UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

DYNAMIC LOAD BALANCING IN WDM NETWORKS

Mauro Brunato, Roberto Battiti, Elio Salvadori

June 2002

Technical Report # DIT-02-0011

Also: to be published on Optical Network Magazine special issue on dynamic optical routing

Dynamic Load Balancing in WDM Networks*

Mauro Brunato Roberto Battiti Elio Salvadori

Università di Trento,
Dipartimento di Informatica e Telecomunicazioni,
via Sommarive 14, I-38050 Pantè di Povo (TN), Italy
Email: brunato|battiti|salvador@dit.unitn.it

June 3, 2002

Abstract

In this paper new load balancing techniques are proposed based on IP-like routing, where the forwarding mechanism is only driven by the destination address. The aim of this work is to consider the applicability of this approach in the context of optical networks.

The proposed algorithm (RSNE — Reverse Subtree Neighborhood Exploration) implements a local search technique where the basic step is the modification of a single entry in the routing table of a node. A randomized method with reduced computational complexity (fRSNE — fast RSNE) is also presented and analyzed. Both schemes allow an incremental implementation where local search steps are continuously performed as traffic conditions change.

Experiments under static and dynamic (time varying) traffic scenarios show a rapid reduction of the congestion. The performance of the incremental scheme while tracking a changing traffic matrix is comparable to that obtainable through the complete re-optimization of the traffic, while the randomized implementation is particularly efficient when scaling properties are considered.

*To be published on *Optical Network Magazine* special issue on dynamic optical routing

1 Introduction

In present days we experience an increasing bandwidth demand caused by the exponential Internet growth and the introduction of communication-intensive applications. In this framework, techniques such as optical Wavelength Division Multiplexing (WDM) and routing schemes such as Generalized Multi Protocol Label Switching (G-MPLS) have been proposed. In WDM networks, each fiber carries multiple independent data streams on different carriers in order to achieve complete spectrum utilization. Relevant efforts are being spent to assign a wavelength to each data stream in such a way that all traffic is handled in the optical domain, without any electrical processing on transmission [4].

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.

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 being developed to enable fast switching of various type of connections, including IP based 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 [21]. In this scenario it is necessary to study the impact of routing mechanisms typical of the IP world, by analyzing the possible integration of label switching techniques (MPLS) with the current optical switching architectures. In addition, it is important to study criteria and algorithms to decide when and how lightpath allocation and release requests are generated in the presence of data traffic.

The optimization in the usage of scarce resources in optical networks with WDM technology leads to the problems of packet routing (in packet networks) and of creating virtual connections by considering both routing and wavelength assignment. Some seminal papers are [6, 19, 17, 1, 24]. A review of algorithms for designing virtual topologies directly on the optical layer is presented in [10]. The network evolution in terms of traffic amount and flexibility requirements indicates that mesh networks must be considered instead of typical SDH/SONET-like ring topologies, thus opening a more complex scenario. A novel heuristic approach for the design of WDM networks under static traffic demands is described in [13]. The work [2] considers the physical topology as completely assigned and exploits the resources of the optical layer for the design of the virtual topology. Heuristics for the design of virtual topologies, based on greedy principles [18] or on linear programming [8] have been recently discussed.

Routing that takes into account the combined topology and resource usage information at the IP and optical layers [7], with constraints on the maximum delay or number of hops, is an area that deserves additional exploration. Paper [23] investigates distributed control mechanisms for establishing all-optical connections in a wavelength routed WDM network: an approach based on link-state routing, and one based on distance-vector routing. A novel algorithm for inte-

grated dynamic routing of bandwidth guaranteed paths in MPλS networks is developed in [7]. Another fundamental issue when designing network algorithms is their ability to function with the limitations of a distributed environment, i.e. local and delayed information. Work [11] considers the case in which nodes explore a limited portion of the search space by considering only a certain number of links along a path.

The technological context of our proposal is summarized in Section 2, then the Reverse Subtree Neighborhood Exploration (`RSNE`) algorithm is introduced in Section 3. In Section 4 a randomized implementation and an incremental implementation are motivated and described. In Section 5 an ILP formulation of the problem is given in order to compare the proposed algorithms with the actual optimum values, at least for small networks. Finally, in Section 6 simulation results are analyzed by considering both the static and the dynamic traffic cases.

2 Adaptive routing and load balancing

The purpose of Load Balancing in WDM based networks is to reduce the congestion in the network. The 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-MPLS 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 in WDM networks consists of two subproblems: the lightpath connectivity and the traffic routing problem. The routing problem has its origin at the beginning of the networking research, see [16] for a review of previous approaches to the problem. In particular, adaptive routing, that incorporates network state information into the routing decision is considered in [15] in the context of all-optical networks, while previous work on state-dependent routing with trunk reservation in traditional telecommunications networks is considered in [14]. It is also known that flow deviation methods [5], 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, algorithms that are based on a sequence of small steps (i.e., on local search from a given configuration) are considered. In [9] "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 [22] that also proposes variable depth local search and simulated annealing strategies. Strategies based on small changes at regular intervals are proposed in [16].

Our technological context is that of dynamic lightpath establishment in wavelength-routed networks reviewed in [23]. 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 [7] for integrated IP and wavelength routing and [12] for a blocking analysis in the context of *destination initiated reservation*.

This paper describes an 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 a single routing table. 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.

In this article we build on our previous work [3]. New contributions include the introduction and experimentation of a randomized version of the algorithm, a comparison with an ILP formulation and a largely extended experimental section.

Let us define the context and the notation. By *physical topology* we mean the actual network composed of passive or configurable optical nodes and their fiber connections. The *logical topology* is given by the lightpaths between the electronic routers, determined by the configuration of the OADMs and transmitters and receivers on each node. 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 $t_{ij} = 0$ if $i = j$. A *routing table* is an array, associated to each node of the network, containing the next-hop information required for routing. In the following we shall consider IP-like routing, where the next hop is maintained for each possible destination, regardless of the index of the source node. For a given traffic pattern and routing tables associated to the nodes, the sum of the number of lightpaths passing through each link is called the *load* of the link. Finally, the maximum load on each link of a path is called the *congestion* of this path. The maximum load on each link of a network is called the *congestion* of the network.

