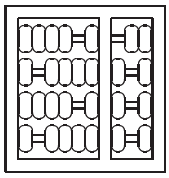

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

A Minimum Interference Routing Algorithm

G. B. Figueiredo *N. L. S. da Fonseca*
J. A. S. Monteiro

Technical Report - IC-03-014 - Relatório Técnico

May - 2003 - Maio

A Minimum Interference Routing Algorithm

Gustavo B. Figueiredo* Nelson L. S. da Fonseca† Jos A. S. Monteiro‡

Abstract

Minimum Interference Routing is instrumental to MPLS Traffic Engineering under realistic assumptions of unknown traffic demand. This work presents a new algorithm for minimum interference routing, called Light Minimum Interference Routing (LMIR). This algorithm introduces a new approach for critical link identification that reduces the computational complexity. Results, derived via simulation, show that LMIR is precise and has low computational complexity.

1 Introduction

The increase of link bandwidths in the Internet has not completely eliminated congestion since it can be caused by unbalanced network traffic [1]. Therefore, Traffic Engineering which aims at mapping flows into a physical structure, plays an important role in the provisioning of Quality of Service for the Internet.

MultiProtocol Label Switching (MPLS) has been recognized as a fundamental tool for traffic engineering since it represents the convergence of two paradigms: connection oriented forwarding and the connectionless model used in IP networks. The ability to dynamically create Label Switched Paths (LSPs) at low operational costs is key to Traffic Engineering. Dimensioning and traffic partitioning among LSPs are the major challenges for MPLS based traffic engineering.

The selection of paths which satisfy multiple independent QoS requirements is an NP-complete problem. Therefore, the solution of QoS based routing problems in the Internet involves the adoption of heuristics to find feasible paths in real time.

Path selection for a given source-destination (SD) pair traffic may follow different criteria. In algorithms that follow the minimum interference criterion path, selection try to minimize the blocking probability of requests for future path establishment.

Several proposals have been made in order to solve the minimum interference routing problem. This paper presents a new heuristic, called *Light Minimum Interference Routing* (LMIR), which tries to optimize resource utilization with low computational complexity. As the existing minimum interference algorithms, LMIR is an on-line algorithm independent from traffic patterns. Its major advantage in comparison to previous solutions is its low

*Institute of Computing, State University of Campinas,
†Institute of Computing, State University of Campinas
‡NUPERC, Salvador University

computational complexity which leads to greater scalability. The rest of this paper is structured as follows. Section 2 presents the existing minimum interference routing algorithms. Section 3 presents LMIR and analyze its asymptotical complexity. Section 4 presents examples of the use of the LMIR algorithm. Finally, conclusions are presented in Section 5.

2 Minimum Interference Routing Algorithms

Minimum interference routing algorithms have attempted to reduce network blocking probabilities by finding paths which minimizes the maximum flow reduction of SD pairs.

The central idea behind these algorithms is that the larger the available maximum flow, the smaller is the blocking probability of requests for an SD pair. Therefore, minimum interference algorithms try to route a bandwidth request into paths which maximizes the available maximum flow of others SD pairs [2]. This is an NP-hard problem, and heuristics have been proposed to deal with this problem.

The *Minimum Interference Routing Algorithm* - MIRA [2], which firstly introduced the terminology "minimum interference", assumes that both SD pairs and the network topology are known. Besides, it assumes the existence of a signaling protocol, which is responsible for the diffusion of the network topology, the residual band and route information.

The available maximum flow between an SD pair is known as *maxflow* and there is a minimum cut set associated with it [3]. The maxflow of a given SD is reduced when the available bandwidth in an link/edge belonging to the minimum cut set, is reduced. Whenever a new request is accepted between a given SD pair, the maximum available flow between this pair is reduced since the accepted flow has to be routed through one of the edges belonging to its min-cut set. The idea behind MIRA is to avoid routing the requests through edges which belongs to the min-cut set of other SD pairs.

