Load Balancing in WDM Networks through Adaptive Routing Table Changes

Mauro Brunato[†] Roberto Battiti[*] Elio Salvadori[*]

March 5, 2002

Abstract

In this paper we develop a Load Balancing algorithm for IP-based Optical Networks. The considered networks are based on a routing protocol where the next hop at a given node depends only on the destination of the communication. Our algorithm (RSNE - Reverse Subtree Neighborhood Exploration) performs at each iteration a basic change of a single entry in a routing table in order to minimize the disruption of the network.

We study the performance of our algorithm in realistic networks under static and dynamic traffic scenarios. Simulation results show a rapid reduction of the congestion for static networks and a performance of the incremental scheme while tracking a changing traffic matrix comparable to the complete reoptimization of the traffic.

Keywords: WDM, load balancing, local search, dynamic traffic.

1 Introduction

Wavelength Division Multiplexing (WDM) and Generalized Multi Protocol Label Switching (G-MPLS) have been proposed to support the growing bandwidth demand caused by the exponential Internet growth and to permit suitable traffic engineering. In WDM networks, a wavelength is assigned to each connection in such a way that all traffic is handled in the optical domain, without any electrical processing on transmission [1]. The established lightpaths form the virtual or logical topology, opposed to the network physical topology composed of nodes (Optical Cross-Connects - OXCs) and fibers.

Current advances in optical communication technology are rapidly leading to flexible, highly configurable optical networks. The near future will see a migration from the current static wavelength-based control and operation to more dynamic IP-oriented routing and resource management schemes. Future optical networks designs should probably be based on fast circuit switching, in which end-to-end optical pipes are dynamically created and torn down by means of signaling protocols and fast resource allocation algorithms [5].

IP modifications are being proposed to take QoS requirements into account and to integrate the IP protocol within the optical layer. At the same time, a generalized version of Multi-protocol Label Switching (G-MPLS) is currently being developed to

[*]Università di Trento, Dipartimento di Informatica e Telecomunicazioni, via Sommarive 14, I-38050 Pantè di Povo (TN), Italy; email: battiti|brunato|salvador@science.unitn.it

[†]Corresponding Author.

enable fast switching of various type of connections, including lightpaths. As soon as protocol modifications can ensure different QoS levels at the IP level, more and more statically allocated traffic can be transmitted on the dynamic portion of the network leading to an all optical and fully dynamic G-MPLS controlled optical cloud [12]. In this scenario it is necessary to study the impact of routing mechanisms typical to the IP world.

The basic motivation behind Load Balancing in computer networks is to reduce the congestion in the network. Congestion is related to delays in packet switching networks, and therefore reducing congestion implies better quality of service guarantees. In networks based on circuit switching (see for example the G-MLPS protocol), reducing congestion implies that a certain number of spare wavelengths are available on every link to accommodate future connection requests or to maintain the capability to react to faults in restoration schemes. In addition, reducing congestion means reducing the maximum traffic load on the electronic routers connected to the fibers.

Load balancing leads to the problem of creating virtual connections by considering both routing and wavelength assignment. The routing problem has its origin at the beginning of networking research (see [9] for a review of previous approaches to the problem). In particular adaptive routing, which incorporates network state information into the routing decision, is considered in [8] in the context of all-optical networks, while previous work on state-dependent routing with trunk reservation in traditional telecommunications networks is considered in [7]. It is also known that flow deviation methods [2], although computationally demanding, can be used to find the optimal routing that minimizes the maximum link load for a given network topology.

Because global changes of the logical topology and/or routing scheme can be disruptive to the network, we consider algorithms that are based on a sequence of small steps (i.e., on local search from a given configuration). In [4] "branch exchange" sequences are considered in order to reach an optimal logical configuration in small steps, upper and lower bounds for minimum congestion routing are studied in [13], where variable depth local search and simulated annealing strategies are also proposed. Strategies based on small changes at regular intervals are proposed in [9].

Our technological context is that of dynamic lightpath establishment in wavelength-routed networks reviewed in [14]. We therefore assume a mechanism to assign resources to connection requests, that must be able to select routes, assign wavelengths and configure the appropriate logical switches, see also [3] for integrated IP and wavelength routing and [6] for a blocking analysis in the context of *destination initiated reservation*.