The Load Balancing problem can be defined as follows.

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

3 Local Search for the Load Balancing Problem

The basic idea of the new Load Balancing scheme is as follows: start from the shortest path routing and then try to minimize the congestion of the network by performing a sequence of local modifications. For each tentative move, the most congested link is located, and part of its load is re-routed along an alternate path.

We shall begin with some definitions and explanations of the functions and variables. We maintain the set *candidatePathSet* containing paths that are candidate to replace those passing through the most congested links of the network. This set is emptied at each iteration of the algorithm.

Given all routing tables, every node d identifies a spanning tree of the network, namely the tree composed of all links that carry lightpaths addressed to d (it is a tree because of the destination based routing). We are interested in

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

Figure 1: the Local Search RSNE algorithm.

identifying a subtree of this tree, and we use the function *routingTree*(*d*,*r*) returning the subtree rooted in node *r* of the routing tree having destination node *d*. The shaded tree shown in Figure 2 is actually *routingTree*(*dest*,*cFrom*), and it contains all nodes whose lightpaths directed to destination *dest* pass through node *cFrom*. The function *shortestPathRouting*(*network*) calculates the shortest path tree for each destination node and returns the corresponding set of routing tables as a matrix. *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, 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.

Figure 1 shows a description of our Local Search algorithm used for the Load Balancing problem. In the rest of the paper we shall refer to it as *Reverse Subtree Neighborhood Exploration* (RSNE).

The initialization section (lines 1–2) starts by generating the routing tables by the application of the Shortest Path Routing algorithm to the specific network (the costs of all the edges are considered as uniform). 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 research of the minimum) and the set of congested links *congestedLinkSet*. Then *candidatePathSet* is empty at the beginning of each iteration.

Every iteration of the local search algorithm (lines 3–23) consists of two distinct parts. First, a set of alternate paths for part of the traffic passing

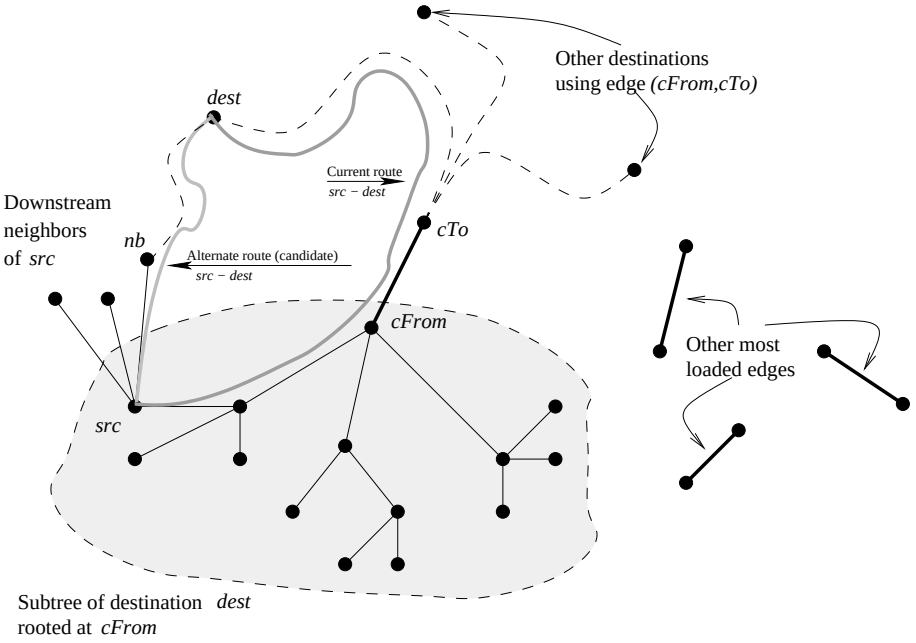

Figure 2: search space for a move of the RSNE algorithm.

through the most congested links is found (lines 6–17): then, once the most promising candidate move is selected, the routing tables of the network (lines 18–22) are corrected accordingly, then the iteration starts by considering the new congested links.

The first part includes the core of RSNE algorithm. Refer to Figure 2 for a visual reference. We consider each congested link in *congestedLinkSet* (loop at lines 6-17). Let us identify this link with its endpoints (*cFrom*,*cTo*). Then the procedure iterates through all lightpaths that use that link. Two nested loops are present: 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 every (*src*,*dest*) pair whose lightpath goes through the link (*cFrom*,*cTo*), we try to reroute such lightpath by altering the routing table in *src*. To do this, we temporarily remove the load of the lightpath from the current route (function *removePartialLoad* at line 9) and iterate through all downstream neighbors *nb* of *src* calculating the maximum load that would be caused by re-routing 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 the candidate set *candidatePathSet*. In particular, the current minimum is stored in *bestCandidateLoad*. If the load obtained after this traffic re-route 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 (lines 14-16). At the end of the alternate paths research, 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 source nodes *src* (line 8).

In the second part of the `RSNE` algorithm, if the resulting set *candidatePathSet* is not empty then one random solution 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 relative 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 *candidatePathSet* is empty (line 22), or until a given number of iterations has been performed (line 23).

4 Algorithm properties and modifications

As an estimate of the CPU time required by the algorithm, let us consider its computational complexity of an iteration (lines 4–22) in terms of *node visits*, i.e. operations on nodes of the network. In detail, a counter is incremented for each node considered in the candidate path at line 11. The proposed algorithm is composed of nested cycles. Let n be the number of nodes in the network; if d is the largest node degree, the number of edges is $O(nd)$. The number of iterations for the loop at line 6 is only bounded by the number of links in the network. Each of the two nested loops at lines 7 and 8 may scan almost all nodes in the worst case. Function `removePartialLoad` may need to operate on a long path (the diameter of the network is potentially equal to n). Loop at line 10 is executed d times at most, and the path check at line 11 is again bounded by the number of nodes. Then the complexity of one `RSNE` iteration is $O(n^4 d^2)$. So in the worst case, when d grows linearly[1] with n, the overall complexity of an iteration is $O(n^6)$.

