A Speed-Up Technique for Distributed Shortest Paths Computation

Gianlorenzo D'Angelo, Mattia D'Emidio, Daniele Frigioni, Vinicio Maurizio

A speed-up technique for distributed shortest paths computation

Gianlorenzo D'Angelo, Mattia D'Emidio,
Daniele Frigioni, and Vinicio Maurizio

Dipartimento di Ingegneria Elettrica e dell'Informazione,
Università dell'Aquila, Via G. Gronchi 18, I-67100 L'Aquila, Italy.
{gianlorenzo.dangelo, daniele.frigioni, mattia.demidio}@univaq.it,
vinicio.maurizio@cc.univaq.it

Abstract. We propose a simple and practical speed-up technique, which can be combined with every distance vector routing algorithm based on shortest paths, allowing to reduce the total number of messages sent by that algorithm. We combine the new technique with two algorithms known in the literature: DUAL, which is part of CISCO's widely used EIGRP protocol, and the recent DUST, which has been shown to be very effective on networks with power law node degree distribution. We give experimental evidence that these combinations lead to an important gain in terms of the number of messages sent by DUAL and DUST at the price of a little increase in terms of space occupancy per node.

1 Introduction

The problem of updating efficiently *all-pairs shortest paths* in a distributed network whose topology dynamically changes over the time is considered crucial in today's communication networks. This problem has been widely studied in the literature, and the solutions found can be classified as *distance-vector* and *link-state* algorithms. In a distance-vector algorithm, a node knows the distance from each of its neighbors to every destination and stores this information in a data structure usually called routing table; a node uses its own routing table to compute the distance and the next node in the shortest path to each destination. Most of the distance-vector solutions for the distributed shortest paths problem proposed in the literature (e.g., see [1–3]) rely on the classical Distributed Bellman-Ford method (DBF from now on), originally introduced in the Arpanet [4], which is still used in real networks and implemented in the RIP protocol. DBF has been shown to converge to the correct distances if the link weights stabilize and all cycles have positive lengths [5]. However, the convergence can be very slow (possibly infinite) due to the well-known *looping* and *count-to-infinity* phenomena. In a link-state algorithm, as for example the *OSPF* protocol widely used in the Internet (e.g., see [6]), a node must know the entire network topology to compute its distance to any network destination (usually running the centralized Dijkstra's algorithm for shortest paths). Link-state algorithms are free of the looping and count-to-infinity problems. However, each

node needs to receive up-to-date information on the entire network topology after a network change. This is achieved by broadcasting each change of the network topology to all nodes [6] and by using a centralized algorithm for shortest paths. In the last years, there has been a renewed interest in devising new efficient light-weight distributed shortest paths solutions for large-scale Ethernet networks (see, e.g., [7–10]), where distance-vector algorithms seem to be an attractive alternative to link-state solutions when scalability is not an issue.

Related works. Among the distance vector algorithms one of the most interesting is surely DUAL (Diffuse Update ALgorithm) [2], which is free of the looping and count-to-infinity phenomena, thus resulting an effective practical solution (it is in fact part of CISCO's widely used EIGRP protocol). Another distance vector algorithm has been recently introduced in [11] and successively developed in [12], where it has been named DUST (Distributed Update of Shortest paThs). Compared with DUAL, DUST suffers of the looping and count-to-infinity phenomena, even though it has been designed to heuristically reduce the cases where these phenomena occur. However, DUST uses an amount of data structures per node which is much smaller than those of both DBF and DUAL. In [12, 11, 13] the practical performance of DBF, DUAL, and DUST have been measured, in terms of both number of messages, and space occupancy per node. This has been done by using both realistic and artificial dynamic scenarios. In particular, the Internet topologies of the *CAIDA IPv4 topology dataset* [14] (CAIDA - Cooperative Association for Internet Data Analysis is an association which provides data and tools for the analysis of the Internet infrastructure), and the random topologies with a power-law node degree distribution, generated by the *Barabási-Albert* algorithm [15], which are able to model many real-world networks such as the Internet, the World Wide Web, citation graphs, and some social networks. The outcome of these studies is that the space occupancy per node required by DUST is always much smaller than that required by both DBF and DUAL. In terms of messages, DUST outperforms both DBF and DUAL on the CAIDA topologies, while in the Barabási-Albert instances, DUST sends a number of messages that is slightly more than that of DUAL and much smaller than that of DBF.