This paper describes a preliminary investigation on protocols that consider IP-like routing strategies, where the next hop at a given node is decided only by the destination of the communication. In particular, we consider a basic change in the network that affects a single entry in the routing table of one node. In the context of all-optical networks this is relevant for optical packet switching networks, or for circuit switching networks (e.g. based on G-MPLS) where the optical cross-connects allow arbitrary wavelength conversion. The focus of this work is to study basic mechanisms in a simplified context. We plan to extend the work in the future by considering more general routing mechanisms (label switched paths in G-MPLS) and limited or no wavelength conversion.

Let us introduce the terminology that shall be used throughout this work. A *routing table* is an array, associated to each node of the network, containing next-hop information required for routing. The *traffic pattern* is available as an $N \times N$ matrix (N being the number of nodes in the network) $T = (t_{ij})$ where t_{ij} denotes the number

of lightpaths (or the number of traffic load units) required from node i to node j. We assume that the entries t_{ij} are non-negative integers and $t_{ii} = 0$ for all i. Given a traffic pattern and a routing table on each node, the sum of the number of lightpaths passing through each link is called the *virtual load* of the link. Finally, the maximum virtual load along a path is called the *congestion* of the path. The maximum virtual load on the whole network is called the *congestion* of the network.

The Load Balancing problem is defined as follows.

> LOAD BALANCING — Given a physical network with the link costs and the traffic requirements between every source-destination pair (number of lightpaths required), find a routing of the lightpaths for the network with least congestion.

In the following sections, first we introduce the Reverse Subtree Neighborhood Exploration (RSNE) algorithm in Sect. 2 and then discuss the implementation of an incremental version (I-RSNE) in Sect. 3. Finally Sect. 4 analyses simulation results, by considering both the static and the dynamic traffic cases.

2 Local Search for the Load Balancing Problem

In this paper we propose a new scheme based on a simple Local Search heuristic, the *Reverse Subtree Neighborhood Exploration* (RSNE). The basic idea behind this scheme is the following: start by setting a shortest path routing, then — iteratively — try to minimize the congestion of the network by rerouting part of the traffic passing through the most congested link in the network. Rerouting is not necessarily performed at the ingress node of the congested link, as all nodes lying on routes that pass through the congested link (the *upstream nodes*) shall be considered by the algorithm for a possible change of their routing tables.

Refer to Fig. 1 for the following explanation. Consider the simplified hypothesis of a network with a unique most congested link as depicted in the upper part of the figure: we can identify the congested link with its endpoints (*cFrom*, *cTo*). In this special case there are six lightpaths crossing that link, three of them coming from node src_a, one coming from src_b and two from src_c. Three lightpaths are directed to destination node $dest_1$, all others to $dest_2$.

A first approach to reduce the load on the congested link is to consider one of the destination nodes (e.g. $dest_2$) and reroute part of the load addressed to it from the congested link to some other neighbor nb_i of $cFrom$, provided that the new route does not end up in a cycle and that the congested link is avoided. This move is achieved by modifying only one single entry of the $cFrom$ routing table (see it on the upper right side of Fig. 1), e.g. from cTo to nb_2. In this example, three lightpaths are removed from the congested link $(cFrom, cTo)$ and are rerouted through the link $(cFrom, nb_2)$. All destinations and all neighbors of $cFrom$ are considered before choosing the actual routing table entry to change and its new value. This allows to choose the best option. Actually, we found (as pointed out in Sect. 4) that even if the best possible move increases the congestion there is still reason to choose it, because further improvement could arise in the following steps. The algorithm stops when a predetermined number of iterations has been performed, or when all possible moves end up with a nonconsistent routing table (one causing loops or disconnected node pairs). The approach just described is called *Reduced Neighborhood Exploration* (RNE); we call it *reduced* to put it in contrast with the following extension.

