Routing with Deceptive Information

Routing with Deceptive Information

C.Casetti, M. Mellia, M.Munafò
Politecnico di Torino - Torino, Italy
Email: {lastname}@tlc.polito.it

C.Racca
Top-ix, - Torino, Italy
Email: christian.racca@topix.it

Abstract— **Routing protocols based on the link-state paradigm notoriously suffer from the problem of stale information in highly dynamic traffic scenarios, leading to an overall loss in routing efficiency. Solutions are not easy to come by, since an increase in the frequency of link state advertisements is equally dangerous: route flapping (i.e., periodic route changes that force traffic to be routed through an alternately underloaded set of paths) is one of the main drawbacks. In this paper we propose a novel, yet simple link state mechanism that may deliberately advertise false link state information with the purpose of stabilizing the routing, while keeping a high network utilization.**

Keywords: Link-state Routing, Traffic Engineering

I. Introduction

In the early years of packet networks, a hot research topic was the definition of efficient *dynamic* routing algorithms, aiming at an optimal exploitation of links by adapting packet routes to instantaneous traffic conditions. Dynamic routing algorithms can in principle offer significant advantages with respect to static routing, since they automatic react to congestion situations, therefore offering better performance and quality of service (QoS). However, the finding that dynamic routing algorithms may lead to route flapping (i.e., a periodic route changes which forces traffic to be routed through an underloaded set of paths, causing a sudden overload of new paths and underload of previous paths) and to the consequent performance degradations, has limited their diffusion. As a consequence, the dynamic features implemented today in routing protocols are mostly limited to automatic reactions to topology changes due to link failures or infrastructure updates.

In recent years, dynamic routing algorithms have again attracted the attention of the networking community. Several aspects have been investigated, such as: i) protocol convergence [1], [2], ii) overhead impact [3], [4], iii) implementation issues [5], [7], [8], iv) impact of update policies [3], and v) performance issues [7], [8], [9], [10], [11], [12].

Despite all the effort spent by the research community, no QoS routing has ever been deployed in the Internet. Several reasons are behind this choice, like traffic unpredictability, protocol complexity, and increased overhead. The last item is particular critical, since each dynamic QoS routing relies on the network status information which must be shared among nodes. Therefore this increases both network load (in term of signalling informations) and node power (in term of computational processing). Indeed, compared to static routing protocol, which relies on static information, dynamic algorithms rely on the knowledge of the current status of the network. The amount of additional overhead depends clearly on how fast the network QoS parameters adopted by the QoS routing algorithm change, which in turns depends on the traffic fluctuations and user behavior. In this paper we therefore investigate how update policies affect the QoS routing algorithm performance, quantifying the intuitive results that if route selection performed by nodes is based on stale information, the QoS routing algorithm may provide worse performance compared to traditional static routing. To overcome this limitation, we propose a novel yet very simple mechanism that is shown to improve performance of QoS routing algorithms when the state information is slowly updated.

Simulation results show that our proposal extends the benefit of QoS routing algorithms even in case of rare link state update, while effectively limits the route flapping.

II. Problem Outline

We extended ANCLES [13], a connection-level simulator that was previously developed at the Politecnico di Torino. Originally conceived for ATM networks, ANCLES gradually evolved to a generic connection-level simulator, where traffic sources request connections and the network performs all the actions required to manage them. The reader interested in the simulation tool is referred to [15].

A. Modeling data connections

The traffic the simulator models is of elastic nature [14] (such as that generated by data connections). Specifically:

- connections are characterized by a given amount of data to transfer;
- the amount of resources the connection can exploit at any given time is the minimum between a target bit rate for the connection, e.g., the access line speed, and what the connection would be assigned under ideal max-min fair resource sharing;
- the connection holding time depends on the resources it receives throughout its "life" which determine the amount of data already transmitted at any given time;
- if the resources usable by a connection fall below a given threshold, the connection may be shut down prematurely, mimicking the behavior of impatient users.

Each connection attempts to perform a bulk data transfer whose size is randomly chosen from an exponential distribution with average 2.5 MBytes (20 Mbit). Traffic is uniformly generated by all sources and evenly distributed among all possible destinations. Results are collected over different load situations.