MIRA may be described as follows. Let $G = (V, E)$ be a graph, with $|V|$ vertices and $|E|$ edges. Let B be the set of residual link capacities and let $r(a, b, bw)$ be a request for bw bandwidth units between SD pair (a, b). MIRA returns a path between (a, b) with bandwidth bw, if available.

MIRA is organized in three steps. The first step involves the maxflow computation between all SD pairs $(s, d) \in P \setminus (a, b)$, where P is the set of all SD pairs. The second step is the identification of critical edges and the allocation of weights to them. The weights are computed according to the following expression:

$$w(l) = \sum_{(s,d):l \in C_{sd}} \alpha_{sd}, \qquad \forall l \in E \tag{1}$$

where α_{sd} is the weight of the (s, d) pair and C_{sd} is the set of critical links. The use of weights for SD pairs are specially useful for assigning priorities to them. The third step of the algorithm is the application of Dijkstra algorithm using $w(l)$ as edge weights.

Another minimum interference algorithm was proposed by Wang, Su and Chen [4]. Their algorithm is based on MIRA and make the same assumptions, i.e., the existence of

specialized protocols to propagate state information and network topology information. It differs from MIRA in critical edge identification and in the cost function.

According to Wang et al. [4], the drawback of MIRA is that the critical edges detection is for a single SD pair which prevents the detection of the criticality of a given edge to a group of pairs, leading to the denial of several requests in specific topologies such as in graphs with concentrating or distributing edges. The authors then proposed a way to identify the impact of using a set of edges in future requests. The idea is to compute the edge contribution with respect to maxflow between an SD pair, avoiding edges with less contribution. Edge weights are computed as

$$w(l) = \sum_{(s',d') \in P} \frac{f_l^{s'd'}}{\theta^{s'd'} \cdot R(l)}, \qquad l \in E \tag{2}$$

where $R(l)$ is link l residual capacity, $\theta^{s'd'}$ represents the maximum flow between pair (s', d') and $f_l^{s'd'}$ is the contribution of edge l for the maximum flow of (s', d').

Once the weights have been computed, the edges with residual capacities smaller than the requested bandwidth are eliminated and Dijkstra's algorithm is executed using $w(l)$ as weights [4].

Critical edges detection is, certainly, the crucial task in both algorithms presented in this section since it determines their efficiencies. Both algorithms use maxflow for critical edges detection. MIRA [2] uses the minimum cut set associated with the maxflow to perform this detection, while Wang, Su and Chen's algorithm [4] detects critical edges by the contribution of each of them in relation to the maxflow.

The complexity of both algorithms is increased by the need to run the maxflow algorithm, which is a prohibitively expensive task for medium to large networks. The most common maxflow computation method is *Ford-Fulkerson*'s [3], and its mostly known implementation, *Edmonds-Karp* algorithm, uses a breadth search to find the augmenting paths. Its complexity is $O(V \cdot E^2)$.

For networks such as the Internet, the average degree is around 3.5 [5]. In other words, the number of edges is significantly larger than the number of vertices, which makes *Edmonds-Karp*'s algorithm perform near the worst case (dense graphs). The fastest maxflow algorithm is Goldberg's [6]. Its complexity is $O(\min(V^{2/3}, E^{1/2}) \cdot E \cdot \log(V^2/E) \cdot \log(U))$ in a network with $|V|$ vertices, $|E|$ edges with capacities in the interval [1,U] [6]. Even though Goldberg's algorithm shows a significant reduction in computational complexity, other approaches that avoid the use of maxflow may be used to reduce even more the complexity of minimum interference algorithms, such as the approach taken in this paper.

Besides the maxflow computation, the minimum interference algorithms detects edges criticality, computes weights and executes Dijkstra's algorithm which is $O(V^2 + E)$. However, the computational complexity of these steps are lower than the complexity of the maxflow algorithm.