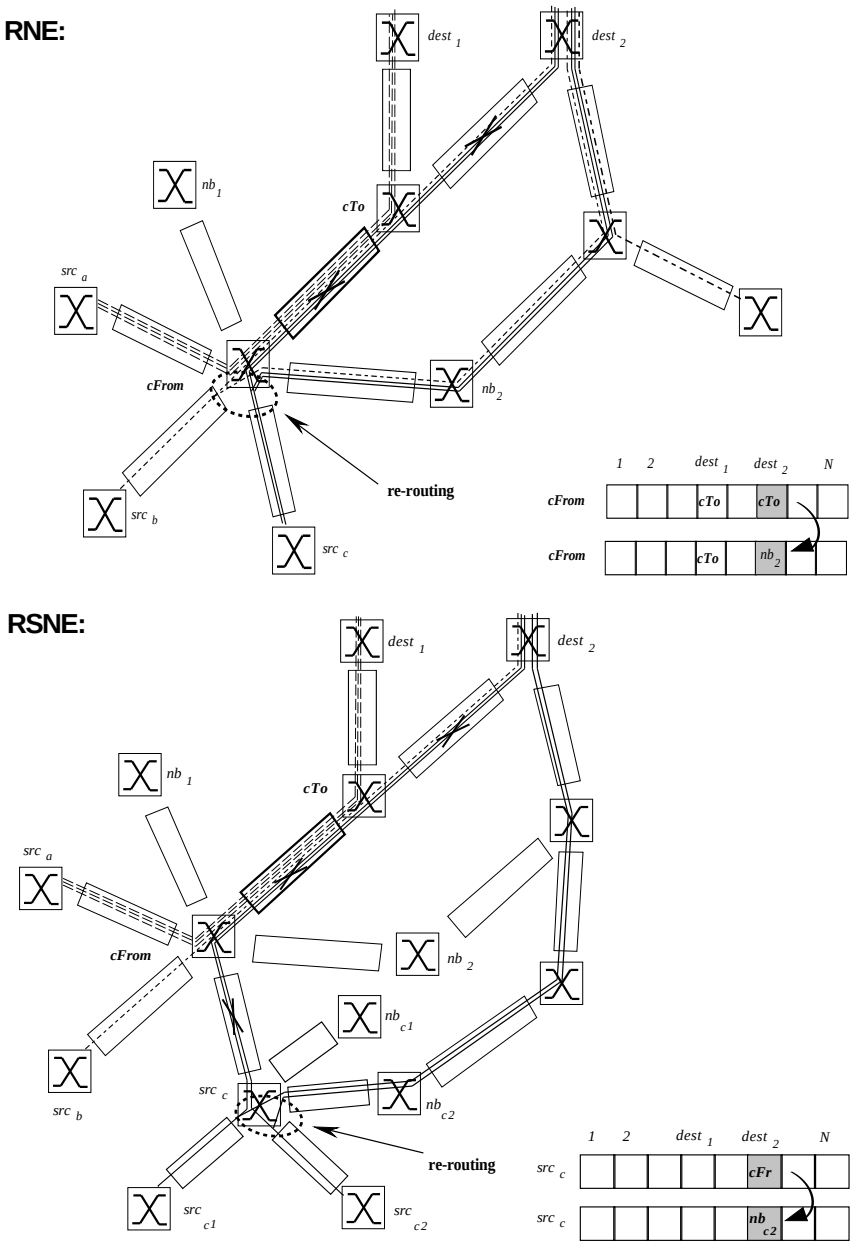

Figure 1: *Restricted Neighborhood Exploration* (RNE) and *Reverse Subtree Neighborhood Exploration* (RSNE).

```
1.  rTable ← shortestPathRouting(network)
2.  <congestion,congestedLinkSet> ← calculateLoad(network,traffic,rTable)
3.  repeat
4.      bestCandidateLoad ← +∞
5.      candidateMoveSet ← ∅
6.      for each link <cFrom,cTo> ∈ congestedLinkSet
7.          for each destination node dest such that rTable[cFrom][dest]=cTo
8.              for each node src ∈ routingTree(dest,cFrom)
9.                  removePartialLoad (src, dest)
10.                 for each neighbor node nb ∈ neighborhood(src)
11.                     vl ← virtual load on the candidate path from nb to dest
12.                     if (vl = bestCandidateLoad)
13.                         candidateMoveSet ← candidateMoveSet ∪ {<src,dest,nb>}
14.                     else if (vl < bestCandidateLoad)
15.                         bestCandidateLoad ← vl
16.                         candidateMoveSet ← {<src,dest,nb>}
17.                 restorePartialLoad (src, dest)
18.     if ( candidateMoveSet ≠ ∅ )
19.         <src,dest,nb> ← pickRandomElement ( candidateMoveSet )
20.         rTable[src][dest] ← nb
21.         <congestion,congestedLinkSet> ← calculateLoad(network,traffic,rTable)
22.     else exit
23. until MAXITER iterations have been performed
```

