield higher (lower) success rates for candidate route computation but more (less) rerouting disruptions. Hence careful selection of \(\rho\) is required in order to achieve a balance between rerouting overheads and request blocking rates.

Next, consider the second step of the rerouting stage, i.e., RCS selection. This step selects a subset of inactive future reservations along the candidate route, \(P_c\), for rerouting, and is detailed separately in Fig. 4 (called from Fig. 3, line 8). The overall objective here is to pick a minimum number of reservations for rerouting so as to limit the number of disruptions, but still extract the required \((1 - \rho)b_i\) of capacity along \(P_c\) (to be combined with the already-available \(q_{b_i}\) capacity). Hence the RCS selection algorithm in Fig. 4 simply loops through all links along the candidate route, and for each link, iteratively moves a sufficient number of inactive reservations in the interval \([t_i, t_i']\) to the RCS until the desired capacity \((1 - \rho)b_i\) is retrieved. In order to minimize the number of rerouted reservations here, the inactive link reservations are sorted by decreasing order of their respective bandwidth demands, \(b_i\) (i.e., line 4, Fig. 4).

Finally, the third and final step in the rerouting stage computes new (alternate) routes for all reservations in the RCS. This is also shown separately in Fig. 5 (and called from line 12 in Fig. 3). This algorithm loops through and processes all reservations in the RCS, and for each reservation, first frees up all of its link resources and then tries to compute another least-congested (load-balanced) path. If all RCS connections are successfully rerouted, the algorithm concludes with a successful setup, otherwise the request is failed. Note that this scheme tracks all rerouted connections in the RCS, and in case of any rerouting failure(s), restores them back to their original routes (i.e., lines 9–12, Fig. 5). This is necessary since all LB-R computations are done using the actual bandwidth availability graph, i.e., no temporary copies. This approach contrasts with many IR heuristics which generate full temporary copies of network resource graphs for intermediate computation purposes. Indeed, this duplication is not done here as it is infeasible in larger networks, i.e., owing to the high overheads associated with copying bandwidth availability state.

In summary, the LB-R algorithm leverages link load information in both of its stages, i.e., regular AR path computation (Section 4.1) as well as rerouting (Section 4.2). As such, the algorithm is designed to improve network link resource distributions and avoid excessive congestion on bottleneck links. Carefully note that additional improvements can also be devised here. For example, rerouting can also be done at departure times, akin to [21]. In addition, the actual order in which the RCS reservations are rerouted can be dynamically selected in order to achieve higher success rates. Finally, the RCS itself can be dynamically modified by replacing reservations which cannot be rerouted. However, all of these improvements are left for future study.

4.3. Complexity analysis

The space and run-time complexities of the proposed “on-line” LB-R scheme are now analyzed. In terms of the former, it is assumed that the LB-R implementation uses linked lists to track
reservations at each network link (the efficiency of using these data structures for temporal AR states is analyzed and justified in [26]). Hence if some reservations overlap in time at a given link, then parts of them can be merged into common reservation blocks. This implies that for $k$ reservations on a link, the maximum number of reservation blocks (for all possible overlap conditions) is $2^{k-1}$, i.e., linearly proportional to $k$. Hence assuming that the maximum number of AR connections in the network is bounded by the square of the number of nodes, $O(|V|^2)$, the resulting space complexity of the LB-R scheme algorithm is $O(|V||V|^2) = O(|V|^3)$. In general, however, the actual number of active connections a link will be well below this amount.

Meanwhile, for run-time complexity, most graph-theoretic algorithms take polynomial time. For example, optimized versions of Dijkstra’s shortest-path algorithm are generally of $O(|E| + |V| \log |V|)$ complexity [27], i.e., for $|V|$ nodes and $|E|$ links. Hence if the first non-rerouting stage is successful (line 4, Fig. 3), the resultant complexity is given by $O(|E| + |V| \log |V|)$. However, if this initial stage is unsuccessful, the second rerouting stage is invoked. The computation of the candidate route (i.e., first rerouting step) again takes a similar amount of time, bounded by $O(|E| + |V| \log |V|)$. Meanwhile, the selection of the RCS (i.e., second rerouting step, Fig. 4) involves traversing the candidate route $P_c$ and selecting reservations for the RCS. Now in the worst case
the candidate route may traverse all network nodes, i.e., \( O(|E|) \), and each link may have up to \( O(|V|) \) reservations. Hence sorting the reservations in decreasing order of bandwidth entails \( O(|E||V|^2 \log(|V|)) \) complexity, which is also the complexity for RCS selection. Finally, RCS rerouting phase (Fig. 5) adds \( O(|E| + |V| \log(|V|)) \) complexity for each connection in the RCS. This yields a total complexity of \( O(|V|^2 |E| + |V|^3 \log(|V|)) \), and also represents the upper bound for the LB-R heuristic, i.e., polynomial time.