3 The Light Minimum Interference Routing Algorithm

The need to execute Maxflow algorithms impacts significantly the complexity of minimum interference routing algorithms. In this section, a new minimum interference routing algorithm, called *Light Minimum Interference Routing (LMIR)*, that does not involve the execution of maxflow algorithms, is presented.

LMIR assumes the existence of signaling protocols (e.g., RSVP-TE or CR-LDP) responsible for the residual bandwidth information updating, $R(l)$, and topology changes. LMIR attempts to find the K paths with the lowest capacities among all SD pairs before determining critical edges. LMIR neither assumes any knowledge about future requests nor any statistical traffic profile.

The central idea is that SD paths with the lowest capacities contain the edges with the lowest capacity, i.e., the critical edge. Therefore, finding the K worst paths implies in finding the K critical edges for the corresponding SD pair. LMIR finds the K critical edges among each SD pair and tries to avoid them.

LMIR seeks the paths which minimize the interference of SD pairs, generating a balanced utilization of its resources. Upon receiving a request for bw bandwidth units for an SD pair (s, d), $ri(s, d, bw)$, LMIR finds the K critical edges for the other SD pairs. This search for critical edges is performed by finding the K paths with the lowest capacities. Given that the path flow is limited by the minimum capacity edge, these K paths contain the K critical edges.

After finding K paths with the lowest capacities, weights, computed according to Equation (2), are assigned to all of their edges. In this way, weights inversely proportional to the residual capacities are assigned to the critical path edges. Lastly, Dijkstra's algorithm using $w(l)$ as weights is executed for selecting non-critical edges or edges with low criticality. The LMIR algorithm is formally presented in Algorithm 1.

Algorithm 1 LMIR

INPUT

A residual graph $G = (V, E)$ and $ri(s, d, bw)$, a request for bw bandwidth units between pair (s, d).

OUTPUT

A path (route) connecting s to d with bw bandwidth units.

LMIR

1: Find the paths with the K lowest capacities $\forall(s', d') \in \delta$.
2: Compute the weights according to equation (2) for all the edges belonging to the paths found.
3: Eliminate all the edges with residual capacities less than bw.
4: Run *Dijkstra*'s algorithm using $w(l)$ as the weights.
5: Create an LSP connecting s to d and update the edge capacities.

The first step of LMIR is the search for the paths with the lowest capacities (where δ is the set of SD pairs). This is done by a variation of Dijkstra's algorithm, called "LowestCapacities" (Algorithm 2).

Algorithm 2 LowestCapacities

```
1: for all ( v ∈ |V|) do
2:    D[v] = ∞
3:    di[v] = ∞
4:    π[v] = NIL
5: D[s] = 0.0
6: di[s]= 0.0
7: Q ← s
8: while (Q) do
9:    u ← extract_min(Q)
10:   for all v ∈ ADJ[u] do
11:      if (γ ≤ D[v]) ∧ (di[u] < di[v]) then
12:         D[v] = γ
13:         di[v] = di[u] + 1
14:         π[v] = u
15:         Q ← Q ⋃ v
```

In LowestCapacities, v represents any network vertex, D is a vector which contains the minimum capacity found in any path from s to v. Vector π contains v's predecessor vertex and vector di stores the distance from s to v in number of edges. Q represents the list of vertices which adjacencies were not yet visited. Function *extract_min(Q)* returns the element $u \in Q$ which value $D[u]$ is the lowest and $ADJ[v]$ represents the adjacency list of vertex v.

As for the minimum interference algorithms presented in section 2, the identification of critical edges is the task which mostly impacts LMIR computational complexity. The major difference among the complexity of the three minimum interference routing algorithms, presented here, is the identification of critical edges. As previously mentioned, LMIR uses the *LowestCapacities* algorithm to find the critical edges.

