Multi-path OSPF Performance of a Software Router in a Link Failure Scenario

Vincenzo Eramo [1], Marco Listanti [2], Antonio Cianfrani [3]

INFOCOM Department - University of Roma " La Sapienza"
Via Eudossiana 18, 00184 Roma, Italy
[1]vincenzo.eramo@uniroma1.it
[2]marco@infocom.uniroma1.it
[3]cianfrani@net.infocom.uniroma1.it

***Abstract*—In this paper we analyze intra-domain routing protocols improvements to support new features required by real time services. In particular we introduce OSPF Fast Convergence and highlights the advantage of using an incremental algorithm instead of Djikstra one to compute the shortest paths. Then we propose a new multi-path incremental algorithm that we have implemented in OSPF code of Quagga open-source routing software. Analyzing an index characterizing OSPF performance we compare our algorithm with an incremental algorithm not supporting multi-path and demonstrate that, even if multi-path support, the reconfiguration times are really similar; moreover,in some cases, our algorithm performs better, especially in a link failure scenario.**

I. INTRODUCTION

OSPF [1] and IS-IS [2] are the most used intra-AS routing protocols in today Internet: they are link-state protocols and they use the well-know Djikstra algorithm to construct the router routing table. Their performance can be characterized with the convergence time index, which represent the time for a network to reconfigure itself when a topological event happens. The convergence time is influenced by three different phases performed by routing protocols: detection, flooding and Shortest Path First (SPF) computation. The detection phase consists in identification of a topological modification and, if no hardware detection mechanism is provided, depends on Hello messages exchange between routers; the propagation phase consists in exchange of Update messages from the router discovering the modification to all other network routers, through flooding mechanism; the SPF computation is the phase in which the shortest paths to all destinations are discovered using Djikstra algorithm.

In the last years there has been a great interest in real time services, such VoIP and distibuted gaming, which require high network performance in terms of delay and jitter. Actual routing protocols are inefficient in such a scenario [3] because they were designed to support best-effort traffic. In particular convergence time has to be hardly reduced from the actual 40-50 seconds to 100-200 ms; this purpose can be achieved introducing some optimizations in the three phases of the convergence process [4]: the detection time can be reduced introducing the milliseconds Hello mechanism, the flooding time can be reduced making the flooding messages the ones with highest priority and the SPF computation can be reduced using an incremental algorithm instead of the Djikstra one.

In this paper we introduce incremental algorithms for the single-source shortest path problem and evaluate the applicability of past algorithms in a networking scenario. In particular we remark the necessity of an algorithm that reacts to link deletion, which represent the most damaging event in a network, and insertion and that support multi-path: multi-path support allows to calculate all possible shortest paths to all destinations, so routers can perform load balancing for some destinations through paths of minimum equal cost. So we define a multi-path incremental algorithm with these features and evaluate its performance implementing it in the OSPF code of an open-source routing software, Quagga [5]. The performance evaluation is performed measuring the SPF computation time index of an OSPF router [6] through white box measurements, comparing our multi-path incremental algorithm with an uni-path incremental algorithm.

The paper is organized into 4 sections. In section 2 incremental algorithms definition, motivation and previous works are described. Our multi-path incremental algorithms is described with an example in section 3. In section 4 the main numerical results of multi-path and no uni-path algorithms are shown. Finally in section 5 conclusions and further research items are illustrated.

II. INCREMENTAL ALGORITHMS

A. Motivations

An algorithm to evaluate the shortest paths from a router to all AS destinations is the key element of a link-state routing protocol. Actual intra AS routing protocols, such as OSPF and IS-IS, make use of Djikstra algorithm: when a topological change happens all the shortest path are calculated from scratch, not using the old shortest paths information. This kind of algorithm has always as start point a database, in which each network element is described, and as a result a tree, even if it is not a real tree as we discuss later, of paths of minimum cost (SPT: Shortest Path Tree) which allows to construct the router routing table.