### 5. Performance evaluation

The performance of the proposed ILP and heuristic rerouting schemes is now analyzed. In particular, the GILP scheme is evaluated using the \( \text{lp}\_\text{solve} \) ILP package, whereas the counterpart LB-R heuristic is evaluated using custom-developed discrete event simulation models in \textit{OPNET Modeler}\textsuperscript{TM}. Meanwhile, the adapted DILP scheme is analyzed using a combination of simulation and ILP techniques, i.e., dynamic run-time calls are made from the \textit{OPNET Modeler}\textsuperscript{TM} tool (upon request arrivals) to the external \( \text{lp}\_\text{solve} \) module.

Now in order to account for the scalability limitations of the various schemes, a range of network topologies are tested. Specifically, two smaller-sized topologies are first used for evaluating all schemes, and in particular the ILP-based GILP and DILP strategies. These networks include a 4-node topology, Fig. 6a, and an 8-node topology, Fig. 6b. Meanwhile, two larger topologies are also used to test the LB-R heuristic. These networks include the well-known 16-node NSFNET topology, Fig. 7a and a denser 27-node \textit{Deutsche Telekom} (DT) network, Fig. 7b. In all of these scenarios, the network elements are assumed to be generic IP/MPLS routers running 10 Gbps links. Furthermore, all requests sizes are uniformly varied from 200 Mbps to 1 Gbps in increments of 200 Mbps, corresponding to fractional Ethernet demands. Meanwhile, all request book-ahead intervals, arrival intervals, and holding times are exponentially-distributed.

Initial simulations are run for the proposed LB-R scheme to gauge the sensitivity of its bandwidth reduction factor, \( \rho \), and select a suitable value for subsequent comparison tests. Namely, the bandwidth blocking rate (BBR) results for the NSFNET topology are measured for three \( \rho \) values (0.1, 0.5, and 0.9) in Fig. 8 and the corresponding average path lengths (utilizations) are also shown in Fig. 9. Note that the BBR is computed as the ratio of bandwidth of all blocked AR requests over the bandwidth of all AR requests. This metric provides a more accurate “capacity-based” blocking rate measure, i.e., versus pure connection blocking probabilities. Overall, these BBR results indicate that small values of \( \rho \) (i.e., 0.1) give the lowest blocking, averaging 11–32% lower than those with larger values (i.e., \( \rho = 0.9 \)) at light-to-median loads. However these values also give slightly higher resource utilizations at higher loads, i.e., by about 5%. In addition, counterpart sensitivity tests with the DT topology (not shown) also yield similar relative performances between these \( \rho \) values. Hence a median value of \( \rho = 0.5 \) is chosen for all subsequent tests in order to maintain a balance between rerouting overheads and BBR reduction.

Next, detailed studies are conducted to evaluate the proposed ILP and heuristic strategies against existing schemes, both non-rerouting and rerouting. In particular, several candidate algorithms are considered here. First, the simple non-rerouting minimum-hop count heuristic is used from the \textit{Energy Sciences Network} (ESnet) OSCARS control plane [8], termed here as \textit{hop-count} (HC) routing. This algorithm basically runs a modified Dijkstra’s shortest path algorithm to compute feasible minimum hop count routes between source and destination nodes, i.e., breadth-first search with pruning of bandwidth-deficient links in requested interval. Next, the \textit{non-rerouting} part of the LB-R scheme (i.e., Section 4.1) is also tested to measure the performance of pure load-balancing traffic engineering, termed here as \textit{load-balancing} routing (LR). Finally, some existing AR rerouting schemes are also considered. Specifically, as per Section 2, most such algorithms pursue one of two strategies, i.e., minimizing path hop-counts or minimizing the number of rerouted reservations. However, since findings have shown that these two strategies yield very similar results, only the former is tested herein as it gives slightly lower blocking [19]. This minimum hop rerouting approach is termed as \textit{hop-count rerouting} (HC-R). Carefully note that the original algorithm in [19] is designed for lightpath rerouting in optical DWDM networks. As a result, this scheme is adapted for bandwidth-only settings by omitting the wavelength selection functionality. Finally the AR rerouting scheme in [20] is also tested here. This solution randomly selects a route from the \( k \)-shortest paths between the source and

![Fig. 6. Small network test topologies.](image)

(a) 4-node topology (6 links)

(b) 8-node topology (13 links)

![Fig. 7. Large network test topologies.](image)

(a) NSFNET backbone topology (16 nodes, 25 links)

(b) DT topology (27 nodes, 52 links)
destination nodes and then reroutes a sufficient number of admitted reservations along this route, termed here as \textit{k-shortest route rerouting} (KR-R).