The complexity analysis of LMIR is as follows. Step 8 of the *LowestCapacities* algorithm is executed $|V|$ times, since all vertices are visited. Step 10 is also executed $|V|$ times resulting in $O(V^2)$ complexity. Moreover, *extract_min(Q)* has complexity $O(E)$ in the worst case, when the queue of vertices which adjacencies not yet visited is implemented as a linked list. Therefore, LMIR complexity is $O(V^2 + E) = O(V^2)$ [3]. Since each pair executes this algorithm K times, the complexity of identifying critical edges is $O(K \cdot V^2) = O(V^2)$. The other minimum interference algorithms, presented in this paper, have $O(\min(V^{2/3}, E^{1/2}) \cdot E \cdot \log(V^2/E) \cdot \log(U))$ complexity in the critical edges identification step.

In order to show that the complexity of *LowestCapacities* is lower than the complexity of Goldberg et al.'s, which is the fastest maxflow algorithm known, a complexity analysis for both algorithms for dense and sparse graphs are derived, separately.

For dense graphs it is assumed that $|E| = |V|^2$. Therefore, Goldberg's algorithm has complexity

$$O((V^{2/3}) \cdot V^2 \cdot \log(V^2/V^2) \cdot \log(U))$$

while *LowestCapacities* has complexity $O(V^2)$. Assuming that the term $\log(V^2/V^2)$ has omitted other terms, since the complexity cannot be zero, and that $\log(V^2/V^2)\cdot\log(U)) > 1$, it follows that

$$\text{Goldberg's } \textit{maxflow } = V^{2/3} \cdot V^2 = V^{8/3}, \text{ and}$$

$$V^{8/3} > V^2 = \text{LowestCapacities}.$$

Hence, $V^2 = O(V^{8/3}) \Rightarrow \text{LowestCapacities} = O(\text{Goldberg's } \textit{maxflow}) \square$

For sparse graphs it is assumed that $|E| = |V|$. In such type of graphs, Q can be implemented as a heap [3]. Therefore,

$$\textit{LowestCapacities} = O(V \cdot \log V),$$

since step *extract_min(Q)* has complexity $O(\log V)$. Besides, assuming that $\log(V^2/E) \cdot \log(U) > 1$, taking the first two factors of $O(\min(V^{2/3}, E^{1/2}) \cdot E \cdot \log(V^2/E) \cdot \log(U))$, it can be shown that

$$\text{Goldberg's maxflow } = V^{1/2} \cdot V, \text{ and}$$

$$\textit{LowestCapacities} = O(V \cdot \log V).$$

Therefore, discarding factor V in both algorithms and taking the log

$$\textit{LowestCapacities} = \log(\log V), \text{ and}$$

$$\text{Goldberg's maxflow } = \log V^{1/2} = 1/2 \cdot \log V.$$

Hence, $\log(\log(V)) = O(1/2 \cdot \log V) \Rightarrow \textit{LowestCapacities} = O(\text{Goldberg's maxflow})\square$

These results show that for both dense and sparse graphs, the critical edges detection in LMIR is less expensive than in the algorithms that use maxflow.

4 Numerical Examples

Simulation experiments were pursued in order to verify the precision and performance of the proposed algorithm. The topology used in the experiments is shown in Figure 1. This same topology was used in [4] and [2]. SD pairs $(S1, D1), (S2, D2), (S3, D3), (S4, D4)$, and $(S5, D5)$ are shown in the figure and are the only network input and output traffic. The lighter links have capacity of 1200 bandwidth units, whilst the darker links have 4800 bandwidth units. These values represent OC-12 and OC-48 rates, respectively.

Each link in the figure is bi-directional, i.e., it represents two links with the same capacity in opposite directions. Requests were generated randomly and are uniformly distributed in the interval $[1, 4]$.

