An experimental study of distributed algorithms for shortest paths on real networks*

Gianlorenzo D'Angelo, Daniele Frigioni, and Vinicio Maurizio

Dipartimento di Ingegneria Elettrica e dell'Informazione, Università dell'Aquila, I-67040 Via Gronchi 18, L'Aquila – Italy. email{gianlorenzo.dangelo, daniele.frigioni}@univaq.it, vinicio.maurizio@cc.univaq.it

1 Introduction

In this paper, we study the problem of dynamically update *all-pairs shortest paths* in a distributed network while edge update operations occur to the network. This problem is considered crucial in today's practical applications. The algorithms for computing shortest-paths used in computer networks are classified as *distance-vector*, as for example the classical Bellman-Ford method [8], and *link-state*, as for example the *OSPF* protocol widely used in the Internet (e.g., see [9]). The main drawbacks of distance-vector algorithms, when used in a dynamic environment, are the well-known *looping* and *count-to-infinity* phenomena (e.g., see [2]) that lead to a very slow convergence. A loop is a path induced by the routing table entries, such that the path visits the same node more than once before reaching the intended destination. A node "counts to infinity" when it increments its distance to a destination until it reaches a maximum distance value.

A number of solutions have been proposed in the literature to update distributed shortest paths [4–6, 10]. Most of them are distance-vector algorithms that rely on the classical Bellman-Ford method which has been shown to converge to the correct distances if the edge weights stabilize and all cycles have positive lengths [2]. However, the convergence can be very slow due to the looping and count-to-infinity phenomena. Furthermore, several known algorithms are not able to concurrently update shortest paths as those in [5, 10], that is, they work under the assumption that before dealing with an edge operation, the algorithm for the previous operation has to be terminated. This is a limitation in real networks, where an edge change can occur while another change is under processing. There are also algorithms which are able to concurrently update shortest paths as those in [3, 6], but they present one or more of the following drawbacks: they suffer of the looping and count-to-infinity phenomena; they are not able to work in the realistic case where an arbitrary sequence of edge change operations can occur to the network in an unpredictable way.

In [4] an experimental study has been performed in the OMNeT++ simulation environment [1] to check the performances of a new algorithm, proposed in the same paper, against the classical Bellman-Ford method. In this paper, we extend the above experimental study by implementing DUAL [5] (a part of CISCO's widely used EIGRP protocol), which is perhaps the best known algorithm, and experimenting it in the same environment. We performed several tests on real-world data [7] and randomly generated update sequences. These experiments show that algorithm in [4] outperforms both Bellman-Ford and DUAL in terms of both number of messages and space occupancy per node.

* Support for the IPv4 Routed/24 Topology Dataset is provided by National Science Foundation, US Department of Homeland Security, WIDE Project, Cisco Systems, and CAIDA.

2 Implemented algorithms

In this Section we briefly describe the three algorithms we have considered for our experimental study: the classical Bellman-Ford method denoted as BF; the algorithm proposed in [5] denoted as DUAL; the algorithm proposed in [4] denoted as ConFu.

Description of BF. In BF, a node v updates its estimated distance to a node s, by simply executing the iteration $\mathtt{D}[v,s] := \min_{u \in N(v)}\{w(v,u) + \mathtt{D}[u,s]\}$, using the last estimated distance $\mathtt{D}[u,s]$ received from a neighbor $u \in N(v)$ and the latest status of its links. Eventually, node v transmits its new estimated distance to nodes in $N(v)$. BF requires to store the last estimated distance vector $\{\mathtt{D}[u,s] \mid s \in V\}$ received from each neighbor $u \in N(v)$.

Description of DUAL. In DUAL, each node v maintains, for each destination s, a set of neighbors called the feasible successor set $F[v,s]$. $F[v,s]$ is computed using a feasibility condition involving feasible distances from each node in $N(v)$ to s, hence node v needs to store the distance from u to s, for each $u \in N(v)$ and each destination s. If the neighbor u, through which the distance to s is minimum, is in $F[v,s]$, then u is chosen as *successor* to s. If $F[v,s]$ does not include u, then v initiates a synchronous update procedure, known as a *diffusing computation*. v sends queries to all its neighbors with its distance through the current successor. From this point onwards v does not change its successor to s until the diffusing computation terminates. When a neighbor $u \in N(v)$ receives a queries, it updates $F[u,s]$. If u has a successor to s after such update, it replies to the query by sending its own distance to s. Otherwise, u continues the diffuse computation: it sends out queries and waits for the replies from its neighbors before replying to v's original query. If there are concurrent updates, the node uses a finite state machine to process these multiple updates sequentially.