Detailed comparison runs are first presented for the simple 4-node topology. Now since the ILP formulations in Section 3 mandate \textit{slotted} timelines, all arrival and holding times here are selected as integral multiples of a timeslot value. Specifically, request holding and bookahead times are set to a mean of 3 timeslots (i.e., exponentially distributed about 3 and then rounded to the nearest integer). The corresponding mean inter-arrival times are also exponential and adjusted according to input load. Furthermore, for scalability purposes, the total number of AR requests is limited to 50. Here these requests are presented all at once to the GILP scheme, but in a sequential manner to the DILP and all other heuristic schemes. Note that “global” ILP-based models are not directly amenable to measuring blocking rates, i.e., as they process full a priori demand sets and simply return either success or failure for the whole set. Hence in order to compute meaningful GILP blocking rates, multiple random permutations of the a priori demand set are measured for success/failure, i.e., up to 1000 (incurring huge computational complexities).

The overall BBR results for the various schemes are shown in Fig. 10, and clearly indicate that the idealized GILP solution achieves the lowest blocking of all, almost several factors lower than the basic HC heuristic. Note that results for the KR-R scheme are not presented here as they largely match those for the HC-R scheme. Meanwhile the DILP scheme achieves a good tradeoff between the GILP and heuristic strategies, and generally provides a much tighter theoretical bound as compared to all other graph-based strategies. Furthermore, the proposed LB-R solution gives the closest performance to the ILP-based schemes out of all of the heuristic schemes. The average path lengths are also shown in Fig. 11 to gauge overall resource usages. Again these findings show the highest efficiencies with the ILP-based schemes, about 10–15% lower than the heuristic strategies, e.g., HC, LR, HC-R and LB-R. Note that the plots in Figs. 10 and 11 exhibit some slight dips/fluxuations. In general, these variations occur due to the fact that ILP runs (and by extension their comparative heuristic runs) can only accommodate relatively smaller batch sizes, i.e., in the few 10 s of requests. As a result, many independent ILP runs are needed to generate smoother averages (with each taking up to 10 h or more). As this is difficult to do, the above plots are generated by averaging over 5 runs per point, and the overall relative performance between the schemes is consistent. Finally, the runtime overheads of the schemes are also shown in Table 1, including the average “Single Pass” durations for single optimization runs (or one heuristic path computation) and total “All Pass” durations for all 50 AR requests. As expected, the GILP scheme has prohibitive computational complexity, even for this small network. By contrast the DILP scheme is much more feasible, with associated run-times well within the acceptable ranges for operational “on-line” scenarios.

The 8-node network is evaluated next and its BBR results are shown in Fig. 12. Note that results for the GILP scheme are not presented here as associated run-times are excessive and do not yield convergence even after several days. Again, these findings show sizeable BBR reductions with the DILP scheme, i.e., averaging about 20–30% lower than the non-rerouting HC and LB heuristics and also the HC-R scheme of [19]. By contrast, the proposed LB-R scheme does the best out of all the heuristics, averaging about 5% lower BBR than the other algorithms. Related path lengths are also plotted in Fig. 13 and show the lowest utilizations (highest efficiencies) with the proposed DILP scheme. By contrast the LB-R scheme tends to give slightly higher path lengths, about 6% higher than DILP, as it chooses routes in an “non-optimal” on-line manner. The run-time overheads of these schemes are also given in Table 2, and show notable increases for the DILP scheme “All Pass” runs, e.g., minutes. Nevertheless, this dynamic optimization scheme is still quite amenable to “on-line” operation, as it yields “One Pass” durations in the 10s of second range.

Finally the various heuristics are also tested for the two larger topologies shown in Fig. 7. Now since ILP computations do not
Hence the average request hold-time (and bookahead) durations are now increased to 300 s (exponential) and the inter-arrival times varied per load. Also, each run is now averaged over 5,000,000 random sequential AR requests. The overall BBR performances for these scenarios are plotted in Fig. 14 (NSFNET) and Fig. 15 (DT) and reveal some key insights. Foremost, the proposed LB-R scheme consistently gives the lowest blocking, with the HC-R and KR-R schemes giving almost identical performances. In particular, rerouting is seen to be more effective in the lower node-degree NSFNET topology, i.e., LB-R gives about 18% lower blocking than HC-R in NSFNET and 12% lower in the DT network. In addition, the “non-rerouting” LR scheme (Section 4.1) notably outperforms the non-rerouting HC scheme and closely matches the HC-R rerouting scheme for the NSFNET topology, i.e. within 5% BBR (Fig. 14). This is a key finding and indicates that a simple change in link load metric, i.e., Eq. (17), can give sizeable blocking reductions in networks with lower connectivity levels.