In the figures shown in this section, the minimum interference algorithms will be denoted by ICC [4], MIRA [2], and LMIR. Two more algorithms that do not take interference into account were also investigated in the simulation studies to better evaluate the benefits of the interfering approach. The Minimum Hop Algorithm (MHA) is an implementation of Dijkstra's algorithm while the Widest Shortest Path Algorithm (WSP) finds the maximum

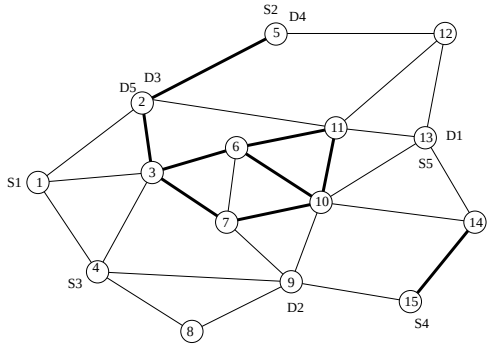

Figure 1: Topology used in the simulations.

capacity path in the network. If there is more than one path with the same capacity, WSP returns the path with the least number of edges.

It is assumed that the LSPs have long lives, i.e., once accepted its resources are kept until the end of the experiments. 8,000 requisitions were randomly generated among the five SD pairs shown before.

It is known that the lower the rate reduction of the maximum flow, the lower is the blocking probability. Figure 2 shows the network maximum flow reduction as a function of the number of arrivals. It can be seen that minimum interfering routing algorithms produce the lowest reduction rate for the total flow. WSP and MHA produced steeply reductions since they use critical edges indiscriminately, causing the reduction of the maxflow of several SD pairs, which demonstrates the importance of minimum interference algorithms. In this example, LMIR has a maximum flow reduction rate smaller than the reduction produced by ICC and it is equal to the one produced by MIRA up to 3,000 requests. From this point on, LMIR has decreased slightly more accentuated than both of them until 5,000 requests. After that, the behavior of all the algorithms are about the same, almost linear.

Figure 3 shows the number of requests rejected as a function of the total number of requests which arrived to the network. MHA is the algorithm that rejects the highest number of requests. MHA always chooses the shortest path, saturating rapidly the edges that belong to this path. WSP starts rejecting connections around 5,000 requests. However, because it does not take into consideration edge criticality, it can choose paths that have critical edges from various pairs causing their saturation. LMIR and MIRA produce the lowest number of rejected requests.

Figure 4 shows the results of an experiment designed to analyze the interference suffered by pair $(S1, D1)$ when requests from other pairs are routed in the network. Connection requests arrive randomly for pairs $(S2, D2)$, $(S3, D3)$, $(S4, D4)$, and $(S5, D5)$. It is intended to verify the influence of these requests in the $(S1, D1)$ flow.

It can be seen that MHA and WSP, which do not take interference into account cause a maxflow reduction between $(S1, D1)$ even before 1,000 requests. Since MHA uses always the shortest path, saturation is reached immediately after 2,000 requests. From there on, the path becomes saturated and there is no more reduction. Even though WSP causes interference, it takes longer to saturate than MHA. This happens because it chooses, other