0-7803-9569-7/06/$20.00

The performance metrics we consider are:

- the average throughput of connections that complete the data transfer;
- the fraction of connections that are shut down due to lack of resources (starvation effect);

To model the starvation effect [14], a starvation threshold B_t is set to 50 kb/s and used to identify starved connections: if the current per-connection bit rate estimate on a bottleneck link drops below B_t, then a connection on that bottleneck is randomly picked and shut down. This is repeated until the sending rate of starved connections raises above the threshold. This allows us to define the *starvation probability* P_s as the ratio between connections that are prematurely aborted and the total number of connections that entered the network.

B. Routing Algorithms

As regards routing algorithms implemented in the simulator, beside the static, hop-count based algorithms, that are unable to cope with the variation of available bandwidth in the network, ANCLES implements several dynamic, traffic-driven routing algorithms, like those proposed in [9], [10], [16]. Considering the available bandwidth in each link in the path lookup procedure, these algorithms can offer a better choice for the routing of connections with QoS requirements. Here, we briefly describe the two algorithms used in the simple performance evaluation carried out in this work:

- *Shortest-Path* (*SP*): for each source-destination pair, the algorithm determines the path with the minimum hop count and routes flows over that path. If two or more "shortest" paths exist, the algorithm would choose one at random and, once and for all, would route all flows over it. This is the routing algorithm commonly used in the current Internet.
- *Minimum-Distance* (*MD*): for each source-destination pair, the path P is chosen which minimizes the quantity: $D(P) = \sum_{l \in P} \frac{1}{b_l}$ where b_l is the max-min fair bandwidth that is available to a new connection over link l belonging to path P [10].

The choice of the MD algorithm was made because it was found to provide good performance among traffic-driven routing algorithms [16]. Indeed, the focus of the paper is on the effect of the information distribution on QoS routing algorithm performance, and not on the QoS routing algorithms themselves.

For consistent routing to be preserved, we assume that the forwarding procedure is integrated so as to signal the routing decision taken at upstream nodes. This solution can be implemented, for example, using the route-pinning property of MPLS. Thus, connections routed on path stick to it even though future connections between the same source-destination pair are routed over a different set of links.

C. Modeling the propagation of LSAs

In our event-based simulation, each node generates an LSA if one of the following two event occurs:

- *triggered update*: there is a sizable change in the state of one or more links;
- *periodic update*: a timeout expires

The first event is triggered whenever a node detects that the relative value of the difference between the stored value of utilization of a link and its actual value is larger than a quantity, referred to as *lsa_threshold*. To limit the amount of triggered update per unit of time, each node periodically checks the state of its links to verify if the *lsa_threshold* has been exceeded (and, thus, imposing a minimum interval between two LSA generations).

The second event is a user-defined timeout, referred to as *periodicUpdate*.

In our simulations, the *lsa_threshold* was set to 90%, while we experimented with different *periodicUpdate* values.

Once an LSA is generated, it is flooded across the network. First, the originating node starts queueing an LSA message in all its link queues, therefore reaching all its neighboring nodes. Then, upon the reception of each LSA message, each node double-checks if that LSA has been previously received. If this is true, no further action is required. Otherwise, the node queues a copy of the LSA message in all queues corresponding to a link directed to neighboring nodes, except for the link it has received the LSA message from.

The procedure provides for the LSA to reach all nodes, and for it (or the copies that were generated) to be discarded when all nodes have been reached. On top of this, a maximum lifetime is assigned to each LSA, so as to avoid that it is indefinitely delayed at one or more node queues, thus carrying stale information.

The propagation time of an LSA between two nodes is modeled by the following expression:

$$prop_{time} = \frac{link\_length}{2/3 \cdot C} + \frac{S_p \cdot Q}{link\_speed} + c_t \qquad (1)$$

where: C is the speed of light, S_p is the average packet size in bits, Q is the number of packets queueing ahead of the considered LSA in the queue, c_t accounts for the time spent in handling the packet headers (all other quantities are self-explaining).

It can be noted that the propagation time includes a "queueing delay" in the second term. Indeed, we have chosen to model each node as a simple $M/M/1/k$ queue, to account for the backlog of packets in the case of heavy-loaded links. Even if this is known not to be a good model for today's packet networks, the simplicity of the model, and the availability of a closed analytical solution to the performance indexes yields a limited, efficient complexity of the simulation.