Figure 2: the Local Search RSNE algorithm

Consider now the lower part of Fig. 1, which reproduces a larger portion of the same graph. To reroute part of the load addressed to, e.g., $dest_2$ and crossing the congested link $(cFrom, cTo)$ the search may be extended to all upstream nodes whose routes to $dest_2$ cross the congested link. The routing is destination-driven, therefore one can always identify the tree composed of all the links lying on lightpaths to a specified destination $dest_i$. It is straightforward to get the subtree rooted in $cFrom$ and composed of all the links lying on lightpaths to node $dest_2$: in Fig. 1 it is identified by nodes src_b, src_c, src_{c1}, src_{c2}. In this case, taking into consideration one of the nodes composing this subtree (e.g. src_c), we could try to reroute part of the load on the most congested link towards some of the downstream src_c's neighbors nodes nb_{ci}, while avoiding cycles and the use of the congested link. This local move is realized again modifying one single entry of the node's routing table (see it on the lower right side of Fig. 1, e.g. from $cFrom$ to nb_{c2}). In this case, only two lightpaths are removed from $(cFrom, cTo)$ by sending them through an alternate path to $dest_2$. Even though the improvement is smaller than in the previous case, where only the neighbors of $cFrom$ were considered, we shall see in Sect. 4 that, by allowing such fine-grain variations, this more general scheme achieves much better results. Again, all possible moves are considered before choosing a routing table change. This implies scanning all possible destination nodes having $(cFrom, cTo)$ in their routing tree and, for each destination node, all neighbors of every upstream node of $cFrom$. Even congestion increases are accepted, if no improving option is found. This technique is called *Reverse Subtree Neighborhood Exploration* (RSNE).

Fig. 2 shows an outline of our Local Search algorithm used for the Load Balancing problem: the initialization section (lines 1–2) starts by generating the routing tables

through the application of the Shortest Path Routing algorithm to the specific network. By using the function *calculateLoad* we initially calculate the load on each link of the network, the initial value of *congestion* (from which the local search algorithm starts its search of the minimum) and the set of congested links.

The rest of the algorithm is a loop (lines 3-23) containing the local search algorithm.

The functions, variables and data structures used throughout this block have the following meaning:

- The set *candidateMoveSet* contains all candidate routing table changes. Its elements are triplets whose components are the node whose table must be changed, the index of the entry and the value that replaces the one already present.
- The function *routingTree*(*d*,*r*) returns the subtree that contains the nodes whose communications directed to destination *d* pass through node *r*.
- The function *shortestPathRouting*(*network*) calculates the shortest path tree for each destination node and returns the corresponding routing table as a matrix.
- The vector *rTable*[*n*] is the routing table of node *n*, whose *i*-th entry *rTable*[*n*][*i*] is the next-hop node index for lightpaths passing through node *n* and with destination *i*.
- Finally, the function *calculateLoad*(*network*,*traffic*,*rTable*) returns the network congestion given the network topology, the traffic pattern and the current routing scheme. The function also returns the set of links having maximum loads (*congestedLinkSet*).

The *candidateMoveSet* is empty at the beginning of each iteration. The local search algorithm (lines 3–23) consists of two parts. First, a set of alternative paths for some of the lightpaths passing through the most congested link is found (lines 6–17); in the second part (lines 18–22) a candidate is chosen and the corresponding routing table change is applied.

The first part (lines 6–17) includes the core of our proposal. The algorithm considers each congested link in *congestedLinkSet* (loop at lines 6-17). Then it iterates through all the routes using that link, identified by its endpoints (*cFrom*,*cTo*). Two nested loops are used: the first (line 7) scans the routing table of node *cFrom* looking for all destination nodes *dest* using that link; the second (line 8) scans all nodes *src* whose lightpaths directed to *dest* run through *cFrom*. These nodes identify the subtree rooted in *cFrom* of the routing tree having destination *dest*.

For each (*src*,*dest*) pair whose lightpaths go through the link (*cFrom*,*cTo*), the algorithm tries to reroute the lightpaths by altering the routing table in *src*. The corresponding load is temporarily removed from the current route (line 9), then an iteration through all neighbors *nb* of *src* calculates the maximum load that would be caused by rerouting the lightpath, provided that the new route does not end up in a cycle and that the congested edge is avoided. The best alternate paths, in terms of maximum load, are collected into *candidateMoveSet*. In particular, the current minimum is stored in *bestCandidateLoad*. If the load obtained after this traffic re-routing is equal to *bestCandidateLoad*, then the re-route is added to the candidate set (lines 12-13); if it is smaller, the candidate set is re-initialized to the current re-route and its load is stored as the new best value (lines 14-16). At the end of the alternate paths search, the partial load associated to the path originating in *src* and terminating in *dest* is reallocated (line 17) in order to allow the search of new paths with different initial nodes *src* (line 8).