Results of the paper. In this paper, we provide a new general, simple, and practical technique, named *Distributed Leafs Pruning* (DLP), which can be combined with every distance-vector algorithm with the aim of overcoming some of their main limitations in large scale networks (high number of messages sent, low scalability, poor convergence) at the price of a little overhead in the space occupancy per node.

In order to check the effectiveness of DLP, we combined it with DUAL and DUST, by obtaining two new algorithms named DUAL-DLP and DUST-DLP. Then, we implemented DUAL, DUST, DUAL-DLP and DUST-DLP in the OMNeT++ simulation environment [16], a network simulator which is widely used in the literature. As input to the algorithms, we considered the same instances used in [12, 11, 13], that is the Internet topologies of the *CAIDA IPv4 topology dataset* and the random topologies with a power-law node degree

distribution, generated by the *Barabási-Albert* algorithm. The results of our experimental study can be summarized as follows: the application of DLP to both DUST and DUAL implies a little memory overhead and provides a significant improvement in the global number of sent messages with respect to the original algorithms. In particular, the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.29 and 0.48. Similarly, the ratio between the number of messages sent by DUST-DLP and DUST is within 0.16 and 0.38. The space overhead is only between $8k$ and $11k$ bytes, which is very small compared, e.g., to the space required by DUAL that is about $5M$ bytes. We also considered for our experiments highly dynamic graphs where many edge weight updates occur, with the aim of checking the new technique in more realistic scenarios. Also in these cases, the application of DLP significantly decreases number of messages sent at the price of a little space overhead. Namely, the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.16 and 0.42 and the ratio between the number of messages sent by DUST-DLP and DUST is within 0.07 and 0.41. Also in this highly dynamic instances the space occupancy overhead is experimentally irrelevant.

2 Preliminaries

We consider a network made of processors linked through communication channels that exchange data using a message passing model, in which: each processor can send messages only to its neighbors; messages are delivered to their destination within a finite delay but they might be delivered out of order; there is no shared memory among the nodes of the network; the system is asynchronous, that is, a sender of a message does not wait for the receiver to be ready to receive the message.

Graph notation. We represent the network by an undirected weighted graph $G = (V, E, w)$, where V is a finite set of nodes, one for each processor, E is a finite set of edges, one for each communication channel, and w is a weight function $w : E \to \mathbb{R}^+ \cup \{\infty\}$ that assigns to each edge a real value representing the optimization parameter associated to the corresponding channel. An edge in E that links nodes $u, v \in V$ is denoted as $\{u, v\}$. Given $v \in V$, $N(v)$ denotes the set of neighbors of v. The maximum degree of the nodes in G is denoted by *maxdeg*. A path P in G between nodes u and v is denoted as $P = \{u, ..., v\}$. The *weight* of P is the sum of the weights of the edges in P. A *shortest path* between nodes u and v is a path from u to v with the minimum weight. The *distance* $d(u, v)$ from u to v is the weight of a shortest path from u to v. Given two nodes $u, v \in V$, the *via* from u to v is the set of neighbors of u that belong to a shortest path from u to v. Formally: $via(u, v) \equiv \{z \in N(u) \mid d(u, v) = w(u, z) + d(z, v)\}$. Given a time t, we denote as $w^t()$, $d^t()$, and $via^t()$ the edge weight, the distance, and the via at time t, resp. We denote a sequence of update operations on the edges of G by $\mathcal{C} = (c_1, c_2, ..., c_k)$. Assuming $G_0 \equiv G$, we denote as G_i, $0 \leq i \leq k$, the graph obtained by applying the operation c_i to G_{i-1}. The operation c_i either inserts a new edge in G_i, or deletes an edge of G_i, or modifies (either increases or

decreases) the weight of an existing edge in G_i. We consider the case in which $\mathcal{C}$ is a sequence of *weight increase* and *weight decrease* operations, that is operation c_i either increases or decreases the weight of edge $\{x_i, y_i\}$ by a quantity $\epsilon_i > 0$. The extension to *delete* and *insert* operations is straightforward.