Further, we have to account for the probability that an LSA is lost: to this end, we have included a loss probability in the simulator, associated to each node crossed by an LSA, and computed as:

$$\rho = \frac{(1-\rho)\rho^{k+1}}{1-\rho^{k+2}} \qquad (2)$$

where ρ is the link utilization and k is the buffer (queue) size. The previous expression is derived from standard queueing theory, and is the loss probability of an $M/M/1/k$ queue.

III. Performance evaluation

To compare the performances of the system, we considered a network scenario with a randomly generated 24-node topology with an average connectivity degree of 3. We also experimented with other random topologies, deriving similar results. The link lengths are also randomly generated, with a uniform distribution between 100 and 700 km so as to mimic a possible long distance backbone. Users generate connections that are upper-bounded at the source by a rate of 1 Mb/s. Therefore, given an average data size of 20 Mbit in an uncongested network, the mean duration of each connection is 20 s: this time can be used as a reference for other time intervals used in the system. Note that being the simulator time reference external parameter, all values selected in this scenario may be scaled without affecting the simulation results.

In this scenario we measured the average per-connection performances of the Shortest Path (labeled 'SP' in the figures) and Minimum Distance (labeled 'MD') algorithms for different values of offered load and LSA transmission timers. Offered load is normalized against the total access network capacity.

Besides the *periodicUpdate* timer, another time interval has been considered in our experiments, i.e. the *periodicPolling* timer, representing the time interval with which a node checks the state of its links to verify if the *lsa_threshold* has been exceeded (and, thus, the minimum interval between two LSA generations).

The values of the timers are shown near the MD with the notation *update/polling* time, so 'MD - 10/1.5 s' means an MD implementation with a *periodicUpdate* every 10 s and a *periodicPolling* interval of 1.5 s. We also considered a simpler implementation of the MD algorithm (labeled 'MD - ideal'), that assumes that each node has a precise indication of the link load at each time: thus, all nodes instantaneously have the same, correct snapshot of the state of the network. The Shortest Path algorithm, depending only on topological metrics, is not affected by the LSA generation frequency, so only one curve is necessary.

Figure 1 presents the average per-connection throughput, as a function of the normalized offered load, while the same results, normalized with respect to the SP case, are shown in Figure 2. These results provide a way of gauging the gains of the different strategies used for the MD algorithm over the "standard" Shortest Path. As expected, the ideal case performs best; however, by allowing as many as 40 seconds and 15 seconds of Periodic Update time and Periodic polling time, respectively, the performance remains quite close to the ideal case, and thus fully acceptable.

Normalizing the time reference with the average connection duration, we intuitively observe that if the LSA generation frequency is comparable with the traffic variation frequency, the performance of QoS routing are excellent. However, as LSA become rare with respect to the traffic variations, performance degrades quickly.

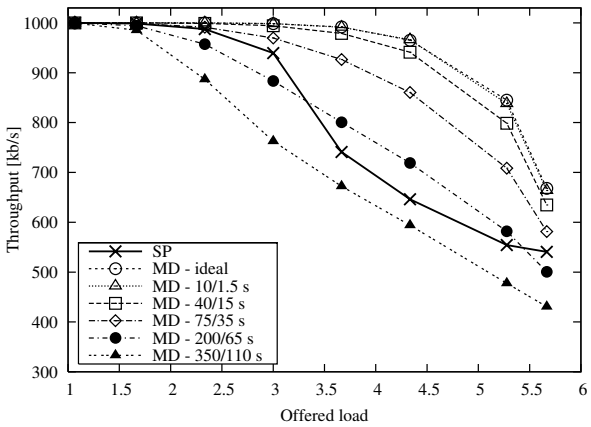

Fig. 1. Average per-connection throughput

Figure 3 depicts the average per-connection starvation probability. In this case, the update policy does not affect the performance of the MD algorithm: all its versions, regardless of the update times provide almost the same performance, and all present a starvation effect only at loads much higher than those for which the SP algorithm is starved.

Even if these results are quite positive in the comparison between the MD and the SP algorithms, the MD algorithm performance depends strongly on the values of the update timers: large timer values make the nodes work with stale information, so the routing decisions taken are not optimal and can reduce the per-connection user throughput below the equivalent SP levels.