In the second part of the RSNE algorithm, if the resulting set *candidateMoveSet* is not empty then one random element is selected from it (line 19), and the routing table of the network is updated (line 20). Finally, a new value of *congestion* and the corresponding set of most loaded links *congestedLinkSet* is calculated again in order to start a new search of alternate paths through the network.

Note that the local search algorithm continues looking for better values of *congestion* until the set of candidate re-routes *candidateMoveSet* is empty (line 22), or until a given number of iterations MAXITER has been performed (line 23).

From this algorithm we can easily obtain the RNE version: in this scheme, node *cFrom* is the only candidate for routing table modifications. This corresponds to the elimination of the loop structure on line 8, which scans the *cFrom*-rooted subtree, by setting *src* equal to *cFrom*. The rationale for RNE is to avoid a large tree exploration and to keep modifications as near as possible to the congested link. In fact, while rerouting at *cFrom* removes a whole bundle of lightpaths from the link, doing the same at some upstream node in the tree may cause a smaller reduction of the load. On the other hand, simulations in Sect. 4 show that, unless very few iterations are allowed before halting, performance of RNE is significantly worse than RSNE.

3 Incremental Implementation on Dynamically Evolving Traffic

Local search heuristics can be seen as stepwise refinements of an initial solution by slight modifications of the system configuration. In our case, the RSNE algorithm starts from a shortest path routing scheme and changes at every step a routing table entry of a single node in the matrix. By performing many such changes, the system reaches a minimal congestion configuration.

This iterative scheme suits in a very appropriate way to a dynamic environment where traffic requirements evolve with time. In particular, if changes in the traffic matrix are reasonably smooth[1] even a small number of steps of the RSNE algorithm in Fig. 2 is sufficient to keep the system in a suitable state as the traffic matrix changes. Of course, only lines 2-23 must be executed, because we don't want to restart from scratch by calculating the shortest path routing tables. Moreover, a very low number of iterations of the outer loop (lines 3-23) must be performed at each step, i.e. MAXITER must be small (1 to 5 should suffice) to avoid excessive traffic disruption. In the following, we shall refer to the incremental algorithm as *Incremental* RSNE with k iterations per step: I-RSNE(k).

The simulations discussed in Sect. 4 show that even a single iteration of the algorithm yields good results under a fairly generic traffic model. The number of iterations of the algorithm is equal to the number of routing table entry modifications in the systems; thus, a very limited number of routing table entries must be modified as traffic evolves in order to keep congestion at low levels.

A similar approach has been proposed in [11], where branch-exchange methods are proposed for a local search heuristic; however, the type of local modification is quite different from our proposal.

[1]The assumption is reasonable even though IP traffic is known to be bursty: in fact, traffic requirements are given as an average over a certain amount of time, with some marginal capacity left to accommodate traffic peaks.

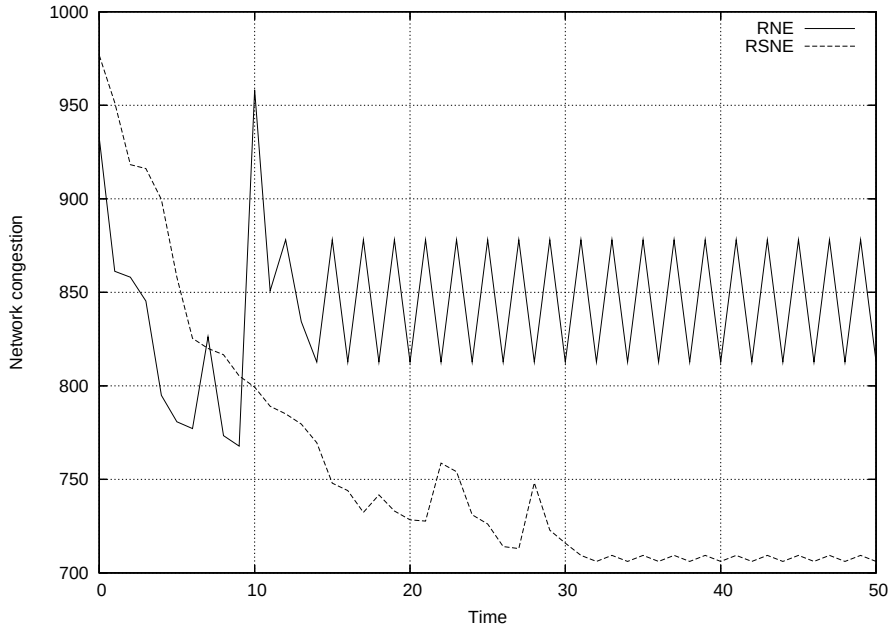

Figure 3: Comparison between RSNE and RNE algorithms.