However, this is a pessimistic, worst-case bound: experiments on networks modeled by Euler disk graphs (see Section 6.2) suggest that the number of node visits (while the algorithm is exploring the routing subtree and checking the new path) has $O(n^\alpha)$ empirical complexity with α slightly higher than 2. However, simplifications of the algorithm shall be introduced in the next subsections in order to lower the worst-case complexity of the algorithm to a reasonable power of n and d.

4.1 Randomized version (`fRSNE`)

Computational complexity and scalability are two major issues in current network algorithm research: algorithms are required to adapt to different network sizes without excessive slowdown. In the present case, a drastic reduction of the computational complexity can be obtained by reducing some of the loops to a fixed, small number of random choices. A fixed-size subset of the congested links can be explored by the loop at line 6, and a fixed number of random destinations can be covered by the loop at line 7. The loop at line 8 is basically a subtree exploration. By limiting the degree of the descent through this tree, the number of source nodes that are explored can be reduced. When all these reductions

[1] this happens in many real-world contexts: in Internet autonomous systems there is evidence [20] of a strong correlation between the logarithms of n and d with a coefficient near to 0.95, so that $d = O(n^{.95})$.

are performed together, the complexity of an iteration becomes $O(n^2 d)$ in the worst case ($O(n^3)$ for unbounded degree). Experiments on the same Euler disk graphs have reported an $O(n^\beta)$ empirical complexity (in terms of node visits), with $\beta \approx 1.67$.

In the following, the randomized version will be called `fRSNE`(e,d,s) (*fast* `RSNE` on e edges, d destinations per edge and degree s of source subtree exploration). As shown in Section 6, when compared to `RSNE`, performance degradation of `fRSNE`(1,1,1) (the randomized version where one edge, one destination and just one source subtree path are considered) is almost negligible in terms of congestion, while average hop length and average edge load are only slightly increased.

4.2 Incremental version (`I-RSNE`)

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 stabilizes to a low congestion configuration.

This iterative scheme is appropriate for a dynamic environment where traffic requirements evolve with time. In particular, if changes in the traffic matrix are reasonably smooth[2] even a small number of steps of the `RSNE` algorithm in Figure 1 are 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 very small to avoid excessive traffic disruption. In the following, we refer to the incremental algorithm as *Incremental* `RSNE` with k outer iterations per step: `I-RSNE`(k).

Simulations discussed in Section 6 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 equivalent 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. Moreover, `I-RSNE` is fully compatible with the `fRSNE` randomized scheme discussed above, and experiments show that the performance of the combination which shall be called `I-fRSNE` does not degrade.

A similar approach has been proposed in [19], where branch-exchange methods are proposed for a local search heuristic. However, the type of local modification is different from our proposal.

4.3 Restricted Neighborhood Exploration (`RNE`)

A simplified version of the algorithm was also tested, which can be called `RNE` (Restricted Neighborhood Exploration); it only considers node *cFrom* as a candidate for re-routing, with no exploration of its *routingTree* (consider the algorithm in Figure 1 where the loop at lines 8–17 is executed only once with *src*

[2]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.

equal to *cFrom*). This would be equivalent to remove as much load as possible from the congested link with a single routing table change. However, as simulations in Section 6 show, such policy does not compare well with `RSNE` and `fRSNE`, probably because too large amounts of load are moved at each step, while finer modifications are more appropriate.

5 ILP formulation

In order to compare the results of the proposed techniques with the actual optimum, the following Integer Linear Programming (ILP) formulation can be used.

Let us consider the n-node oriented graph $G = (V, E)$ where $V = \{1, \ldots, n\}$ and $E \subseteq V \times V$. For each couple of nodes $s, d \in V$, $s \neq d$, let us define the requested traffic as T_{sd}.

For each $i, j, s, d \in V$ such that $(i, j) \in E$ and $s \neq d$, the binary flow variable $F_{sd}^{ij} \in \{0, 1\}$ is introduced; its value is 1 if and only if data from source s to destination d is routed through edge (i, j). The usual set of flow conservation (solenoidal) constraints can be imposed (each family of constraints is identified by an indexed label on the right):

$$\sum_{\substack{i \\ (i,j)\in E}} F_{sd}^{ij} - \sum_{\substack{i \\ (j,i)\in E}} F_{sd}^{ji} = \begin{cases} -1 & s = j \\ 1 & d = j \\ 0 & \text{otherwise,} \end{cases} \qquad (\text{FLOW}_{sd}^{j})$$

one equation for each combination of $j, s, d \in V$ with $s \neq d$. These constraints avoid imbalances between the incoming and outgoing flow at all nodes with the exception of the source s and the destination d.

IP routing is destination-driven, so a new family of binary variables is introduced in order to consider only the flow destination. For each $d, i, j \in V$ such that $(i, j) \in E$, let $R_d^{ij} \in \{0, 1\}$ equal to 1 if and only if edge (i, j) carries traffic towards destination d. This information can be extracted from the F_{sd}^{ij} variables by imposing $R_d^{ij} = 1$ as soon as there is at least one source s such that $F_{sd}^{ij} = 1$. This can be obtained by the following linear constraints:

$$R_d^{ij} \geq F_{sd}^{ij}, \qquad (\text{ROUTE}_{sd}^{ij})$$

one equation for every combination of indices $s, d, i, j \in V$ where $s \neq d$ and $(i, j) \in E$.

IP routing is achieved when for every node and every destination there is at most one outgoing edge carrying flow for that destination, regardless from the source. The following constraints impose that limit:

$$\sum_{\substack{i \\ (i,j)\in E}} R_d^{ij} \leq 1, \qquad (\text{IP}_d^i)$$

one equation for every $i, d \in V$.

After imposing IP routing on the graph, congestion must be minimized by introducing a new variable $F_{\max}$, constrained in order to be larger than every

link load:

$$F_{\max} \geq \sum_{\substack{s,d \\ s \neq d}} T_{sd} F_{sd}^{ij}, \qquad (\text{LOAD}^{ij})$$

for every $(i,j) \in E$.