Figure 4 shows an example of the number of active connections over a sample link for a 1000 s interval, in a scenario where the load was fixed at 3.7 and the timers set at 200 s (update) and 65 s (polling). The time evolution of the number of connections shows slow variations with high variance (with respect to the reported average number of connection over the same link). This shows that the stale bandwidth information due to distant LSAs tends to trigger period of link overloading, followed by periods of under-utilization, that will trigger future overloads when the corresponding LSAs are received. Apparently, LSAs advertising almost empty links tend to induce route flapping if the information they carry is too old.

IV. The DIE protocol: Deceptive Information Exchange

From the results presented in the previous section, we can foresee at least two solutions to the performance decrease experience by the QoS algorithm in presence of inaccurate information:

1) increase the frequency of updates when a link is congested;

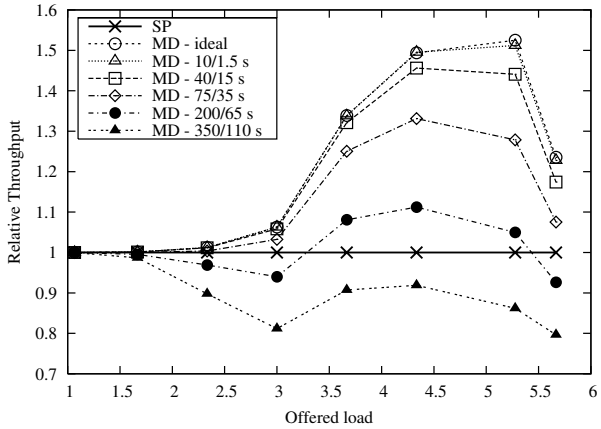

Fig. 2. Average per-connection throughput gain with respect to SP

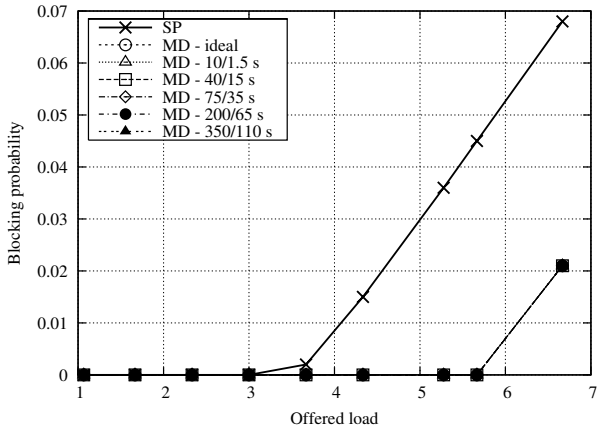

Fig. 3. Average per-connection starvation probability

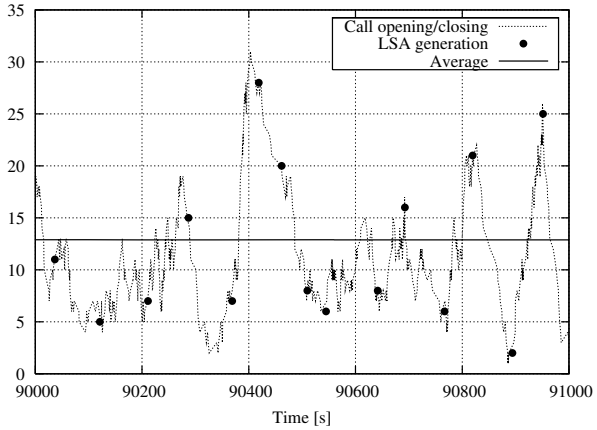

Fig. 4. Sample of number of connections on a generic link over time, using standard advertisement

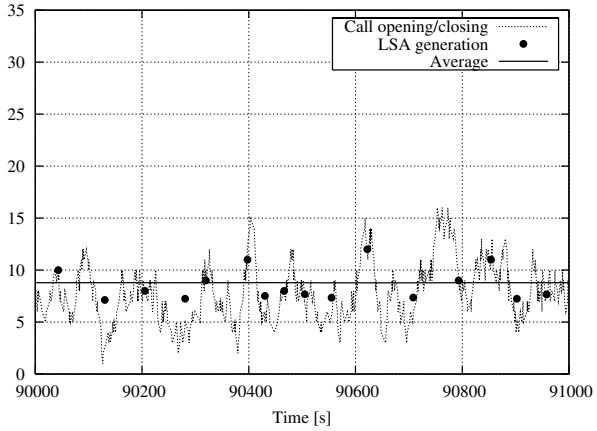

Fig. 5. Sample of number of connections on a generic link over time, using the DIE protocol

2) disseminate "doctored" LSA information with the purpose of decreasing congestion on overloaded links.