Distance-vector algorithms. We consider the generic routing problem between all the nodes of a network, in which each node needs to find a shortest path to each other node. This problem can be tackled in different ways. The most reliable, robust and used approach is that based on distributed all-pairs shortest paths. We are interested in the practical case of a dynamic network in which an edge weight change (increase/decrease) can occur while one or more other edge weight changes are under processing. A processor v of the network might be affected by a subset of these changes. As a consequence, v could be involved in the *concurrent* executions related to such changes.

Distance-vector routing algorithms based on shortest-paths usually share a set of common features. In detail, given a graph $G = (V, E, w)$, a generic node v 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; maintains and updates its own routing table that has one entry for each $s \in V$, which consists of at least two fields: $\mathtt{D}^t[v, s]$, the estimated distance between v and s at time t, and $\mathtt{VIA}^t[v, s]$, the neighbor used to forward data from v to s at time t; handles edge weight increases and decreases either by a single procedure (see, e.g., [2]), which we denote as WeightChange, or separately (see, e.g., [12]) by two procedures, which we denote as WeightIncrease and WeightDecrease; requests information to its neighbors through a message denoted as *query*, receives replies by them through a message denoted as *reply*, and propagates a variation to the estimated routing information as follows:

- if v is performing WeightChange, then it sends to its neighbors a message, from now on denoted as *update*; a node that receives this kind of message executes procedure Update;
- if v is performing WeightIncrease or WeightDecrease, then it sends to its neighbors message *increase* or *decrease*, resp.; a node that receives *increase*/*decrease* executes procedure Increase/Decrease, resp.

3 The new technique

The main goal of Distributed Leafs Pruning (DLP) is to reduce the number of messages sent by a generic distance-vector algorithm, at the price of a little overhead in the space occupancy per node. DLP has been designed to be efficient mainly in a class of networks which is highly important in practice. This is the class of networks having a power-law node degree distribution which includes many of the currently implemented communication infrastructures, like Internet, WWW, some social networks, and so on [17]. For sake of simplicity, from now on we refer to this class as "power-law networks". Examples of power-law networks are the Internet topologies of the *CAIDA IPv4 topology dataset* [14]

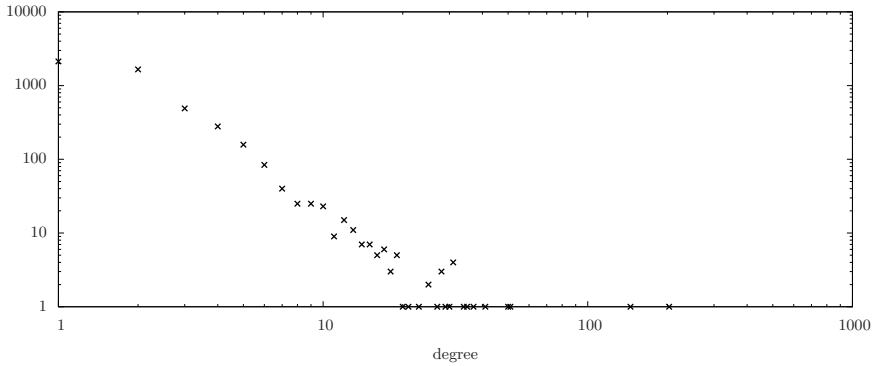

Fig. 1. Power-law node degree distribution of a graph of the *CAIDA IPv4 topology dataset* with 5000 nodes and 6109 edges.