4 Simulation Results

4.1 Static Traffic

To test the proposed algorithms we performed two sets of tests, static and dynamic. The first, using a static traffic matrix, explores the convergence speed of the RSNE and RNE algorithms.

Fig. 3 plots the evolution of the congestion value for a 50-iteration run of the RNE and RSNE algorithms with the same initial conditions; here the 14-node NSFNET backbone topology is used [10], while the traffic is randomly generated: every nondiagonal entry of the traffic matrix is a uniform value between 10 and 100. It turns out that the more complete RSNE algorithm outperforms its simplest version, although it sometimes achieves better results in the initial phase, probably because the algorithm is forced to move larger portions of load from edge to edge, achieving temporary better results but ending up with a complex, non-improvable routing scheme. Note that the congestion does not increase in a monotonic way: the algorithms do not halt when no improvement is possible, and the move leading to the smallest increase is chosen. This allows the system to escape local minima positioned in some shallow attraction basin. In many cases, this causes oscillation to take place once the minimum is achieved.

In the simulation shown here the maximum hop length corresponding to the lowest congestion configuration is 4. The corresponding value for the shortest path routing scheme is 3. The average hop length increases from 2.14 (shortest path) to 2.2 (RNE) and 2.21 (RSNE).

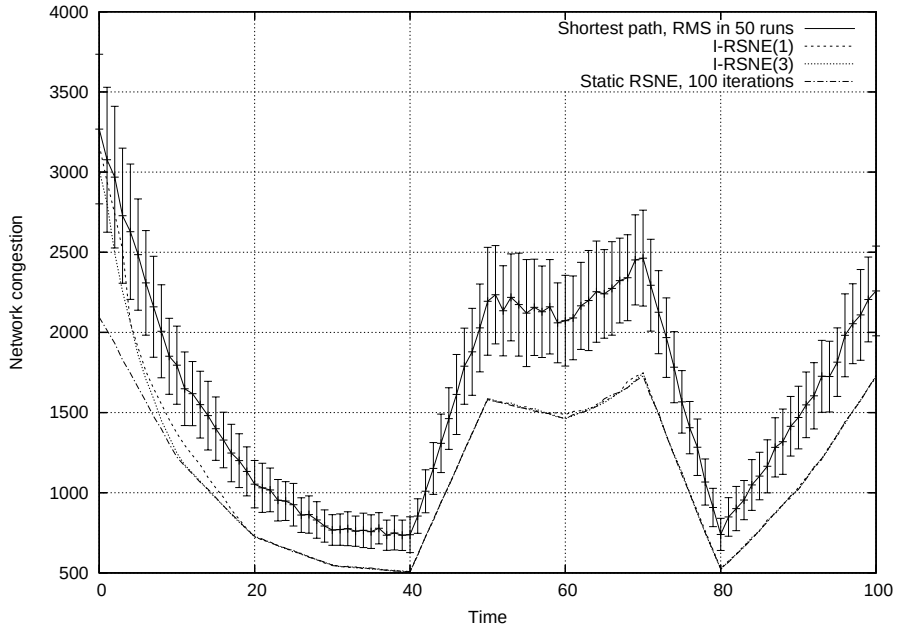

Figure 4: Comparison in terms of congestion: shortest path, `RSNE` and `I-RSNE`(*k*).

4.2 Dynamic Traffic

To investigate the behavior of the incremental version `I-RSNE`(*k*) with a dynamically evolving traffic pattern, we considered another topology, the 24-node regional network presented in [15].