The SPT computation, performed by Djikstra algorithm, is the hardest operation for a router, because all CPU is used, but also the most problematic. In fact when a topological change

occurs and SPT computation is executing, the routing table used not reflects the real network topology. In this way data packets can pass through sub-optimal paths or even through paths not more available and so they can be lost, degrading network performance; this last case can happens when there is a link failure, which represents the most dangerous event for a network. In the last years some studies has been done to characterize the evolution of a network topology, analyzing routing protocols messages and SPTs successions. In particular the work presented in [7], where an ISP has been analyzed for a year, demonstrates as 65% of SPT computations produces the same SPT that was used before the topological event and in the other cases the SPT calculated is really similar to the last one. These results highlights that Djikstra algorithm is inefficient in a real scenario where only few operations have to be done to calculate new SPT instead of a full computation. These motivations have lead to the introduction of a new class of algorithms for the computation of the shortest paths, the incremental ones [8]–[11]. They make use of the previous SPT to calculate the new one, computing the only part of SPT influenced by the topological event. In this way incremental algorithms meet real network requirements: computing resources are saved, best paths are available first and so routing convergence can be highly reduced.

B. Previous works

In the past years various incremental algorithms for the single-source shortest path problem in a directed and oriented graph has been presented. All this works has been conceived for a most generic problem than a networking scenario, so we analyze for each one its relevant proprieties.

The first interesting work has been presented by Ramalingam and Reps [8], which propose an algorithm reacting to edge deletion or insertion. The most interesting aspect of this work is the introduction of output complexity model, in which the cost of a dynamic algorithm is evaluated as a function of the number of output updates caused by each input modification; this model, with some variations, has been used by all other works. The weak point of the algorithm is that for each node of the SPT it maintains only distance and outdegree information and not the useful information to calculate the sequence of nodes representing the shortest paths from root to every topology node.

In [9] Frigioni et al. propose a dynamic algorithm reacting to edge deletion, insertion and modifications of weight, which maintains for each node a list of children and a single parent, which means it does not support multi-path equal cost feature; moreover two further structures, backward-level and forward-level lists, are maintained for each node allowing the scanning of a less number of edges when the algorithm is performed, with respect to [8].

In [10], [11] Narvaez et al. propose two different incremental algorithms to be used when a link increases or decreases its weight. The first one [10] is the dynamic version of three static algorithms, Djikstra, Bellman-Ford and D Esopo-Pape, while the second one [11] is based on a Ball-and-String model, using an original search criteria. As for [9], the algorithm maintains a completed SPT structure, with parent and children list attributes for each node, but does not support multi-path aspect.

C. Incremental algorithms in a networking scenario

Incremental algorithms described before can not be directly used in a routing protocol because in a networking scenario they have to satisfy specific requirement.

First at all an incremental algorithm has to be developed to react to the most common events in a network. As demonstrated in [7] edge deletion and insertion are the topological changes that characterize an ISP routing protocol, while edge metric modifications are really rare; a more deep analysis, presented in [12], demonstrate that 70% of unplanned events involves single links. We can conclude that deletion and insertion of a single edge have to be events the algorithms react to. In this way algorithms proposed by Narvaez et al. can not be used as they have been presented.

A really important aspect that a routing protocol algorithm needs to have is multi-path support. This feature consists in calculating all shortest paths from a router to each destination. This means that the algorithm needs to calculate a sub-graph of minimum paths and this is the reason because we said SPT is not a real tree. Multi-path additional information is then computed by IP protocol: when the router receives a packet directed to a specific destination, it can choose among different next-hop routers, in general one for each shortest path. In this way router can perform load balancing, choosing a specific next-hop for each packet, or flow of packets, through some functions [13]. Multi-path support, as well as allowing load balancing, also permits to improve TCP performance: in particular works in [14]–[16] demonstrate that with some TCP modifications to support path diversity, transport-layer performance can be increased. Multi-path is a feature of static Djikstra algorithm while is not supported by Frigioni and Narvaez algorithms.