The first solution, though viable, would entail periodical LSA floods that increase the network and node overhead and might even worsen the problem. Moreover, being it related to traffic variations, it may be difficult to set the advertisement frequency. The second solution, which we are pursuing in this paper, is based on a simple line of reasoning. If a link is congested, it is sensible to inform all nodes, so that new connections are routed away from that link. If a link is uncongested, advertising its state might trigger a gold rush to that link by a great number of incoming connections; furthermore, if the link update period is larger than the arrival rate of new connections, the link might be easily overloaded in a short time.

A. Protocol description

We propose a modification to the LSA distribution mechanism, called DIE (Deceptive Information Exchange). In the following, we describe the implementation related to the MD algorithm. It can be generalized to similar traffic-driven routing algorithms by replacing the proper link-state metric.

According to DIE, the link state information of a generic link l that a router must advertise is altered by a threshold mechanism. In the case of the MD algorithm in our scenario, being $b_l = link\_speed/c_l$, where $link\_speed$ is fixed for all links and c_l is the number of connections on link l, the link-state metric was simply chosen as c_l. If the number of connections crossing a link is higher than a threshold, T_l, the LSA carries correct information on the link load; if it is lower or equal to the threshold, the LSA carries a *deceptive* link load, higher than the actual one.

The threshold T_l is computed by each node as running average of c_l for every link l, following the expression:

$$T_l[n] = \alpha \cdot T_l[n-1] + (1-\alpha) \cdot c_l[n]$$

where n is an index that associates the threshold calculation

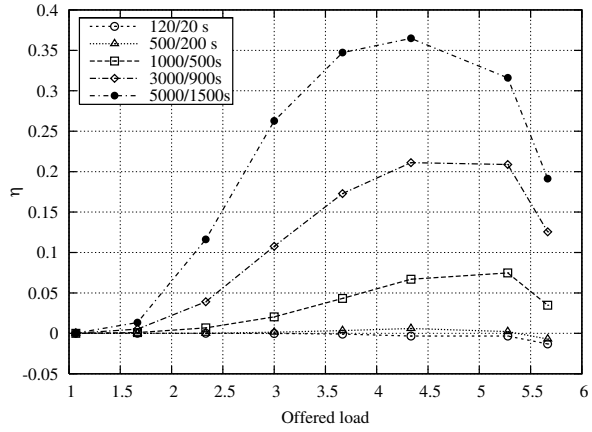

Fig. 6. Throughput gain for the DIE algorithm

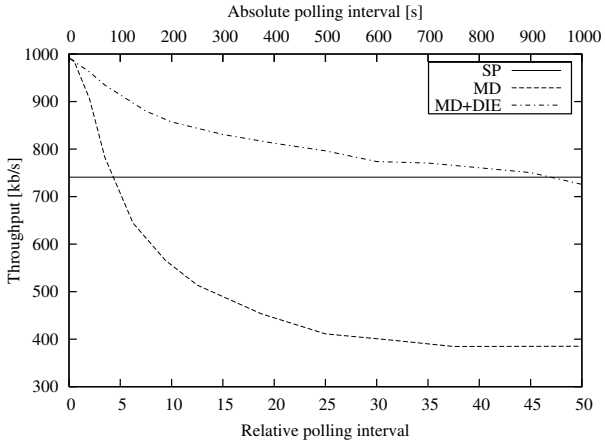

Fig. 7. Throughput versus update period.