Description of ConFu. ConFu assumes that each node of G knows the identity of every other node of G, the identity of all its neighbors and the weights of the edges incident to it. Each node v maintains its own routing table that has one entry for each $s \in V$, which consists of two fields: (i) $\mathtt{D}[v,s]$, the estimated distance between nodes v and s in G; (ii) $\mathtt{VIA}[v,s] \equiv \{v_i \in N(v) \mid \mathtt{D}[v,s] = w(v,v_i) + \mathtt{D}[v_i,s]\}$, the estimated via from v to s. Given a destination s the set $\mathtt{VIA}[v,s]$ contains at most $deg(v)$ elements. Algorithm ConFu consists of three procedures denoted as Decrease, Increase and Dist and it is described wrt a source $s \in V$. The algorithm starts every time an operation c_i on edge (x_i, y_i) is performed. Operation c_i is detected only by nodes x_i and y_i. If c_i is a *weight increase* (*weight decrease*) operation, x_i sends the message *increase*(x_i, s) (*decrease*$(x_i, s, \mathtt{D}[x_i,s])$) to y_i and y_i sends the message *increase*(y_i, s) (*decrease*$(y_i, s, \mathtt{D}[y_i,s])$) to x_i, for each $s \in V$. If a node v receives *decrease*$(u, s, \mathtt{D}[u,s])$, then it performs procedure Decrease, that relaxes edge (u,v). In particular, if $w(v,u) + \mathtt{D}[u,s] < \mathtt{D}[v,s]$, then v updates $\mathtt{D}[v,s]$ and $\mathtt{VIA}[v,s]$, and propagates the updated values to nodes in $N(v)$. If $w(v,u)+\mathtt{D}[u,s] = \mathtt{D}[v,s]$, then u is a new estimated via for v wrt s, and hence v adds u to $\mathtt{VIA}[v,s]$. If a node v receives *increase*(u,s), then it performs procedure Increase which checks whether the message comes from a node in $\mathtt{VIA}[v,s]$. In the affirmative case, v needs to remove u from $\mathtt{VIA}[v,s]$. To this aim, v reduces its $\mathtt{VIA}$. As a consequence, $\mathtt{VIA}[v,s]$ may become empty. In this case, v computes the new estimated distance and via of v to s. To do this, v asks to each node $v_i \in N(v)$ for its current estimated distance, by sending message *get-dist*(v,s) to v_i. When v_i receives *get-dist*(v,s) by v, it performs procedure Dist which sends $\mathtt{D}[v_i, s]$ to v, unless one of the following two conditions holds: 1) $\mathtt{VIA}[v_i, s] \equiv \{v\}$; 2) v_i is updating its routing table wrt destination s. If this is true, then v_i sends ∞ to v. When v receives the answers to the *get-dist* messages by all its neighbors, it computes the new estimated distance and via to s. Now, if the estimated distance has been increased, v sends an *increase* message to its neighbors. In any case, v sends to its neighbors *decrease*, to communicate them $\mathtt{D}[v, s]$. In fact, at some point, v could have sent ∞ to a neighbor v_j. Then, v_j receives the message sent by v, and it performs procedure Decrease to check whether $\mathtt{D}[v, s]$ can determine an improvement to the value of $\mathtt{D}[v_j, s]$.

3 Experimental analysis

Experimental environment. The experiments have been carried out on a workstation equipped with a 2,66 GHz processor and 8Gb RAM. The experiments consist of simulations within the OMNeT++ environment, version 4.0p1 [1]. OMNeT++ is an object-oriented modular discrete event network simulator, useful to model protocols, telecommunication networks, multiprocessors and other distributed systems. In our model, we defined a basic module *node* to represent a node in the network. A node v has a communication *gate* for each node in $N(v)$. Each node can send messages to a destination node through a *channel* which is a module that connects gates of different nodes. A channel connects exactly two gates and represents an edge between two nodes. We associate two parameters per channel: a *weight* and a *delay*. The former represents the cost of the edge in the graph, and the latter simulates a finite but not null transmission time.

Executed tests. For our experiments we used real-world data consisting of CAIDA IPv4 topology dataset [7]. CAIDA (Cooperative Association for Internet Data Analysis) is an association which provides data and tools for the analysis of the Internet infrastructure. The CAIDA dataset is collected by a globally distributed set of monitors which collect data by sending probe messages to randomly selected IP addresses. For each destination selected, the path from the source monitor to the destination is collected, in particular, data collected for each path includes the set of IP addresses of the hops which form the path and the Round Trip Times (RTT) of both intermediate hops and the destination.