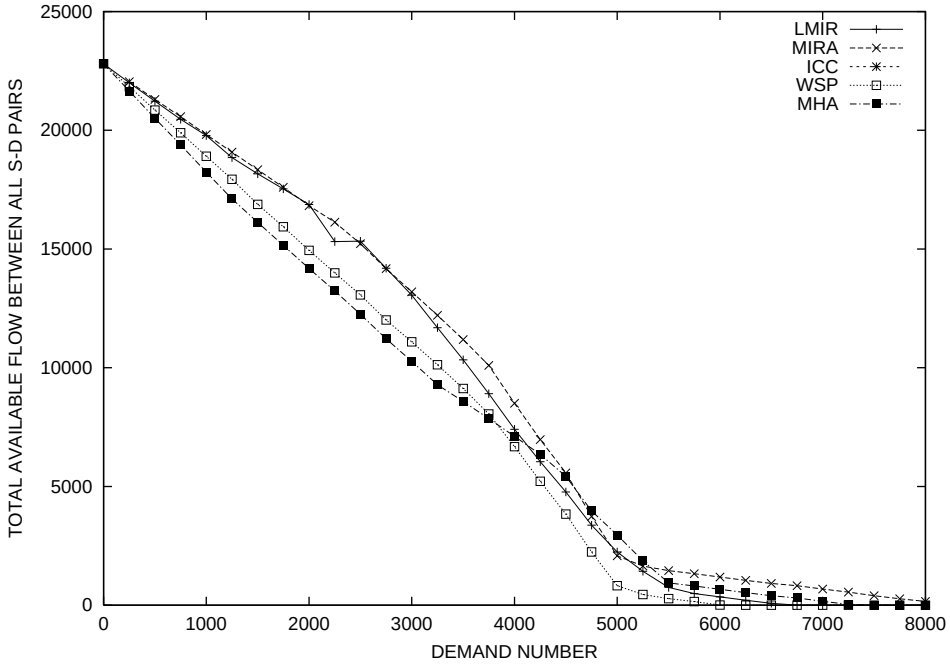

Figure 2: Total bandwidth among all network source-destination pairs.

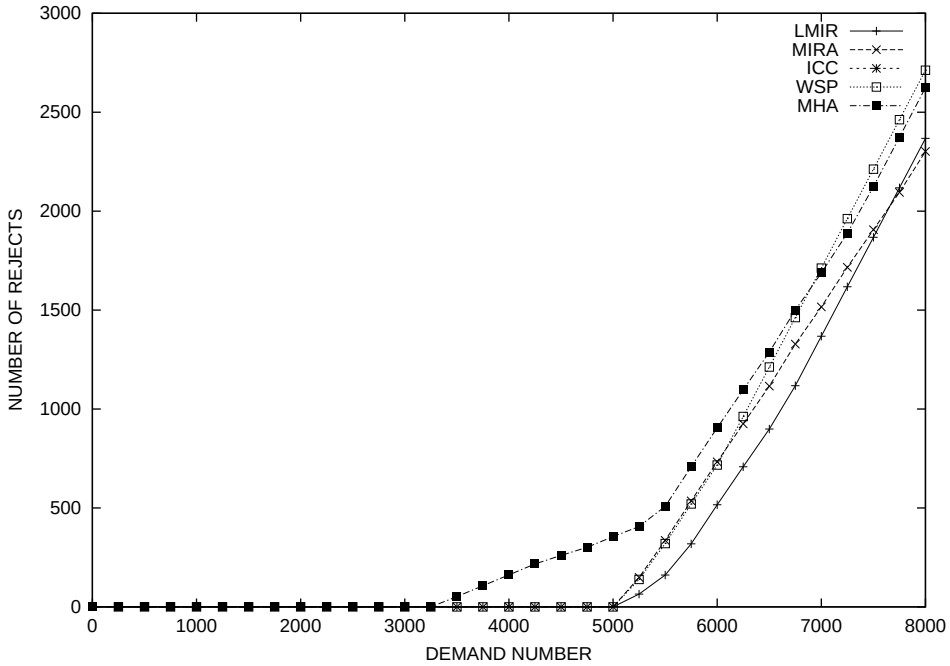

Figure 3: Number of rejections.