The problem can be stated as follows:

$$\begin{array}{ll} \text{Minimize} & F_{\max} \\ \text{Subject to} & \begin{cases} \text{FLOW}_{sd}^{j} & \forall s,d,j \in V, \quad s \neq d \\ \text{ROUTE}_{sd}^{ij} & \forall s,d,i,j \in V, \quad s \neq d, \quad (i,j) \in E \\ \text{IP}_{d}^{i} & \forall d,i \in V \\ \text{LOAD}^{ij} & \forall i,j \in V, \quad (i,j) \in E. \end{cases} \end{array} \qquad (\text{ILP})$$

This formulation will be used in the following section in order to test the `RSNE` heuristic against the optimal value on small networks, and against lower bounds determined by the CPLEX optimizer if the optimum search could not be completed because of time limits.

6 Simulation results

Several experiments have been performed in order to test the `RSNE` heuristic on different network sizes and topologies, both computer generated and real, with static and dynamic traffic conditions.

6.1 Experimental setting

Shortest path and the proposed heuristics were simulated by a C++ program, with some classes devoted to generate network and traffic instances according to various models. The following network models were considered:

- *random graphs*, parameterized by the number of nodes and edge density, i.e. the probability of an edge to exist for every couple of nodes;
- *Euler disk graphs*, parameterized by the number of nodes and by a radius r, where nodes are scattered in a unit square, and two nodes are connected if and only if their distance is less than r;
- *real-world networks*, with connection matrices read from a file.

Traffic models are of the following types:

- *static uniform matrices*, where all non-diagonal entries have the same value;
- *static random matrices*, where all non-diagonal entries are taken by a uniform random distribution between a given minimum and maximum;
- *dynamic random matrices*, generated as follows (see [16] for a similar model): given two positive integers N and Δ, consider a sequence of $N\Delta + 1$ traffic matrices $(T^0, T^1, \ldots, T^{N\Delta})$ where matrices $T^{k\Delta}$, $k = 0, 1, \ldots, N$ are random and independently generated, by choosing a random maximum value between 10 and 100 and calculating every entry as a

random number between 10 and this maximum; 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:

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

- *real-world traffic matrices*, read from a file.

To generate pseudo-random sequences, a Mersenne Twister algorithm of period $2^{19937} - 1$ was used[3]. Every random-sensitive object (the graph generator, the traffic generator and the heuristic routing algorithms) was endowed with a different instance of the generator, in order to ensure independence and reproducibility of initial conditions for different algorithms.

The C++ program was also used to output problem instances to files in MPS format in order to solve them via a linear problem optimizer. CPLEX 7.1 was used to solve these instances.

6.2 Empirical complexity tests

The `RSNE` and `fRSNE` algorithms have been tested on random Euler disk graphs in order to obtain a measure of the growth of computational time (in terms of node visits per iteration) as the network size increases. Euler disk graphs were selected because they show some properties similar to real-world networks, such as local connection schemes and a larger number of multi-hop paths when compared with completely random graphs. Moreover, by letting the number of nodes n increase while keeping the radius constant, the degree is proportional to n, thus unbound.

Figure 3 plots the number of node visits against the size (in nodes) of the network: 10 samples for each network size have been generated, and both `RSNE` and `fRSNE` have been tested. Least-squares linear regression has been calculated on the logarithmic transforms of the data to estimate the highest exponent in the dependence formula. If v is the number of node visits, the resulting dependencies are $v \approx 0.14 \cdot n^{2.04}$ for the `RSNE` algorithm and $v \approx 0.02 \cdot n^{1.67}$ for `fRSNE`. Thus, even though experimental data show a large variability, the `fRSNE` algorithm has a lower asymptotic complexity in the case considered as well as a much lower constant multiplier.

6.3 Tests on static traffic

The first tests of the `RSNE` algorithm aim at assessing its capacity to outperform the simple shortest path scheme in a static traffic context.

The 24-node regional network presented in [24] and the traffic pattern presented in the same work have been chosen for the first static simulation. We executed 1000 steps of the `RSNE` and `fRSNE` schemes and compared the results with the initial shortest path configuration.

[3]The code, written by Makoto Matsumoto (Keio University, Japan) and Takuji Nishimura (Yamagata University, Japan), is available at:
`http://www.math.keio.ac.jp/matumoto/emt.html`.

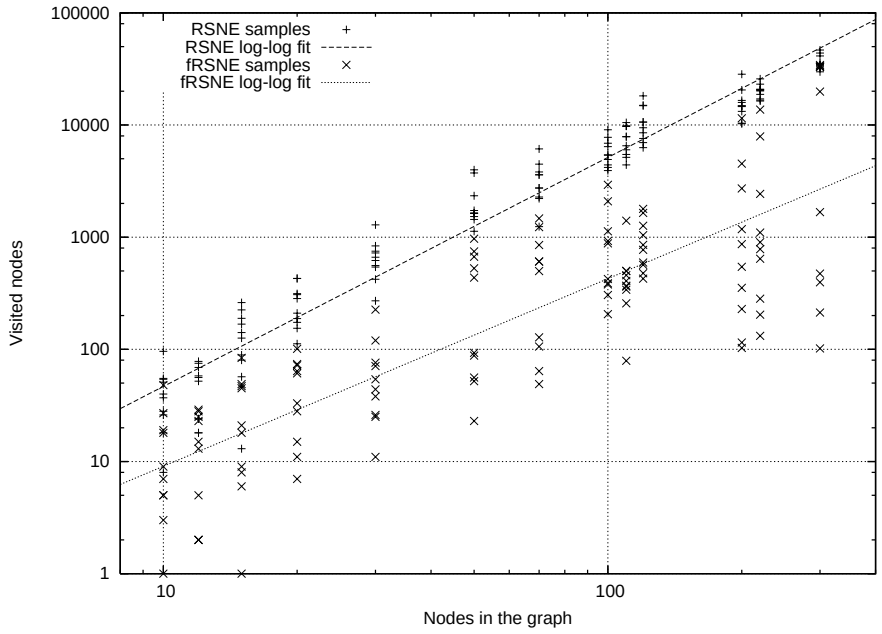

Figure 3: number of node visits in one iteration of `RSNE` and `fRSNE` for Euler disk graphs with radius .3 on a unit square.