To generate dynamic traffic we followed a model similar to that described in [9]. Given two positive integers N and Δ, we consider a sequence of $N\Delta + 1$ traffic matrices $(T^0, T^1, \ldots, T^{N\Delta})$ where matrix $T^{k\Delta}$, $k = 0, 1, \ldots, N$ is random and independently generated. For each of these matrices a random maximum value between 10 and 100 is generated, and each entry of the matrix is calculated as a random number between 10 and this maximum. The random maximum value has been introduced to take into account the variability of internet traffic in the mid term. All other matrices are linear interpolations of the immediately adjacent random matrices. In other words, given $h = 0, \ldots, \Delta - 1$ and $k = 0, \ldots, N - 1$, entry $T_{ij}^{k\Delta+h}$ of matrix $T^{k\Delta+h}$ is computed as follows:

$$T_{ij}^{k\Delta+h} = \text{round}\left[\left(1 - \frac{h}{\Delta}\right) T_{ij}^{k\Delta} + \frac{h}{\Delta} T_{ij}^{(k+1)\Delta}\right].$$

Fig. 4 describes the behavior of the proposed algorithms in the dynamic traffic case by comparing their congestion values. The upper plot represents the results achieved by the shortest path routing; for every traffic matrix, 50 different shortest path configurations were computed (with a random tie-breaking scheme), and the graph represents the $\mu \pm \sigma$ interval, where μ is the average and σ is the corresponding root mean square value. In fact, a large variability in the congestion (up to 35%) has been observed depending on random choices.

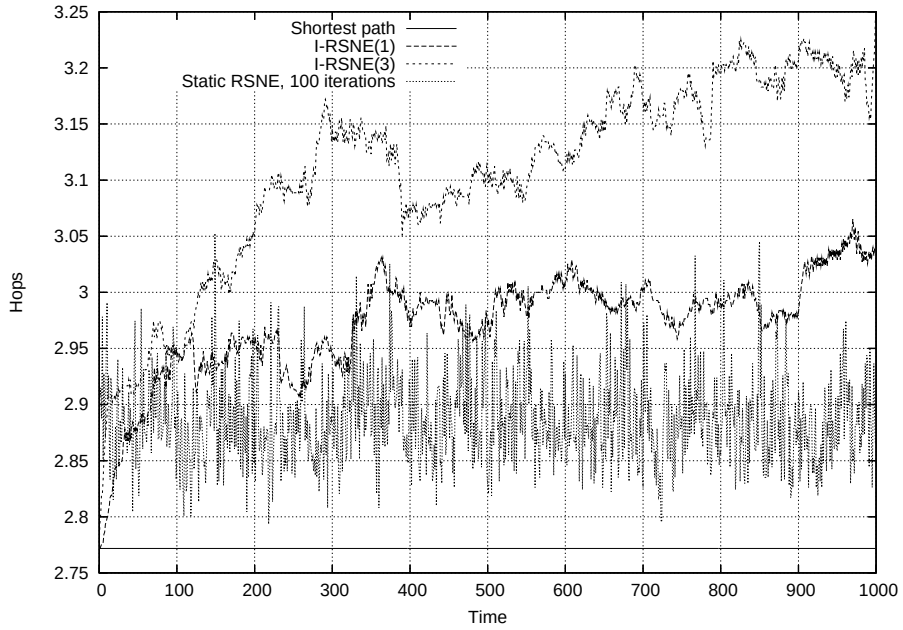

Figure 5: Comparison in terms of average hop length: shortest path, RSNE and I-RSNE(k).

Note that all RSNE and I-RSNE results are almost equivalent, well under the shortest path values. The only difference can be seen in the initial transient, when the incremental versions begin to differ from the pure shortest path configuration. This is a very important feature of the algorithm, because I-RSNE(1) requires the modification of a single entry of the routing table of a single node for each change in the traffic conditions. The RSNE and I-RSNE algorithms achieve results that are 8% to 12% better than the shortest path minimum over all the 50 runs, and up to 32% better than the average shortest path result.

If Fig. 4 is assumed to represent the traffic evolution during a day of real time, then a single change every fifteen minutes (in order to obtain about 100 changes per day) is sufficient to keep congestion at a local minimum, well below the shortest path routing.

Fig. 5 shows a comparison among the same algorithms in terms of average hop length, calculated as the mean value of hop distances (in the given routing scheme) between every node pair in the graph. The average hop length of shortest path routing, represented by the continuous bottom line, is obviously constant, and by definition it is the minimum (its value is 2.77).

The other plots, in particular the one representing the behavior of the offline RSNE algorithm, are particularly irregular when compared to those in Fig. 4; this is partly due to the narrower timescale, but it also depends on the fact that routing table changes are aimed at load reduction, and therefore hop lengths may vary from step to step. Note also that the I-RSNE outcomes are smoother, because adjacent results are strongly correlated, while the RSNE procedure performs a complete restart at every time step.