Another aspect to be considered is that the algorithm has to create a routing table: in this way it has to operate with appropriate data structures. In particular for each node of the SPT some attributes has to be maintained: distance from root, list of parents (because of multi-path support), list of children and list of next-hops. This last element represents the first routers in the paths from root to node and it is just the next-hop routers to be inserted in the routing table: it can be easily calculated from the list of parents attribute of each node. The algorithm of Ramalingam, unlike others, does not maintain such a structure for SPT.

Finally, an incremental algorithm to use in a networking scenario has to react to single edge deletion and insertion, to support multi-path and to create specific data structures. In the next section we introduce our incremental algorithm with the previous aspects and highlights as multi-path support can be used as a strong point to reduce network re-configuration time especially in a link failure scenario.

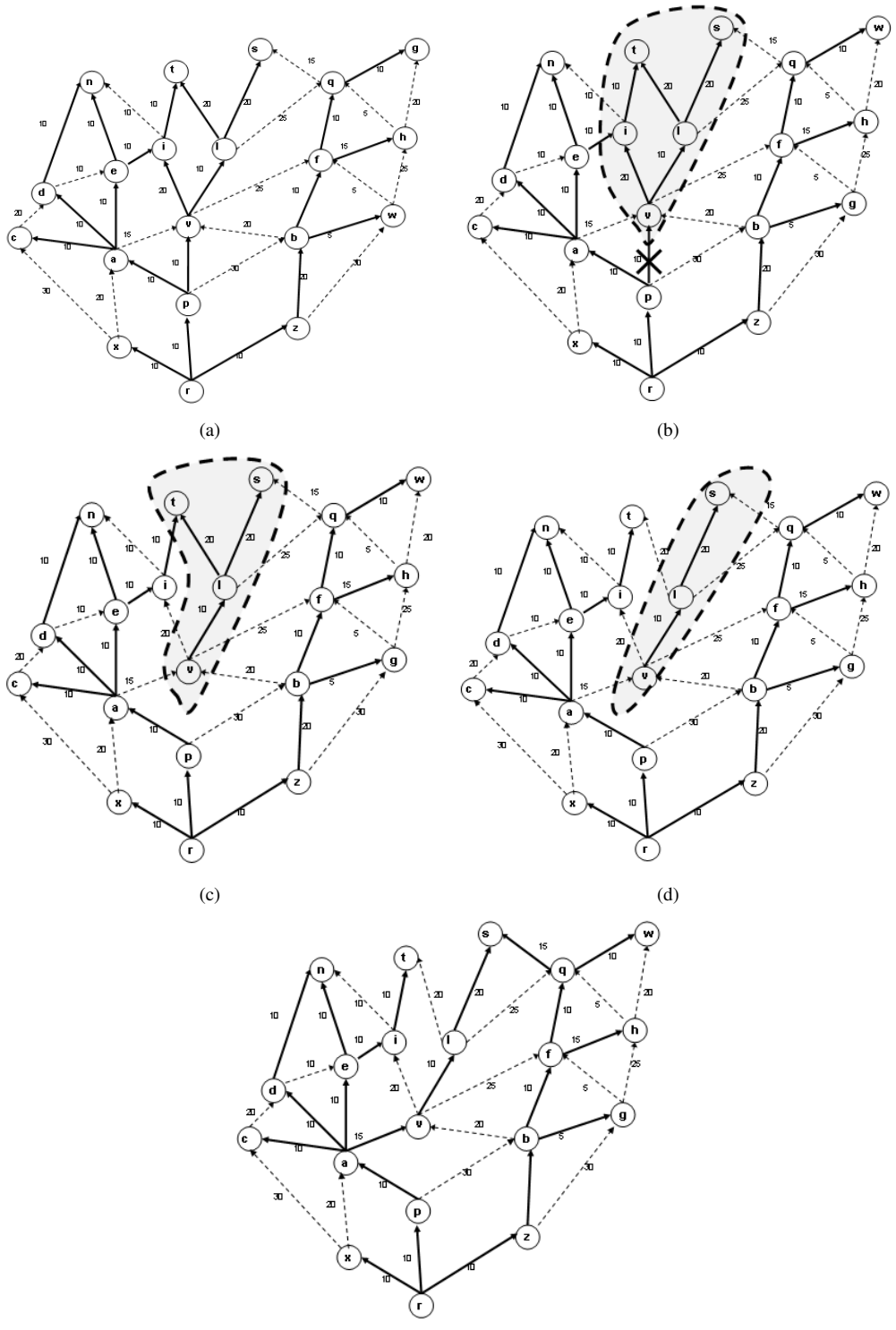

Fig. 1. Example of edge deletion. a) Network graph with only a subset of edges with their cost; the solid thick arrows represents all the SP(G) edges. b) Set of affected nodes after edge deletion. c) and d) SP(G) during the initialization phase of the algorithm; because of the multi-path, shortest paths to i and t exist and the nodes i and t can be deleted from the set D(v). e) Final SP(G) after that the operations of the algorithm have been performed.

III. Our multi-path incremntal algorithm

In this section we introduce our multi-path incremental algorithm and show its behaviour in a link failure scenario. Our algorithm is a dynamic version of Djikstra static algorithm which react to single link deletion and insertion; its relevant characteristic is how it make use of multi-path information to

speed up SPT computation.

The algorithm is composed by an initialization phase, different for link deletion and insertion, and a main phase. Initialization phase in case of link insertion and main phase are modified versions of the same phases presented in [10]: they have been modified to support multi-path and new data structures our algorithm uses. The initialization phase in case of link deletion is really innovative because allows to minimize number of nodes affected by link deletion thanks to multi-path information. A detailed description of our algorithm can be found in [17], so in this section we highlight algorithm behaviour in the case of link deletion through an example.