Figure 4 shows how edge loads distribute under the three policies for a random traffic matrix. Load values have been grouped in classes of 20, and the histogram represents the population of each class. The maximum load for the shortest path routing was 242, `RSNE` and `fRSNE` both reduced it to 157 (`RSNE` found it at the 26th iteration), obtaining a 36% reduction. The overall shape of the load distribution is not much different in the three cases, apart from the longer tail in the shortest path distribution. In particular, the number of light-loaded edges remains similar[4], and the average edge load is increased by 6.8%, from 30.37 to 32.42.

Another important verification concerned the distribution of hop lengths. In this case, the maximum hop length of 7 hops was left unchanged by both `RSNE` and `fRSNE` algorithms, while a 1.8% increase in the average hop length (from 2.77 to 2.82 with `RSNE`) was verified. This is unavoidable, because the shortest path routing minimizes hop length by definition, so it necessarily outperforms all other schemes. However, the imbalance is very small.

Figure 5 plots the best congestion value against the number of steps for one run of the `RNE` and `RSNE` algorithms; here the NSFNET topology was used with a random traffic pattern from 10 to 100 for each couple of nodes. It turns out that the more complete `RSNE` algorithm outperforms its simplest `RNE` version, although the latter seems to have a better result in its earlier phase: this behavior shows up in many cases, probably because the algorithm is forced to move larger portions of load from edge to edge, achieving temporary better

[4]Indeed, a scheme which reduces congestion but substantially increases the load of many other edges would be unacceptable.

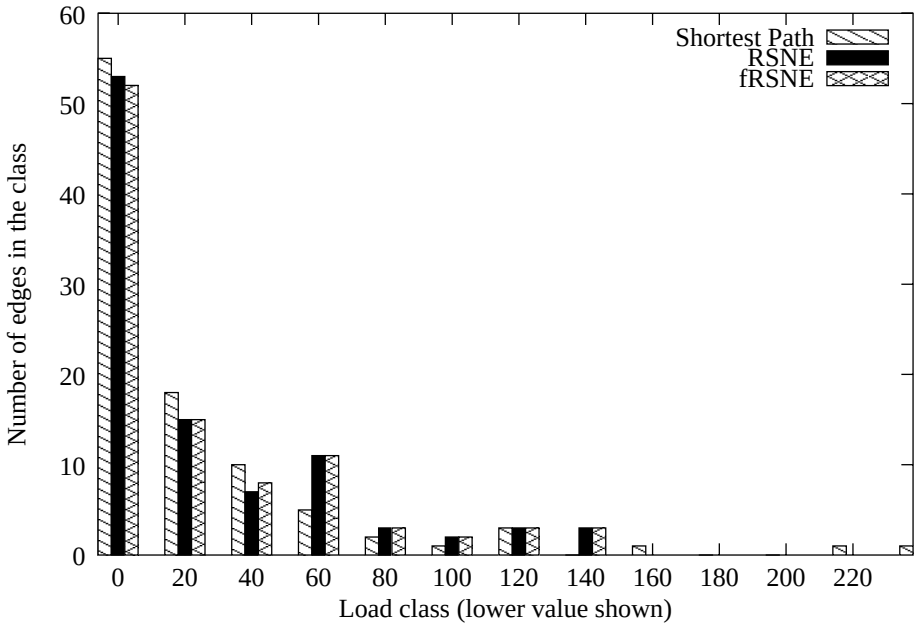

Figure 4: distribution of edge loads for Shortest Path routing, `RSNE` and `fRSNE`.

results but ending up with a complex, non-improvable routing scheme. For clarity, only the first 50 iterations are shown. However, the `fRSNE` scheme, showing an intermediate behavior due to the smallest move space at each step, eventually finds the same values shown by the complete heuristic. The above simulation gave a maximum hop length equal to 7 (i.e. a lightpath needs to travel 7 links from source to destination) at each iteration. This is the minimum, because it also results from the shortest path routing assignment that initiates all algorithms.

Table 1 shows the results of a comparison between the shortest path algorithm, the `RSNE` technique and the solutions provided by the CPLEX optimizer when the same instances were put in the ILP form. Every line reports the

Table 1: comparison between algorithms on small random networks with 60% density. Average figures on 10 experiments per size, intervals are shown where ILP could not find solution in scheduled time.

Nodes	Shortest Path	`RSNE`	ILP
5	333.83	312.41	312.24
6	379.05	348.91	340.12
7	345.86	263.98	[257.16, 254.51]
8	416.03	325.24	[345.77, 305.92]
10	374.87	254.88	[374.87, 228.45]
12	463.10	249.99	[495.73, 249.13]

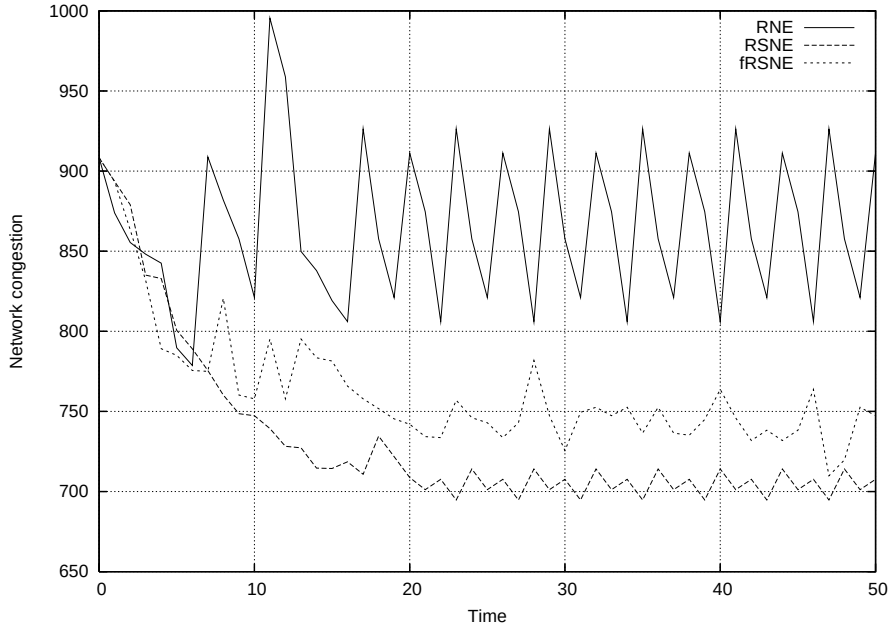

Figure 5: behavior of heuristics during a single run.

average of 10 experiments; a time limit of 10 minutes for each instance was imposed to CPLEX (running on a 1.5GHz PIII Linux machine with no other CPU-consuming tasks), and if the optimum was not found then the current feasible solution and the lower bound were reported as the interval where the optimum lies. Note that the `RSNE` algorithm always outperforms the shortest path routing, and when relatively large networks are considered the CPLEX integer solver needs much more than 10 minutes in order to find the optimum (we let a 12-node test run for three days without improving the estimate before the branch-and-bound tree caused a memory overflow).