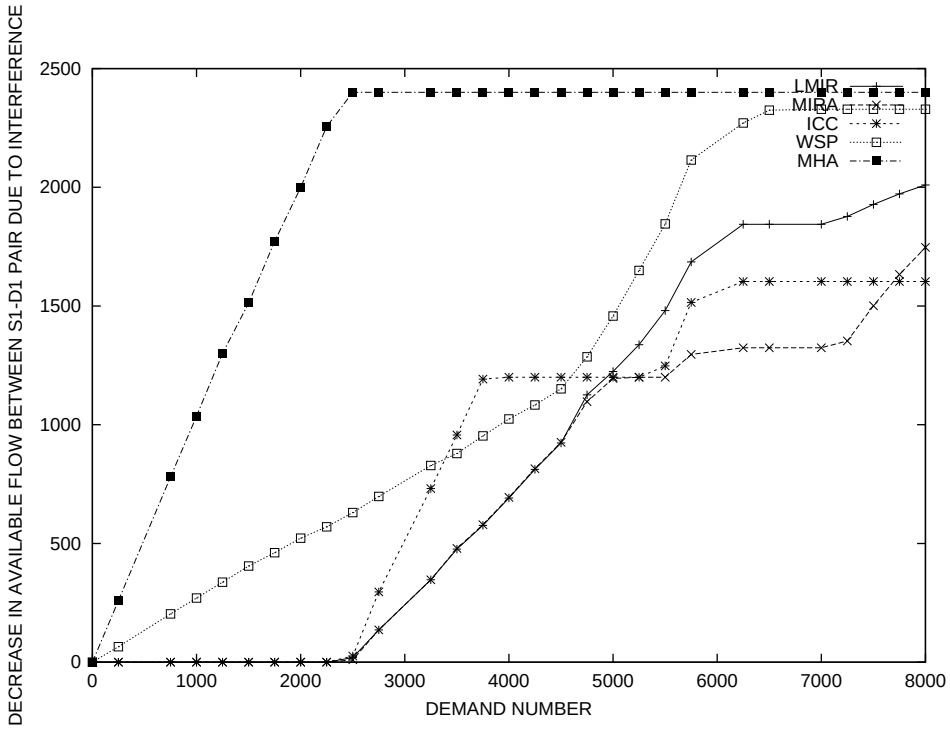

Figure 4: Interference suffered by pair $(S1, D1)$.

paths with higher capacities, as long as the path is saturated. A higher degree of interference when using WSP is noticed between 4,500 and 6,500, where there is a curve stagnation until 8,000 requests.

Interference is noticed only after the arrival of 2,500 requests, for the minimum interference algorithm, i.e., there is no maxflow reduction for $(S1, D1)$ up to this arrival. After that, LMIR and MIRA have better performance until very close to 5,000 requests. Even though causing accentuated reduction in $(S1, D1)$ maxflow right after 2,500 requests, ICC achieves better performance in the long term because it reaches a better stabilization than LMIR and MIRA. LMIR and MIRA have similar performances up to 5,000 requests, which can be explained by the fact that up to that point, the K critical edges seen by LMIR constitute min-cut sets since the critical edges would be the same for both algorithms. After 5,000 requests MIRA finds a low number of paths which do not affect $(S1, D1)$. LMIR keeps causing interference, even slightly up to 8,000 requests. After 8,000 requests there is a significant difference between LMIR and WSP. LMIR was the minimum interference algorithm which caused the greater interference and WSP was the algorithm with better performance that do not take interference into consideration. Note that in this experiment only the interference in the $(S1, D1)$ pair is considered. It is important to emphasize that LMIR produces better performance than ICC does when the interference among all SD pairs is accounted, which can be seen in Figure 2.

LMIR finds the K paths with the lowest capacities for each pair. The choice of this parameter can change the algorithm performance. Figure 5 shows the number of rejected requests as a function of K. It can be seen that the minimum number of rejected requests,

Table 1: LMIR execution times for 8,000 requests.

	ICC	MIRA	LMIR k=1	LMIR k=2	LMIR k=3	LMIR k=4	LMIR k=5	LMIR k=6
`Time`	7.23s	7.24s	5.017s	5.67s	6.54s	6.94s	7.01s	7.14s

in this topology, was reached for $K = 5$. Increasing this value, the paths starts to overlay and critical edge identification becomes difficult.