Before describing the example, we have to introduce some notations regarding a network graph, in particular data structures used to maintain SPT information on each node. A node v has different attributes: $P(v)$ is the set of v parents, $C(v)$ is the set of v children , $D(v)$ is the set of v descendents, $d(v)$ is v distance and $NH(v)$ is the set of v next-hops. The algorithm also maintains a data structure, the Candidate List Q, that contains nodes whose attributes must be updated; an element in Q is the triple (v, P, d_{new}), where v is the node to be updated, P is the new set of parents and d_{new} is the new v distance.

Figure 1(a) shows a network graph, where each link is bidirectional and weighted: for simplicity the two edges of the same bidirectional link have the same cost. The solid thick arrows are all the SPT links while the thin dashed ones do not belong to SPT.

Let us suppose that link from node p to node v fails. In the initialization phase of the incremental algorithm it is first checked if link deleted belongs to SPT, otherwise the algorithm stops, and then all nodes affected by failure are checked: this set of nodes is represented in Figure 1(b) with the dashed curve and contains all v descendents $D(v)$, nodes belonging to sub-tree having v as root.

The algorithm has to evaluate v set of parents to find possible multi-paths: in fact if v has other parents and so other paths of minimum cost, the algorithm only has to remove the deleted path for v and recomputed next-hop attribute for all its descendents. In this case the only parent of v is p so v is unreachable. The search of external multi-path is performed for all v descendents, scanned in an ordered way. For node l there are not external multi-paths while for node i there is an external multi-path, with e as parent: i is deleted from $D(v)$, its set of parents now contains the only node e and its distance remains 40. The SPT at this time is represented in Figure 1(c). Scanning v descendents, the algorithm find an external equal cost path for node t too. It has two parent: l, a v descendent, and i, just removed from set of v descendents. So t is in turn removed from $D(v)$, its distance is not changed and its set of parents contains the only node i. The last v descendents is s and for it there are not external multi-path so it is set to an unreachable state. The unreachable nodes maintain their parent-child relationships ($C(v) = l$, $P(l) = v$, $C(l) = s$, $P(s) = l$). After initialization phase the algorithm produces the SPT represented in Figure 1(d).