To allow a better comparison among heuristics, a series of experiments were performed on random networks (Figures 6 and 7) and on Euclidean disk networks (Figure 8).

Figures 6, 7 and 8 have been obtained by averaging 50 runs of the algorithms for each plotted bar, representing the 95% confidence interval of the true mean congestion value. In Figure 6 node size was varied from 10 to 50 in steps of 10, with a constant 50% edge density, and disconnected networks were discarded. Figure 7 compares algorithms on random networks with a constant size of 20 nodes and different densities from 40% to 95%. Figure 8 has been obtained from the same experiments on Euler disk networks with radius equal to .3 (remember that points are randomly scattered throughout a unit square). All graphs plot mean congestion values. In all cases, the two lower lines, representing `RSNE` and `fRSNE`, are almost equal, and this motivates support of the randomized version. On the other hand, the simplified `RNE` scheme does not show a significant performance improvement over shortest path routing in the random network case. Its use, however, may be motivated in the Euler disk case, where an average

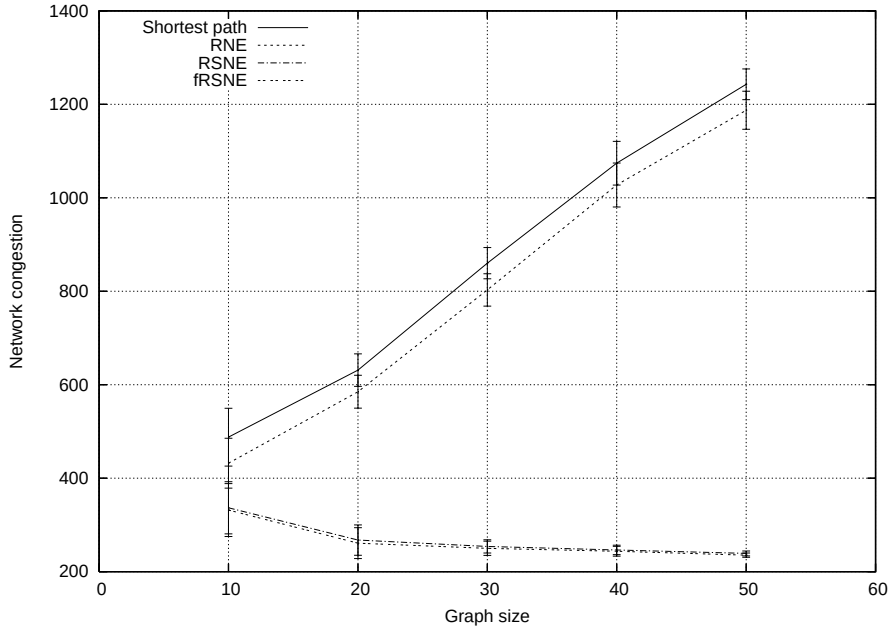

Figure 6: congestion on random networks of different node size, 50% edge density. Random bars represent the 95% confidence interval, the two lower plots, representing `RSNE` and `fRSNE`, are almost equal.

43% congestion reduction is obtained in the 50-nodes case.

Figure 6 plots an interesting behavior: while edge congestion obtained by `RSNE` (or equivalently `fRSNE`) is almost constant, the shortest path result is linearly increasing. The same can be seen in Table 1. This can be explained by considering that both the overall traffic requirement and the number of edges increase as the square of the number of nodes. Techniques aiming at congestion reduction are able to exploit this fact, while shortest path policies, which do not aim at congestion reduction as their primary target, tend to crowd paths along eventual shortcuts. While congestion can be drastically decreased (up to 5.5 times on 50 nodes), the average hop length increases up to 4% for `RSNE` and 5.7% for `fRSNE`. Likewise, increase in average edge load has been detected up to 3.6%.

In Figure 7, the congestion reduction operated by `RSNE` and `fRSNE` reaches its maximum (67%) at 70% edge density, then it becomes less and less significant, and the different plots tend to the same congestion value as density approaches 100%. This is obvious, as all policies will use one-hop routing on a clique. For the same reason, all average hop lengths tend to 1 as density approaches 100%. The highest increase (3.6% for `RSNE`, 6% for `fRSNE`) has been detected for the lowest considered density, 40%.

Figure 8 depicts a more realistic situation in which connection depends on distance. In this case confidence intervals are much larger because topology is more variable. Clusters of nodes with few congested inter-cluster links can appear with significant probability, as well as uniformly distributed networks

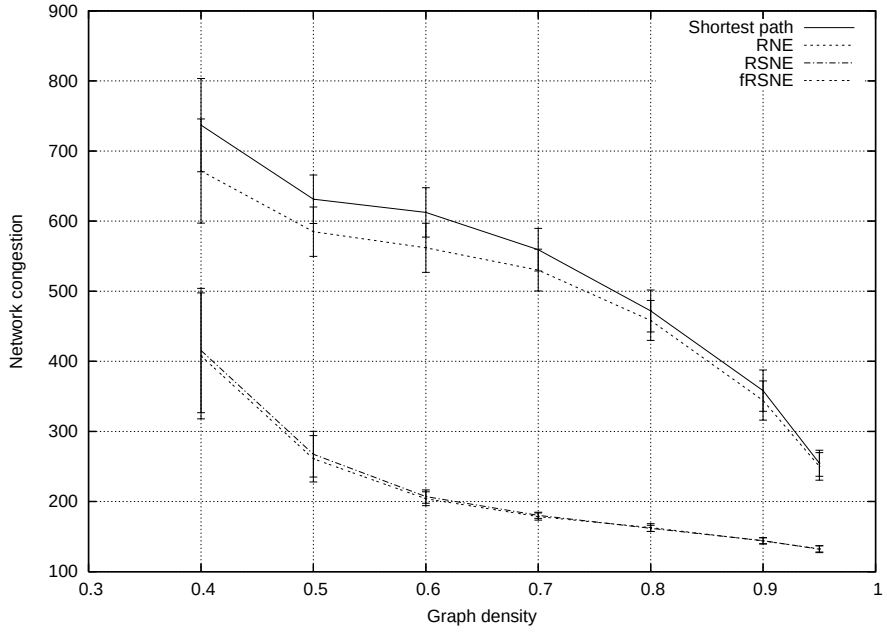

Figure 7: congestion on random networks of different densities, 20 nodes.

