Efficient algorithms for distributed shortest paths on power-law networks

Gianlorenzo D'Angelo[1], Mattia D'Emidio[2], Daniele Frigioni[2], and Daniele Romano[2]

[1] MASCOTTE Project INRIA/I3S(CNRS/UNSA) gianlorenzo.d_angelo@inria.fr
[2] Dip. di Ingegneria e Scienze dell'Informazione e Matematica, University of L'Aquila
{mattia.demidio,daniele.frigioni}@univaq.it, daniele.romano.vis@gmail.com

1 Introduction

The problem of computing and maintaining shortest paths in a distributed network whose topology dynamically changes is a core functionality of today's communication networks. The problem has been widely studied in the literature, and the solutions found can be classified as *distance-vector* and *link-state*.

Distance-vector algorithms, as for example the distributed Bellman-Ford method [6], require that a node knows the distance from each of its neighbors to every destination and stores them in a data structure called *routing table*; a node uses its own routing table to compute the distance and the via to each destination. The main drawbacks of distance-vector algorithms are the massive use of communication resources and the well-known *looping* phenomena.

Link-state algorithms, as e.g. the OSPF protocol used in Internet [7], require that a node knows the entire network topology to compute its distance to any destination, usually running Dijkstra's algorithm. They are free of looping, although each node needs to receive and store up-to-date information about the entire network topology after a change, thus requiring quadratic space per node.

In the last years, there has been a renewed interest in devising new efficient distance-vector solutions for Large-Scale Ethernet networks, where usually scalability and reliability are highly desirable properties or where the memory power of the nodes is limited (see, e.g., [2, 8, 9]). Notwithstanding this increasing interest, the most interesting distance-vector algorithm is still DUAL (Diffuse Update ALgorithm) [4], which is free of looping and is part of the CISCO's widely used EIGRP protocol, although it requires a high space per node.

In this paper, we present two contributions in this field. First, we describe LFR (*Loop Free Routing*) a new loop-free distance vector routing algorithm, recently proposed in [3], which: (i) from the theoretical point of view, is able to update the shortest paths of a distributed network in fully dynamic scenarios using the same message complexity and less space per node than DUAL; (ii) from the experimental point of view, has been shown to be the best choice, both in terms of messages sent and space requirements, in networks having a power-law node degree distribution, a highly important class of networks which includes

many currently implemented communication infrastructures, like the Internet, the WWW, and so on [1]. Second, we introduce DP (*Distributed Pruning*), a new general and practical technique which is a generalization of DLP [2]. DP can be combined with every distance-vector algorithm based on shortest paths with the aim of overcoming some of their limitations (high number of messages sent, low scalability, poor convergence) in power-law networks. We give experimental evidence of the effectiveness of DP, showing that the combination of DP with DUAL and LFR provides a huge improvement in both the global number of messages sent and the space occupancy per node wrt DUAL and LFR, resp..

2 Description of LFR

In this section, we describe LFR, introduced in [3]. LFR stores, for each node v, the estimated distance $\texttt{D}[v,s]$ and the *feasible via*, $\texttt{VIA}[v,s]$, that is the node which provides the minimum distance to s and satisfies SNC (*Source Node Condition*), a sufficient condition for loop-freedom [4]. In addition, node v maintains for each $s \in V$, the following data structures: (i) $\texttt{ACTIVE}[v,s]$: the state of node v wrt a source s; v is in *active* state ($\texttt{ACTIVE}[v,s] = true$), if and only if v is trying to update $\texttt{VIA}[v,s]$ after a weight increase operation; (ii) $\texttt{UD}[v,s]$: the distance from v to s through $\texttt{VIA}[v,s]$; if v is active then $\texttt{UD}[v,s] \geq \texttt{D}[v,s]$, otherwise they coincide. In addition, in order to implement SNC, node v stores a temporary data structure $\texttt{tempD}_v$ which is allocated only when v is active wrt s, and it is deallocated when v turns passive wrt s. The entry $\texttt{tempD}_v[u,s]$ contains $\texttt{UD}[u,s]$, for each $u \in N(v)$.