We parsed the files provided by CAIDA to obtain a weighted undirected graph G_{IP} where a node represents an IP address contained in the dataset, edges represent links among hops and weights are given by RTTs. Graph G_{IP} consists of $n \approx 35000$ nodes, hence we cannot use it for the experiments, as the amount of memory required to store the routing tables of all the nodes is $O(n^2 \cdot maxdeg)$, where $maxdeg$ is the maximum degree of a node in the graph. Hence, we performed our tests on connected subgraphs of G_{IP} induced by the settled nodes of a breadth first search starting from a node taken at random. We generated a set of different tests, each test consists of a dynamic graph characterized by: a subgraph of G_{IP} of 5000 nodes, a set of $k \in \{5, 10, \ldots, 100\}$ concurrent edge updates. An edge update consists of multiplying the weight of a random selected edge by a value randomly chosen in $[1/2, 3/2]$. For each test, we performed 5 different experiments and we reported average values.

Analysis. BF is always outperformed by both ConFu and DUAL. In fact, it sends a number of messages that is a factor between 32 and 295 (24 and 166, resp.) higher than the number of messages sent by ConFu (DUAL, resp.). Moreover, in the tests for $k \in \{25, 30, \ldots, 100\}$ BF always falls in looping, while ConFu and DUAL always converge to the correct routing tables.

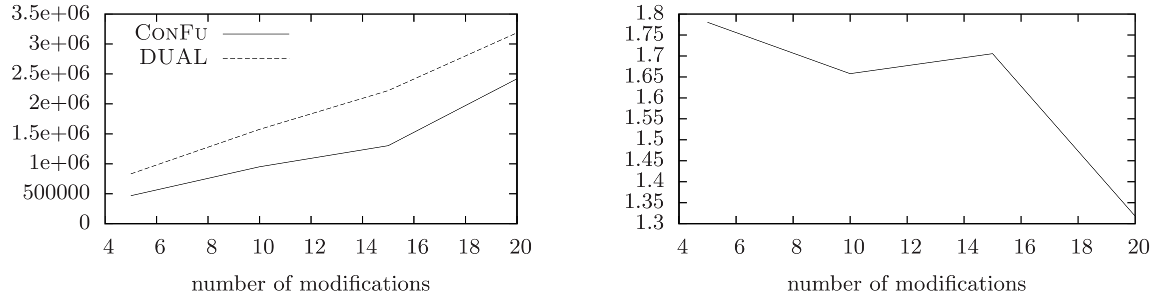

Fig. 1. Left: Number of messages sent by ConFu and DUAL on subgraphs of G_{IP}. Right: Ratio between the number of messages sent by DUAL and ConFu on subgraphs of G_{IP}

In Fig. 1 (left) we report the number of messages sent by ConFu and DUAL on subgraphs of G_{IP} having 5000 nodes and an average value of 6109 edges in the cases where the number k of modifications is in $\{5, 10, 15, 20\}$. The figure shows that ConFu always sends less messages than DUAL. The tests for $k \in \{25, 30, \ldots, 100\}$ are not reported as the inferred results do not change. Fig. 1 (right) shows the results of Fig. 1 (left) from a different point of view, that is, it shows the ratio between the number of messages sent by DUAL and ConFu. It is worth noting that the ratio is within 1.32 and 1.78 which means that DUAL sends a number of messages which is between 32% and 78% higher than the number of messages sent by ConFu.

To conclude, we experimentally analyze the space occupancy per node. DUAL requires a node v to store, for each destination, the estimated distance given by each of its neighbors, while ConFu only needs the estimated distance of v and the set `VIA`, for each destination. Since in these sparse graphs it is not common to have more than one via to a destination, the memory requirement of ConFu is much smaller than that of DUAL. In particular, ConFu requires in average 40000 bytes per node and 40088 bytes per node in the worst case. DUAL requires in average 186090 bytes per node and $5.2M$ bytes per node in the worst case. This implies that DUAL requires in average 4.65 times the space required by ConFu and 130 times the space required by ConFu in the worst case.

References

1. Omnet++: the discrete event simulation environment. `http://www.omnetpp.org/`.
2. D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall International, 1992.
3. S. Cicerone, G. D'Angelo, G. Di Stefano, and D. Frigioni. Partially dynamic efficient algorithms for distributed shortest paths. *Theoretical Comp. Science*, 411:1013–1037, 2010.
4. S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and V. Maurizio. A new fully dynamic algorithm for distributed shortest paths and its experimental evaluation. In *Proc. Int. Symp. on Experimental Algorithms*, volume 6049 of *LNCS*, pages 59–70, 2010.
5. J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking*, 1(1):130–141, 1993.
6. P. A. Humblet. Another adaptive distributed shortest path algorithm. *IEEE Transactions on Communications*, 39(6):995–1002, Apr. 1991.
7. 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`.
8. J. McQuillan. Adaptive routing algorithms for distributed computer networks. Technical Report BBN Report 2831, Cambridge, MA, 1974.
9. J. T. Moy. *OSPF - Anatomy of an Internet routing protocol*. Addison-Wesley, 1998.
10. K. V. S. Ramarao and S. Venkatesan. On finding and updating shortest paths distributively. *Journal of Algorithms*, 13:235–257, 1992.