The last thing to do in the initialization phase is to find new shortest paths for node affected, through nodes external to $D(v)$. In particular all external incoming edges have to be evaluated. In this case for node v the best path has a as parent and a cost equal to 35, so the element $(v, a, 35)$ is enqueued in Q, for node l it has t as parent and a cost equal to 70, $(l, t, 70)$ is enqueued in Q, and for node s it has q as parent and a cost equal to 65, $(s, q, 65)$ is enqueued in Q. Notice as during the initialization phase the set of affected nodes has to be reduced from five to three elements, using multi-path information.

In the main phase of the algorithm, the first element extracted from Q is v: its new possible distance (35) is obviously better than the present one (infinite) so all its attributes, except set of children, are changed ($d(v) = 35$, $P(v) = a$, $NH(v) = \{p\}$); node v will certainly not be modified during the last part of the algorithm, as explained in the correctness analysis. All v descendents have to be updated, so $d(l) = 45$, $NH(l) = p$, $d(s) = 65$ and $NH(s) = p$. The second element extracted is s: its new possible distance is equal to its distance, updated in the first step, so the algorithm simply stores this new equal cost path adding q to $P(s)$ and z to $NH(s)$. Considering equal cost multi-path the router r can reach node s through two different next-hop routers and so it can balance traffic. The last element extracted from Q is l but its new possible distance is bigger than the present one, so nothing has to be done. The candidate heap is empty so the algorithm stops. Final SPT is represented in Figure 1(e).

IV. Performance Evaluation

A. Test methodologies

As discussed in [17], complexity analysis cannot be a full characterization of our incremental algorithm; so we have decided to evaluate our algorithm behavior in a real environment, implementing it in a routing protocol, OSPF, and measuring protocol performance indexes in different topology scenario. To implement incremental algorithm we have used a routing software with an open code (Open-source routing software), Quagga [5]. Quagga is designed for Unix operating systems (Linux, BSD and Solaris) and it provides TCP/IP based routing protocols, including OSPF, RIP and BGP. The most interesting aspect of an open-source routing software is its flexibility that allows evaluation of new routing feature, such as our algorithm. We have implemented our multi-path algorithm in OSPF code of Quagga software so that every time the deletion or insertion of a single link happens, incremental algorithm, instead of Djikstra one, is performed.

To characterize algorithm performance we have realized white box measures, introducing specific timestamps in the OSPF code; in particular these timestamps allows to calculate exactly the SPT computation time, the time for the router to compute all shortest paths. To evaluate this index we have used the test configuration reported in Figure 2: DUT (Device Under Test) is the PC based router equipped with Quagga and our multi-path algorithm.

The network topology is made up of two real routers (a testing PC and the DUT) and a variable number of fictitious

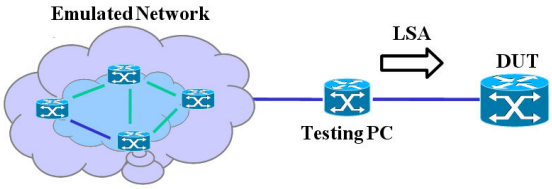

Fig. 2. Test-bed for the evaluation of the SPF computation time in a Device Under Test (DUT).

routers and networks, so that the DUT will have to find the shortest paths to all the vertexes of the emulated network, a vertex being either a network or a router. The testing PC is running a C++ software allowing the generation of network topology and the generation, and sending to the DUT, of Link State Advertisements (LSA) describing the network topology.