The algorithm starts when the weight of an edge $\{x_i, y_i\}$ changes. As a consequence, x_i (y_i, resp.) sends to y_i (x_i, resp.) an *update* message carrying the value $\texttt{UD}[x_i,s]$ ($\texttt{UD}[y_i,s]$, resp.). If an arbitrary node v receives an *update* message from $u \in N(v)$, then it performs an update procedure in which, basically, v compares the received value $\texttt{UD}[u,s] + w(u,v)$ with $\texttt{D}[v,s]$ in order to determine whether v needs to update its estimated distance and $\texttt{VIA}[v,s]$. If node v is active, the processing of the message is postponed by enqueueing it into the FIFO queue associated to s. Otherwise, if $\texttt{D}[v,s] > \texttt{UD}[u,s] + w(u,v)$, then v sets $\texttt{D}[v,s] = \texttt{UD}[u,s] + w(u,v)$ and $\texttt{VIA}[v,s] = u$, while if $\texttt{D}[v,s] < \texttt{UD}[u,s] + w(u,v)$ and $\texttt{VIA}[v,s] = u$, node v performs a phase called Local-Computation in which it sends a *get.dist* to all its neighbor, except $\texttt{VIA}[v,s] = u$, in order to know the corresponding estimated distances to s. Each neighbor $u \in N(v)$ replies with its $\texttt{UD}[u,s]$. When v receives these values, it tries to compute a new $\texttt{VIA}[v,s]$, by comparing the received distances with $\texttt{D}[v,s]$. If this phase succeeds, node v updates its routing information and propagates the change, Otherwise, node v initiates a distributed phase, named Global-Computation. It sets $\texttt{UD}[v,s] = \texttt{UD}[\texttt{VIA}[v,s],s] + w(v,\texttt{VIA}[v,s])$ and sends to all its neighbors, except $\texttt{VIA}[v,s]$, a *get.feasible.dist* message, carrying $\texttt{UD}[v,s]$. A node $k \in N(v)$ that receives such a message first verifies whether $\texttt{VIA}[k,s] = v$ or not. In the first case, it replies to v with $\texttt{UD}[k,s]$. In the second case, it performs the Local-Computation and possibly the Global-Computation, in order to update its routing informa-

tion and to reply to v. Note that this distributed phase can involve the whole network. Finally, if either $\texttt{D}[v,s] = \texttt{UD}[u,s] + w(u,v)$ or $\texttt{D}[v,s] < \texttt{UD}[u,s] + w(u,v)$ and $\texttt{VIA}[v,s] \neq u$, the message is discarded and the procedure ends.

If we denote as Φ the total number of nodes *affected* by a set of updates on the edges of the network, as ϕ the maximum number of destinations for which a node is affected, and as Δ the maximum node degree of the network, then LFR requires $O(\Delta \cdot \Phi)$ messages and $O(n + \Delta \cdot \phi)$ space per node, while DUAL requires $O(\Delta \cdot \Phi)$ messages and $\Theta(\Delta \cdot n)$ space per node.

3 Distributed Pruning

In this section, we introduce DP (Distributed Pruning), a new technique that can be combined with every distance-vector algorithm, with the aim of reducing the messages sent and the space occupancy per node of that algorithm on power-law networks. The idea underlying DP mainly rely on two facts: (i) a power-law network with n nodes typically has average node degree much smaller than n and a number of nodes with low degree which is generally high (for example, the graphs of the *CAIDA IPv4 topology dataset* [5] have average degree approximately equal to $n/2000$, and a number of nodes with degree smaller than 3 approximately equal to $2n/3$); (ii) there are many topological situations in which nodes with degree smaller than 3 should neither perform nor be involved in the distributed computation of shortest paths, as they cannot provide any useful information. DP has been designed to improve the performances of a generic distance-vector algorithm, by exploiting these configurations.