with balanced loads. Also in this case the highest congestion reduction operated by `RSNE` and `fRSNE` (56%) is detected for 50 nodes. The average hop length is increased in the worst case by 12% (from 2.6 to 2.92 for 50 nodes), while the average load is 11% higher.

In all cases considered in this section, a large improvement on congestion reduction has been obtained at the expense of a slight degradation of the other performance indices that had been considered, average hop length and average load.

6.4 Tests on dynamic traffic

The 24-nodes network already considered for static traffic has been tested with traffic evolving in time. The traffic evolution pattern is that discussed earlier. The complete time span is of 1000 time steps, with a new independent matrix every 20 steps and linear interpolation on intermediate matrices.

Figure 9 reports the results of the simulation for the first 100 steps. As the shortest path outcome is highly variable, we chose to calculate 50 shortest path routings with random tie-breaking per time step. All runs were executed over the same traffic pattern. Error bars represent the span from the lowest to the highest congestion value obtained. The other lines represent one run of `RSNE` with full restart and 100 iterations at each time step, one run of `I-RSNE`(1) with just one optimizing iteration per time step and one run of `I-fRSNE` with one optimizing iteration per time step and all parameters set to 1. As expected from static tests, results achieved with the proposed techniques are always below the shortest path range. The fact that incremental implementations occasionally outperform the

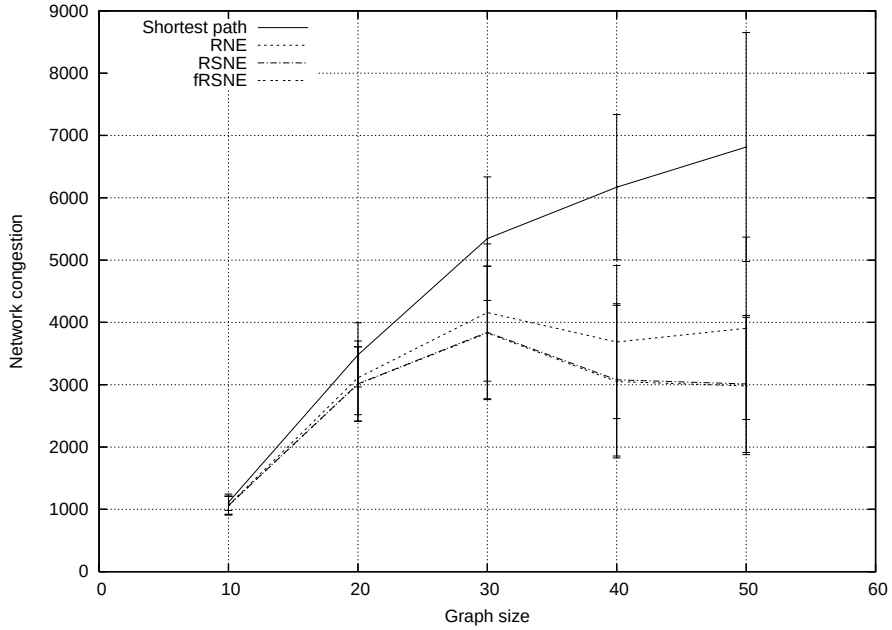

Figure 8: congestion on Euler disk networks of different node sizes, radius .3 on a unit square uniform scattering.

static `RSNE` depends from the low number of iterations per step. An initial transient can be seen for the first 10 steps, while the incremental algorithms descend from their initial shortest path configurations to lower congestions.

These results suggest that modifying a single entry in the routing table of a single node at each step is enough to adequately follow the traffic pattern, at least with this traffic model, even with a restricted random exploration of the move space.

While congestion is significantly reduced, a slight increase in the average path length has to be expected. Figure 10 represents the behavior of average hop length in the same experiment described above on the complete 1000-step time span. It can be observed that, while all shortest path computations result in the same value (lower horizontal line), and the increase of the static `RSNE` algorithm remains around the same level (about 5%), the incremental policies tend to suffer from a higher degradation (up to 17.3%) due to progressive abandon of the shortest path configuration. However, this problem can be fixed by triggering a shortest-path restart whenever the average hop length reaches a given threshold, or periodically.

7 Conclusions

We proposed new Load Balancing algorithms for Optical Networks based on IP-like routing and Local Search, where every move modifies a single entry in the routing table of a node. The comparisons between the new scheme and the optimum solutions found via an ILP solver show both calculation time savings

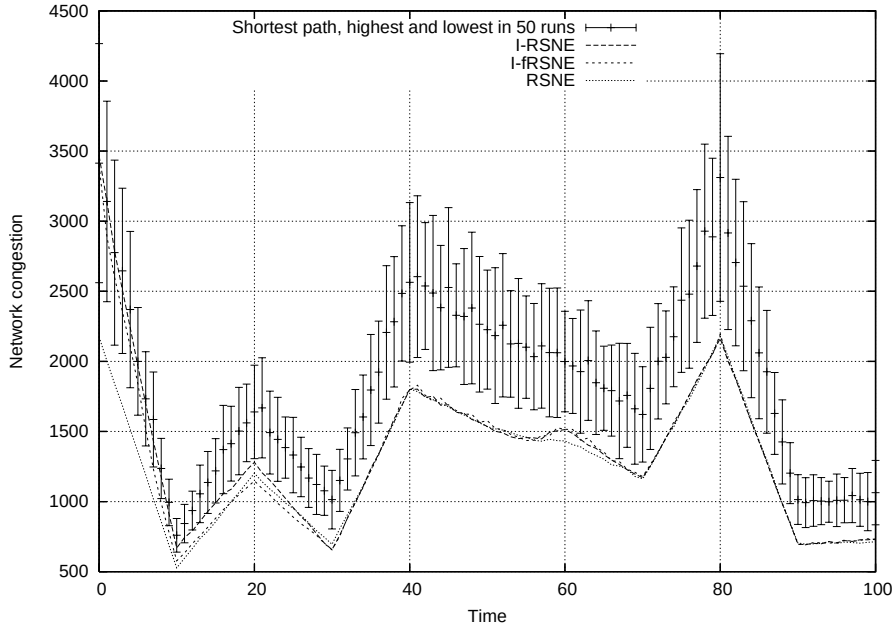

Figure 9: congestion in an online setting.