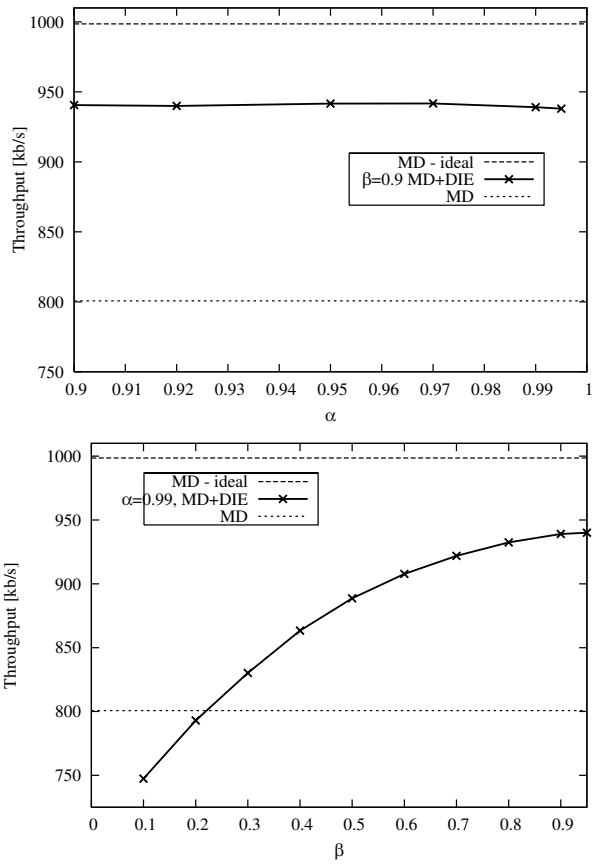

Fig. 8. Parameter sensitivity analysis.

to the issuing time of the n-th LSA by a router, while $c_l[n]$ indicates the actual number of connections on link l at the same time. α is chosen between 0 and 1.

We can now redefine our algorithm by stating that, if $c_l[n] > T_l[n]$ when the n-th LSA for link l must be transmitted, such LSA will carry the correct load information L; otherwise, a *deceptive* load information L' will be carried, determined as

$$L' = (1 - \beta)c_l[n] + \beta T_l[n]$$

depending on the value of $\beta \in [0, 1]$, L' advertises the actual c_l, or its average value T_l, as determined from recent samples.

B. Performance evaluation of the DIE protocol

Figure 5 shows the number of active connections for the same link and in the same scenario of Figure 4 with the difference that the DIE algorithm has been implemented, with $\alpha = 0.99$ and $\beta = 0.9$. Comparing Figures 4 and 5, it can be noticed that the average number of connections is lower when the DIE algorithm is used, indicating a larger bandwidth share, and its variability was reduced. Notice also how the number of connections advertised at the LSA generation time (black dots) is now different from the current number of connections when the LSA is generated in an underutilization phase.

In Figure 6 we compare the performance of the DIE algorithm with the unmodified LSA distribution plotting the relative gain $\eta = \frac{thr_{DIE} - thr_{old}}{thr_{old}}$, being thr_{old} the throughput in the unmodified version of the algorithm. The plots show an increase in the gain at the rarefaction of the LSAs, especially for medium/high load. At higher load, as usual, the network status is such that no algorithm can perform much better than SP [11]. The curves in the plot are for very high timer values, especially if compared with the average connection duration 20 s, showing that the algorithm is performing well even in high (and unrealistic) undersampling scenarios.

The gain introduced by the DIE algorithm can be also seen in Figure 7, where the average per-connection throughput is compared among the algorithms in a scenario where the network load is 3.7, $\alpha = 0.99$ and $\beta = 0.9$ in the DIE algorithm, and *periodicUpdate* and *periodicPolling* are the same. The plot shows the throughput vs. the timer durations and demonstrates how the DIE algorithm yields consistently

better performances than its unmodified version, providing a throughput higher than SP up to timer values 45 times larger than the average connections duration time (20 s).

The final step in our analysis of the DIE algorithm performances is to consider its sensitivity to the α and β parameters. In a scenario with 3.7 network load, update 200 s, polling 25 s, we studied the effects of varying α and β separately. The upper plot in Figure 8 shows that, for $\beta = 0.9$ there is no significant effect in changing α in the $[0.9, 0.99]$ range, indicating that the average filtering factor is not so important in the DIE performances. A similar behavior was observed also for other β values.