In particular, when applied to a generic distance-vector algorithm, DP forces the distributed computation to be carried out only by those nodes of the network which has degree greater than two (*central* nodes). The non-*central* nodes receive updated routing information passively and do not start any kind of distributed computation. To implement this strategy, DP requires that a generic node stores and updates information about non-*central* paths. To this aim, each node v maintains a data structure, called *CHain Path*, which is an array containing one entry $\texttt{CHP}_v[s]$, for each *central* node s, where the list of all edges, with the corresponding weight, belonging to the non-*central* paths containing s are stored. The space overhead induced by the *CHain Path* is clearly $O(n)$ per node. However, DP globally induces a reduction in the space occupancy per node, as the overhead required to store the *CHain Path* is counterbalanced by a reduction in the space occupancy per node of the original algorithm, which can avoid to store some information about non-*central* nodes.

In order to check the effectiveness of DP, we combined it with DUAL and LFR by obtaining two new algorithms named DUAL-DP and LFR-DP, resp.. Then, we implemented the four algorithms in OMNeT++[3] and performed a preliminary experimental study. As input to the algorithms, we considered instances similar to the ones used in [2]. We generated a set of different tests, each test consists of a CAIDA graph and a set of k edge updates, where $k \in \{5, 10, \ldots, 200\}$

[3] OMNeT++, Discrete event simulation environment: `http://www.omnetpp.org`.

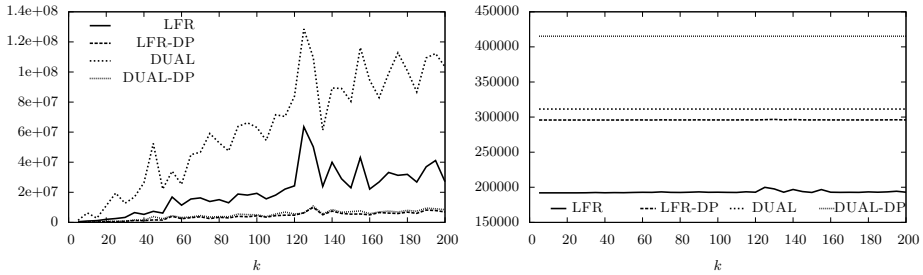

Fig. 1: Number of messages sent (left) and average space occupancy per node in Bytes (right) of LFR, LFR-DP, DUAL and DUAL-DP on a CAIDA graph with 8000 nodes subject to a set of k edge updates.

and each edge update consists of multiplying the weight of a randomly selected edge by a percentage value randomly chosen in $[50\%, 150\%]$.

Our experiments show that the combinations of DUAL and LFR with DP provide a huge improvement both in the global number of sent messages (Fig. 1(left)) and in the space occupancy per node wrt DUAL and LFR, resp.. The reduction in the space occupancy per node is significant both in the average and in the maximum case. We have noticed improvements also wrt the combinations of DUAL and LFR with DLP ([2]), thus allowing us to state that DP represents a step forward wrt the results presented in [2].

References

1. R. Albert and A.-L. Barabási. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
2. G. D'Angelo, M. D'Emidio, D. Frigioni, and V. Maurizio. A speed-up technique for distributed shortest paths computation. In *ICCSA 2011*, volume 6783 of *LNCS*, pages 578–593, 2011.
3. G. D'Angelo, M. D'Emidio, D. Frigioni, and V. Maurizio. Engineering a new loop-free shortest paths routing algorithm. In *SEA 2012*, volume 7276 of *LNCS*, pages 123–134, 2012.
4. J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Trans. on Networking*, 1(1):130–141, 1993.
5. Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, and K. Claffy. The CAIDA IPv4 routed/24 topology dataset. `http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml`.
6. J. McQuillan. Adaptive routing algorithms for distributed computer networks. Technical Report BBN Report 2831, Cambridge, MA, 1974.
7. J. T. Moy. *OSPF: Anatomy of an Internet routing protocol*. Addison-Wesley, 1998.
8. S. Ray, R. Guérin, K.-W. Kwong, and R. Sofia. Always acyclic distributed path computation. *IEEE/ACM Trans. on Networking*, 18(1):307–319, 2010.
9. C. Zhao, Y. Liu, and K. Liu. A more efficient diffusing update algorithm for loop-free routing. In *5th Int. Conf. on Wireless Communications, Networking and Mobile Computing (WiCom09)*, pages 1–4. IEEE Press, 2009.