and comparable results of network congestion and of average length of the resulting routes. The incremental scheme `I-RSNE` produces congestion values that are almost undistinguishable from those of the `RSNE` algorithm. The randomized scheme called `fRSNE` has a substantially reduced complexity in comparison with `RSNE` and, again, very similar final congestion results.

Many extensions can be envisioned, in particular when considering specific properties of the optical medium. At the moment these schemes could work properly on Optical Packet Switching networks where IP-like routing is assumed or in wavelength routed networks where wavelength conversion at each node is considered. The context of networks having links with different capacities should also be considered in future extensions, as well as more general routing mechanisms (e.g. G-MPLS). Further investigation will determine how the randomized version could be exploited in a distributed environment, where complete information is not available at each node, in order to have a fast implementation on an asynchronous network.

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. We also thank the anonymous referees for their suggestions that helped improving the quality of our work.

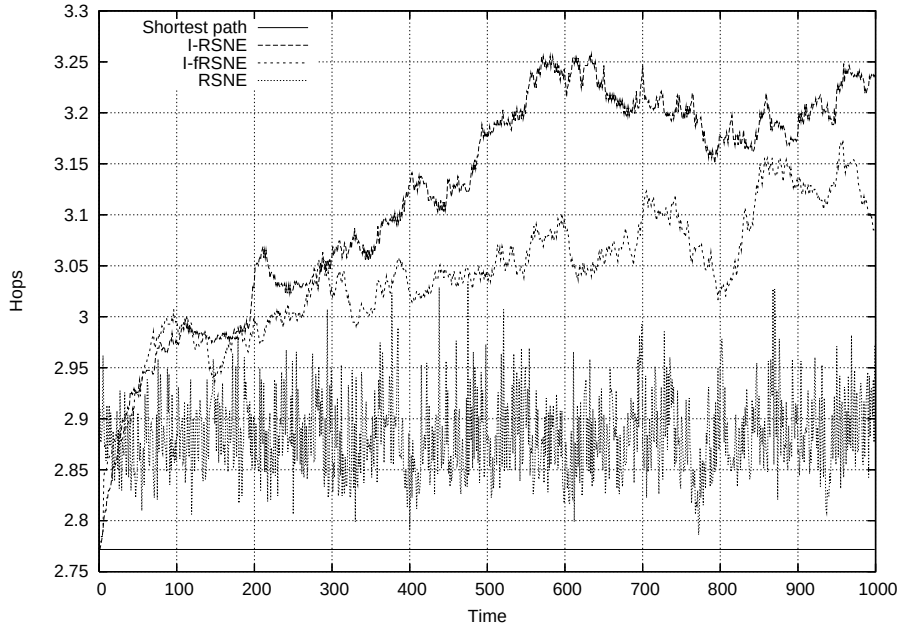

Figure 10: average hop length in a dynamic setting.

References

[1] Dhiritiman Banerjee and Biswanath Mukherjee. A practical approach for routing and wavelength assignment in large wavelength-routed optical networks. *IEEE Journal on Selected Areas in Communications*, 14(5):903–908, June 1996.

[2] S. Bregni, U. Janigro, and A. Pattavina. Optimal allocation of limited optical-layer-resources in WDM networks under static traffic demand. In *Globecom 2001 Proceedings*, pages 25–29, San Antonio, Texas, 2001.

[3] Mauro Brunato, Roberto Battiti, and Elio Salvadori. Load balancing in WDM networks through adaptive routing table changes. In E. Gregori, M. Conti, A.T. Campbell, G. Omidyar, and M. Zukerman, editors, *Networking 2002 Proceedings*, LNCS 2345. Springer, May 2002.

[4] 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.

[5] 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.

[6] Aura Ganz and Xudong Wang. Efficient algorithm for virtual topology design in multihop lightwave networks. *IEEE/ACM Transactions on Networking*, 2(3):217–225, June 1994.

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

[8] Rajesh M. Krishnaswami and Kumar N Sivarajan. Design of logical topologies: A linear formulation for wavelength routers with no wavelength changers. *IEEE/ACM Transactions on Networking*, 9(2):186–198, April 2001.

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

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

[11] Ling Li and Arun K. Somani. Dynamic wavelength routing using congestion and neighborhood information. *IEEE/ACM Transactions on Networking*, 7(5):779–786, 1999.

[12] K. Lu, G. Xiao, and I. Chlamtac. Blocking analysis of dynamic lightpath establishment in wavelength-routed networks. In *ICC2002 Proceedings*, volume 5, pages 2912–2916, 2002.

[13] G. Maier, A. Pattavina, L. Roberti, and T. Chich. Static-lightpath design by heuristic methods in multifiber WDM networks. In *Opticomm 2000 Proceedings*, pages 64–75, Dallas, TX, 2000.

[14] 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.

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

[16] 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.

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

[18] D. A. Schupke and D. Sellier. Lightpath configuration of transparent and static WDM networks for IP traffic. In *ICC2001 Proceedings*, 2001.

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

[20] Hongsuda Tangmunarunkit, John Doyle, Ramesh Govindan, Sugih Jamin, Scott Shenker, and Walter Willinger. Does AS size determine degree in AS topology? *Computer Communication Review*, 31(5), October 2001.

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

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

[23] 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.

[24] 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.