B. Numerical results

Our multi-path incremental algorithm has been compared to the uni-path algorithm proposed by Narvaez [10]; to do that we have implemented uni-path algorithm, with some modifications regarding mainly deletion support and data structures used, in OSPF Quagga code. We have used a PC with 2,6 GHz processor and 512 MB RAM. The comparison has been carried out by emulating on the DUT real network topologies measured within the Rocketfuel project [18]. In particular we have considered the topology of Verio, an USA Internet Service Provider whose network is composed by 893 routers and 4154 links. All of the link costs have been set to 10. We have decided to characterize algorithms performance in a link failure scenario because it can cause data lost and so network performance degradation. Moreover the SPF computation time in an incremental algorithm depends on link position and on type of change occurring, so we have chosen to measure this time when the deletion of each single link of the Verios network occurs; after each deletion we have re-inserted the link just deleted and then we have performed the successive link deletion measure. The SPF computation time is reported in Figure 3 and 4 in the case of multi-path and uni-path incremental algorithm respectively, as a function of the link interested; we have decided to order links in x-axis in decreasing order of SPT time in multi-path algorithm. In both the figures we also report the time that the DUT takes to run the static Dijkstras algorithm, which is different in the two cases of uni-path and multi-path: in the first case Djikstra compute a single path of minimum cost for each destination, in the second case all shortest paths to each destination and so we have two different values in the two cases. Obviously Djiksra uni-path and multi-path times are independent of the link position in which failure occurs.

Observing Figure 3 and 4 you can notice that SPT computation time is always less than static algorithm one. Furthermore the average SPF computation time is 0,35 ms and 0,349 ms in multi-path and uni-path algorithm respectively, while the static SPF computation time is 8,116 ms and 7,407 ms in

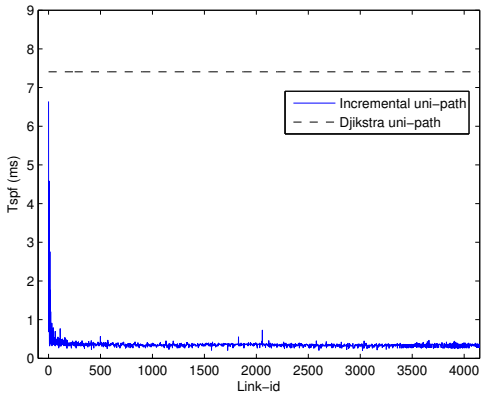

Fig. 3. Performance evaluation of the uni-path incremental algorithm in the case of link failure.

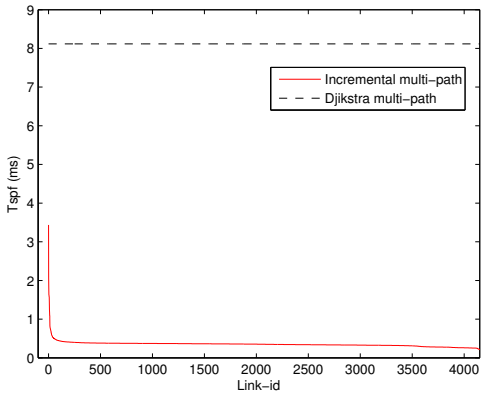

Fig. 4. Performance evaluation of the multi-path incremental algorithm in the case of link failure.

multi-path and uni-path cases; this means that the incremental algorithms allow a saving of about 95% in processing time with respect the case in which the shortest paths would be evaluated by using the Dijkstras algorithm. A really interesting consideration is that SPT computation time trend is really similar for the two algorithms despite multi-path algorithm allows to compute a more complex structure and so more information. In fact in the case of a uni-path algorithm SPT, uni-SPT, is a tree and so it has exactly $(N-1)$ links, where N is number of nodes; in the case of a multi-path algorithm SPT, multi-SPT, is a graph, because all paths of minimum cost have to be considered, and so it can have more than N links. In Verio topology uni-SPT has 892 links while multi-SPT has 1429 links, so in this last case 500 links more belongs to SPT causing a higher number of operations to be performed; uni-SPT and multi-SPT are referred to the situation before each link deletion, because after that SPTs can change their structure.