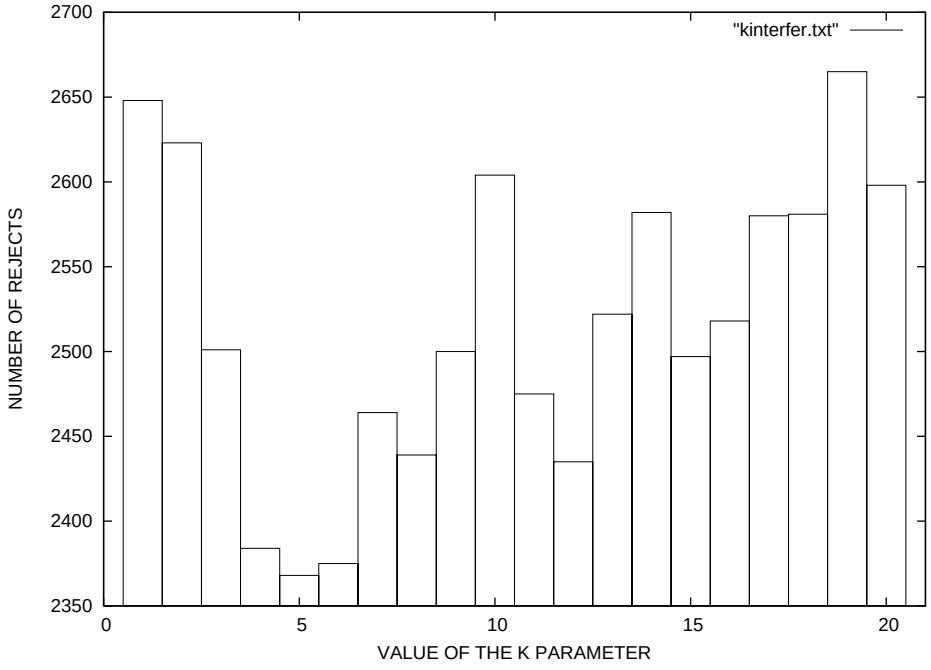

Figure 5: Influence of parameter K in the number of rejected connections.

Table 1 shows the execution times for the algorithms with 8,000 requests among all pairs. The values were obtained using Linux's *time* command, in Intel Celeron machines with a 1.2 GHz clock, and 256 MB of RAM.

As it was expected, it can be noticed that the time spent running LMIR is less than the time to run ICC and MIRA. Besides, LMIR has the advantage of offering a trade-off between speed and precision, depending on the choice of K. For values greater than 7, paths are overlayed and the detection of critical edges becomes more difficult, deteriorating the algorithm performance. WSP and MHA have execution times around 4 seconds. Even though they are faster, because these algorithms do not take interference into consideration, they cause an excessive number of rejections and for this reason they were omitted.

5 Conclusions

In this paper, a new minimum interference routing algorithm which does not use maxflow for critical edges detection was presented.

Modifications in Dijkstra's algorithm were pursued in order to find the paths with lowest capacities. After finding K critical edges (edges which limit the flow of the shortest paths), weights are assigned to the edges and an algorithm to find the shortest path is applied.

Results obtained via simulation show that the LMIR precision is similar to the other minimum interference routing algorithms. However, LMIR presents lower computational complexity.

References

[1] G. Banerjee and D. Sidhu, "Comparative analysis of path computation techniques for MPLS traffic engineering," *Computer Networks*, vol. 40, no. 1, pp. 149–165, 2002.

[2] M. S. Kodialam and T. V. Lakshman, "Minimum interference routing with applications to MPLS traffic engineering," in *INFOCOM (2)*, 2000, pp. 884–893. [Online]. Available: citeseer.nj.nec.com/kodialam00minimum.html

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press and McGraw Hill, 1990.

[4] B. W. X. Su and C. P. Chen, "A new bandwidth guaranteed routing algorithm for MPLS traffic engineering," in *Proceedings of IEEE International Conference on Communications*, April 2002.

[5] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *IEEE Infocom*, vol. 2. San Francisco, CA: IEEE, March 1996, pp. 594–602. [Online]. Available: citeseer.nj.nec.com/zegura96how.html

[6] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," in *J. ACM 45 (5)*, 1998, pp. 783–797.