Meanwhile the average path lengths for the various heuristic schemes are also plotted in Fig. 16 (NSFNET) and Fig. 17 (DT). As per earlier findings in Figs. 11 (4-node) and 13 (8-node), these results confirm that the minimum hop heuristics (HC, HC-R) tend to give the lowest utilization at lower loads, i.e., under 1% BBR. Conversely the rerouting schemes tend to drive up average utilization at higher loads regardless of path selection strategies (minimum hop or load-balancing). In particular, the KR-R scheme gives the longest routes across all loads. Nevertheless the maximum increases here are still bounded to under 11% of the base HC (minimum hop) scheme for both network topologies.

Tests are also done to compare the actual rerouting performances of all the heuristic strategies, i.e., HC-R, KR-R, and the proposed LB-R scheme. First, the respective rerouting success rates are plotted for the NSFNET and DT topologies in Figs. 18 and 19, respectively. In general, these results show that the HC-R and

<table>
<thead>
<tr>
<th>Load (Erlang)</th>
<th>25</th>
<th>35</th>
<th>45</th>
<th>55</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBR (%)</td>
<td>0.001</td>
<td>0.01</td>
<td>0.1</td>
<td>0.001</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load (Erlang)</th>
<th>25</th>
<th>35</th>
<th>45</th>
<th>55</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBR (%)</td>
<td>0.0001</td>
<td>0.001</td>
<td>0.01</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load (Erlang)</th>
<th>50</th>
<th>70</th>
<th>90</th>
<th>110</th>
<th>130</th>
<th>150</th>
<th>170</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBR (%)</td>
<td>0.00001</td>
<td>0.0001</td>
<td>0.001</td>
<td>0.01</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Fig. 14. BBR results for NSFNET network.
LB-R heuristics give sizeably better success rates than the KR-R approach. Meanwhile, the proposed load-balancing LB-R scheme gives slightly higher success rates than the HC-R scheme. For example in the denser DT network, the LB-R scheme averages about 5% higher success than HC-R (Fig. 19). Meanwhile, in the NSFNET topology the HC-R scheme gives slightly higher success rates in lower load regimes (Fig. 18).
Finally, rerouting overheads are also gauged by measuring the average rerouting "loads". Specifically, this is done by taking the total number of rerouted connections (during the whole simulation) and dividing by the simulation run time, i.e., effectively computing the average number of rerouted connections per second. Hence longer intervals here imply reduced overheads and vice versa. Now the related results are plotted in Fig. 20 (NSFNET) and Fig. 21 (DT), and clearly indicate the lowest overhead rates with the proposed load-balancing LB-R scheme. For example commensurate overheads with the HC-R scheme are up to 130% higher in NSFNET and 25% more in DT (at low-medium range input loads). Although the KR-R scheme does better than the HC-R solution, it still gives about 10–15% higher rerouting loads than LB-R. Overall, these findings indicate that load-balancing rerouting can also yield notably lower run-time complexity versus existing strategies.

6. Conclusion and future work

This paper develops novel solutions for rerouting in advance reservation bandwidth networks. Specifically, a dynamic ILP re-optimization scheme is presented to handle "on-line" request arrivals and achieve load-distribution across network links. In addition a load-balancing scheduling heuristic is also proposed to resolve the high computational overheads associated with the above ILP schemes. Here complete algorithmic specifications are presented and the corresponding temporal and space complexities analyzed. Detailed performance analyses are then conducted to gauge the performance of the proposed ILP and heuristic solutions versus existing rerouting and non-rerouting strategies. Overall results show that the ILP-based schemes give the lowest blocking, but are only scalable to very small network sizes, i.e., 4–8 nodes. Meanwhile the load-balancing heuristic achieves the tightest bounds on ILP blocking rates and outperforms all other rerouting strategies evaluated. Albeit related path length utilization is slightly higher (by about 5%), this heuristic also gives the lowest operational overheads in terms of disrupted reservations. Future efforts will look at extending this work along several key directions. Foremost, modified heuristic schemes will be developed using alternate rerouting triggering policies, i.e., at connection departures or intermittent intervals, etc. Furthermore, survivability considerations will also be treated using path-pair rerouting strategies, i.e., including dynamic ILP formulations.

Acknowledgment

This research has been supported in part by the United States Department of Energy Office of Science under Award #ER25828. The authors are very grateful for this support.

References