Let us consider a subset of links to better understand the results: we only consider the first 50 links because they cause the highest SPT times in both algorithms. Results of this subset

TABLE I
MOST RELEVANT LINKS STATISTICS

	Uni-path			Multi-path			
Link-id	Affiliation to SPT	Descendents	SPT time (ms)	Affiliation to SPT	Descendents	Multi-path descendents	SPT time (ms)
1	Yes	448	$6,643$	Yes	571	441	$3,435$
3	Yes	261	$3,648$	Yes	409	357	$1,826$
5	Yes	331	$4,584$	Yes	466	435	$1,622$
7	Yes	325	$4,199$	Yes	369	328	$1,576$
16	Yes	91	$1,446$	$Yes\ multi-path$	91	/	$0,774$
21	Yes	83	$1,204$	$Yes\ multi-path$	83	/	$0,719$
2	Yes	55	$1,397$	Yes	187	146	$2,072$
4	Yes	28	$0,684$	Yes	202	188	$1,628$
6	Yes	35	$1,004$	Yes	103	32	$1,581$
8	Yes	26	$0,749$	Yes	198	189	$1,431$
9	No	/	$0,366$	$Yes\ multi-path$	219	/	$1,238$
10	No	/	$0,365$	$Yes\ multi-path$	207	/	$1,185$

of links are presented in Figure 5. Moreover we have report in Table I the most significant links with some informations. In the case of uni-path algorithm we report for each link, identified by its id, information about its affiliation to uni-SPT, that can be Yes or No, and number of descendents in uni-SPT; in the case of multi-path algorithm we report for each link information about its affiliation to multi-SPT, that can be Yes, No or Yes multi-path if link end-node has other paths of minimum cost, number of descendents in multi-SPT and number of descendents with other paths of minimum cost.

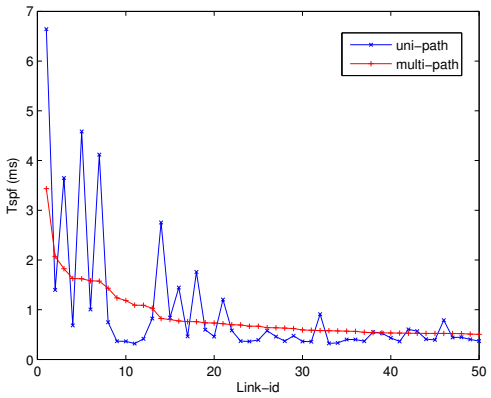

Fig. 5. Performance comparision of the uni-path and multi-path incremental algorithm in the case of link failure.

Analyzing results we can immediately see that multi-path algorithm results for the first six points of Table I are really better than uni-path algorithm ones. These points reflect a common situation: in both multi-SPT and uni-SPT the deleted link has many descendents but multi-path algorithm allows a quicker SPT computation, two or three time faster than uni-path algorithm, because it exploits multi-path information. For example the first point regards a link with 448 descendents in uni-SPT and 571 descendents in multi-SPT, but in this last case 441 descendents have at least another path of minimum cost so the multi-path algorithm makes use of these information to stabilize these descendents in the initialization phase without inserting them into Candidate List structure. This result is the most important for our multi-path algorithm; in fact in a link failure scenario, which is the most dangerous for a network because it can causes packet loss, when a lot of nodes are involved, our algorithm allows a quick reconfiguration with respect to an uni-path algorithm.

Another situation in which our algorithm performs better is when the end node of deleted link has itself other paths of minimum cost: this happens for links 16 and 21 in which the number of descendents is 91 and 83 respectively and it is the same in uni-SPT and multi-SPT, but multi-path algorithm has better reconfiguration times, an half of uni-path algorithm ones.