Fig. 5 highlights the main drawback of the incremental schemes I-RSNE(k): the shortest path configuration is never reimplemented, as was the case with RSNE, so the

average hop length is slightly growing in time.

While RSNE is constantly above the shortest path value by about 2%, the I-RSNE schemes tend to accumulate longer paths, getting to a 7% increase after 1000 time steps. Note that the difference grows in time. To overcome the problem a simple modification consists of restarting from a shortest path configuration every time the average (or the maximum) hop length trespasses a given threshold.

5 Conclusions

The paper proposed and motivated a heuristic technique for load balancing in IP-based optical networks (RSNE) built on simple modifications of routing tables. Some variations were introduced to reach lower algorithmic complexity (RNE) and to obtain a faster, incremental evaluation in the case of dynamically evolving traffic (I-RSNE).

Comparisons between the new techniques and the shortest path routing scheme, both in terms of network congestion and length of the resulting routes, show that the proposed algorithms are effective to reduce congestion, and outperform shortest path routing by up to 32%. The resulting increase in hop length is limited to a small amount (up to 7% in the worst case considered in the paper).

The RSNE algorithm explores all possible improvements before taking a step. Further investigation will determine how the quality of the solutions deteriorates if a randomized approach is followed in order to distribute the algorithm.

Acknowledgments

We would like to thank Imrich Chlamtac and Jason Jue of the University of Texas at Dallas, for their interesting and fruitful discussions with the authors about the subject of this work, and the anonymous referees for their precious comments.

References

[1] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: A novel approach to high bandwidth optical WANs. *IEEE Transactions on Communications*, 40(7):1171–1182, 1992.

[2] L. Fratta, M. Gerla, and L. Kleinrock. The flow deviation method: An approach to store-and-forward communication network design. *Networks*, 3:97–133, 1973.

[3] M. Kodialam and T. V. Lakshman. Integrated dynamic IP and wavelength routing in IP over WDM networks. In *Proceedings of IEEE INFOCOM 2001*, pages 358–366, 2001.

[4] J. Labourdette and A. Acampora. Logically rearrangeable multihop lightwave networks. *IEEE Transactions on Communications*, 39:1223–1230, August 1991.

[5] Emilio Leonardi, Marco Mellia, and Marco Ajmone Marsan. Algortihms for the topology design in WDM all-optical networks. *Optical Networks Magazine*, 1(1):35–46, January 2000.

[6] K. Lu, G. Xiao, and I. Chlamtac. Blocking analysis of dynamic lightpath establishment in wavelength-routed networks. In *Proceedings of ICC2002*, 2002. submitted.

[7] D. Mitra, R. Gibbens, and B. Huang. State-dependent routing on symmetric loss networks with trunk reservations. *IEEE Transactions on Communications*, 41(2):400–411, 1993.

[8] Ahmed Mokhtar and Murat Azizoğlu. Adaptive wavelength routing in all-optical networks. *IEEE/ACM Transactions on Networking*, 6(2):197–206, April 1998.

[9] Aradhana Narula-Tam and Eytan Modiano. Dynamic load balancing in WDM packet networks with and without wavelength constraints. *IEEE Journal of Selected Areas in Communications*, 18(10):1972–1979, oct 2000.

[10] R. Ramaswami and K. N. Sivarajan. Design of logical topologies for wavelength-routed optical networks. In *Proceedings of IEEE INFOCOM 1995*, 1995.

[11] Jadranka Skorin-Kapov and Jean-François Labourdette. On minimum congestion routing in rearrangeable multihop lightwave networks. *Journal of Heuristics*, 1:129–145, 1995.

[12] C. Xin, Y. Ye, T.S. Wang, and S. Dixit. On an IP-centric control plane. *IEEE Communications Magazine*, 39(9):88–93, 2001.

[13] Bülent Yener and Terrance E. Boult. A study of upper and lower bounds for minimum congestion routing in lightwave networks. In *Proceedings of INFOCOM 1994*, pages 138–149, 1994.

[14] H. Zang, J.P. Jue, L. Sahasrabuddhe, R. Ramamurthy, and B. Mukherjee. Dynamic lightpath establishment in wavelength routed networks. *IEEE Communications Magazine*, 39(9):100–108, 2001.

[15] Zhensheng Zhang and Anthony S. Acampora. A heuristic wavelength assignment algorithm for multihop WDM networks with wavelength routing and wavelength re-use. *IEEE/ACM Transactions on Networking*, 3(3):281–288, June 1995.