Since α seems to be not so relevant, we chose to set it to 0.99 and then to study the effect of the β variation in the $[0.1, 0.95]$ range. As shown in the lower plot of Figure 8, the effect of $beta$ is much more evident, since the lower the β value, the lower the *deception* effect, pushing the performance below the unmodified algorithm. In any case, the performance gain is positive for a wide range of β values, reducing the necessity of a perfectly matched β value.

ACKNOWLEDGMENT

This work was partly supported by ALCATEL under the research contract ULC/14.03, and partly sponsored by the FP6 NoE Euro-NGI.

V. CONCLUSIONS

The paper has presented a novel solution to address the problem of routing with stale link-state information, without increasing the frequency of link state advertisement. The basic idea behind the proposed protocol is that nodes are allowed to deliberately deceive other nodes in the network by advertising load information that may be higher the the actual values. Simulation results show that the performance of the proposed algorithm are very promising even in the case of far-and-between link state updates.

REFERENCES

[1] J.L.Sobrinho, "Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet," *IEEE-ACM Transactions on Networking*, Vol. 10, N. 4, August 2002, pp. 541-550.

[2] J.L.Sobrinho, "Network Routing with Path Vector Protocols: Theory and Applications." *ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

[3] G.Apostolopoulos, R.Guérin, S.Kamat, S.K.Tripathi, "Quality of Service Based Routing: A Performance Perspective," *ACM SIGCOMM 1998*, Vancouver, Canada, September 1998.

[4] A.Shaikh, J.Rexford, K.Shin, "Load-Sensitive Routing of Long-lived IP Flows," *ACM SIGCOMM 1999*, Cambridge, MA, August 1999.

[5] G.Apostolopoulos, D.Williams, S.Kamat, R.Guerin, A.Orda, T.Przygienda, "QoS Routing Mechanisms and OSPF Extensions," *RFC 2676*, IETF, August 1999.

[6] L.Fratta, M.Gerla, L.Kleinrock, "The Flow Deviation Method - An Approach to the Store-and-Forward Communication Network Design," *Networks*, Vol. 3, 1973, pp. 97-133.

[7] Z.Wang, Y.Wang, L.Zhang, "Internet Traffic Engineering Without Full Mesh Overlaying", *IEEE Infocom 2001*, Anchorage, AK, April 2001.

[8] A.Sridharan, R.Guèrin, C.Diot, "Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks", *IEEE Infocom 2003*, San Francisco, CA, March 2003.

[9] Z.Wang, J.Crowcroft, "QoS Routing for Supporting Multimedia Applications," *IEEE JSAC*, Vol.14, N. 7, September 1996, pp 1228-1234.

[10] Q.Ma, P.Steenkiste, H.Zhang, "Routing High-Bandwidth Traffic in Max-Min Fair Share Networks," *ACM SIGCOMM 1996*, Stanford, CA, August 1996.

[11] C.Casetti, R.Lo Cigno, M.Mellia, M.Munafò, Z.Zsoka, "A new class of QoS routing strategies based on network graph reduction", *Computer Networks*, Vol. 41, No. 4, pp. 475–487, February 2003.

[12] A.Basu, A.Lin, S.Ramanathan, "Routing Using Potentials: A Dynamic Traffic-Aware Routing Algorithm." *ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

[13] M.Ajmone Marsan, A.Bianco, C.Casetti, C.F.Chiasserini, A.Francini, R.Lo Cigno, M.Mellia, M.Munafò, "An Integrated Software Environment for the Simulation of ATM Networks", *SCSC'97, Summer Computer Simulation Conference*, Arlington, Virginia, USA, July 1997

[14] C.Casetti, R.Lo Cigno, M.Mellia, M.Munafò, Z.Zsoka, *A Realistic Model to Evaluate Routing Algorithms in the Internet*, Globecom 2001, San Antonio, Texas, USA, November 25-29, 2001.

[15] TNG-Polito *ANCLES home page*, `http://www.tlc-networks.polito.it/ancles/`

[16] C. Casetti, G. Favalessa, M. Mellia, M. Munafò, "An Adaptive Routing Algorithm for Best-effort Traffic in Integrated-Services Networks", *16th International Teletraffic Congress (ITC-16)*, Edinburgh, UK, June 1999