Obviously there are also situations in which uni-path algorithm performs better. Links 2, 4, 6, and 8 have a lot of descendents in multi-SPT and a few in uni-SPT so nodes and links involved in SPT computation during uni-path algorithm are much less and reconfiguration times are better; for example link 4 has 28 descendents in uni-SPT while it has 202 descendents in multi-SPT. A most propitious situation for uni-path algorithm is the ones of links 9 and 10: in this cases the deleted link does not belong to uni-SPT while it belongs to multi-SPT and it has also a lot of descendents. For example link 9 does not belong to uni-SPT but it has 219 descendents in multi-SPT; the difference in terms of SPF computation time is limited because the end node of the deleted link has other paths of minimum cost and so multi-path algorithm performs a limited number of operations.

V. CONCLUSIONS

The aim of our work was to introduce the advantage of using an incremntal algorithm in a networking scenario and to propose a new incremental shortest path algorithm with multi-path support. We have implemanted our algorithm in Quagga open-source routing software and realized a test-bed to calculate SPF computation time. Algorithm performance has been compared with ones of a uni-path incremntal algorithm. We have demonstrated that an incremntal algorithm allows to higly reduce SPT computation time with respect to a static algorithm and that our multi-path algorithm has the same performance of a uni-path algorithm. Moreover we have seen that multi-path algorithm performs better than uni-path one

when link deletion affects an high number of topology nodes, because it can make use of multi-path information to speed up SPT computation. In future topics, different network scenarios will be evaluated and the proposed algorithm will be modified to support multiple changes.

REFERENCES

[1] J. Moy. OSPF Version 2 , Request for Comments 2328, April 1998.

[2] R. Callon. "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, December 1990.

[3] C. Boutremans, G. Iannaccone and C. Diot. Impact of link failures on VoIP performance, in Proceedings of ACM NOSSDAV, May 2002.

[4] C. Alaettinoglu, V. Jacobson, H. Yu, Towards Milli-Second IGP Convergence, draft-alaettinoglu-ISIS-convergence-00, November 2000.

[5] Quagga Project [Online]. Available http://www.quagga.net/.

[6] V. Eramo, M. Listanti, A. Cianfrani, OSPF Performance and Optimization of Open Source Routing Software, International Journal of Computer Science and Applications, Vol. IV Issue 1, 2007 .

[7] D. Watson, F. Jahanian, C. Labovitz. Experiences With Monitoring OSPF on a Regional Service Provider Network, In Proceedings of the 23rd International Conference on Distributed Computing Systems, page 204, IEEE Computer Society, 2003.

[8] G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems, Theoretical Computer Science, vol. 158, pp. 233277, 1996.

[9] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees, Journal of Algorithms, vol. 34, pp 251-281, February 2000.

[10] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng. New dynamic algorithms for shortest path tree computation, IEEE Transaction on Networking, vol. 8, pp. 734-746, 2000.

[11] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng. New dynamic SPT algorithm based on a Ball-and-String model, IEEE Transaction on Networking, vol. 9, pp. 706-718, 2001.

[12] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.N. Chuah, and C. Diot. Characterization of failures in an IP backbone, in IEEE Infocom2004.

[13] D. Thaler, C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection, RFC 2991, November 2000.

[14] H.Han, S. Shakkottai, C.V. Hollot, R. Srikant, D. Towsley. Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet, IEEE/ACM Transactions on Networking , vol. 14, Issue 6, pp. 1260-1271, December 2006.

[15] Y. Lee, I. Park, and Y. Choi, Improving TCP performance in multipath packet forwarding networks, Journal of Communications and Networks, vol. 4, no. 2, pp. 148-157, June 2002.

[16] P. Key, L. Massoulie, D. Towsley. Combining Multipath Routing and Congestion Control for Robustness, 40th Annual Conference on Information Sciences and Systems, pp. 345-350, 2006.

[17] V. Eramo, M. Listanti, A. Cianfrani. Implementation and performance evaluation of a multi-path incremental shortest path algorithm in Quagga Routing Software, accepted at DRCN 2007, La Rochelle, October 2007.

[18] Rocketfuel Project [Online]. Available http://www.cs.washington.edu/research/networking/rocketfuel/.