(see, e.g., Fig. 1), and the artificial instances generated by the *Barabási-Albert* algorithm [15].

The idea underlying DLP is very simple and it is based on the following observations: (i) a power-law network with n nodes typically has average node degree much smaller than n and a number of nodes with unitary degree which is generally high. For example, some graphs of the *CAIDA IPv4 topology dataset* have average node degree approximately equal to $n/2000$, and a number of nodes with unitary degree approximately equal to $n/2$; (ii) nodes with unitary degree does not provide any useful information for the distributed computation of shortest paths. In fact, any shortest path from a node with degree one v to any other node of the network has necessarily to pass through the unique neighbor of v in the network.

To describe the technique we need to introduce some preliminary definitions. Given an undirected weighted graph $G = (V, E, w)$, the *core* of G is the graph $G_c = (V_c, E_c, w_c)$ which represents the maximal connected subgraph of G having all nodes of degree greater than one. A node $v \in V$ is a *central node* if $v \in V_c$, otherwise v is a *peripheral node*. An edge of G that links two central nodes is a *central edge*, an edge that links a central node with a peripheral node is a *peripheral edge*. For each peripheral node u, the unique central node v adjacent to u is called the *owner* of u.

Data structures. Given a generic distance-vector algorithm **A**, DLP requires that a generic node of G stores some additional information with respect to those required by **A**. In particular, a node v needs to store and update information about central and peripheral nodes and edges of G. To this aim, v maintains a data structure called *Classification Table*, denoted as CT_v, which is an array containing one entry $CT_v[s]$, for each $s \in V$, representing the list of the peripheral neighbors of s. A central node is not present in any list of CT_v. A peripheral node is present in $CT_v[s]$, for exactly one $s \in V$, and s is its owner. Each list

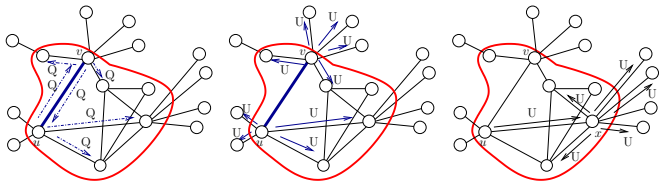

Fig. 2. (a) Nodes u and v ask for information to their central neighbors by sending them message Q (query); (b) Nodes u and v propagate updated routing information to all their neighbors through message U (update); (c) A central node x receiving an update message U, propagates it to its neighbors.

contains at most *maxdeg* entries and the sum of the sizes of all the lists is always smaller than n. Hence the space overhead per node due to CT_v is $O(n)$.

Properties. The main purpose of DLP is to force distributed computation to be carried out only by the central nodes. The peripheral nodes receive updates about routing information passively from the respective owners, without starting any kind of distributed computation. Then, the larger is the set of the peripheral nodes of the network, the bigger is the improvement in the global number of messages sent by the algorithm. The following lemma introduces some basic relationships between the paths that link central and peripheral nodes.

Lemma 1. *Given an undirected weighted graph $G = (V, E, w)$, and its core $G_c = (V_c, E_c, w_c)$, let $\{p, c\}$ be a peripheral edge in which $c \in V_c$ at time t. The following relations hold:*

- $d^t(x, p) = d^t(x, c) + w^t(c, p),\ \forall\ x \in V \setminus \{p\}$;
- $via^t(x, p) = via^t(x, c),\ \forall\ x \in V \setminus \{p\}$.

Proof. By definition, node p has a unique adjacent node at time t, then the shortest path $\{x, ..., p\}$ can be split in two sub-shortest paths: the path $\{x, ..., c\}$ with weight $d(x, c)$, and the edge $\{c, p\}$ with weight $w(c, p)$. □

Some useful additional relationships can be derived from Lemma 1. In particular, if between the time instants t_i and t_{i+1} the weight of the edge $\{p, c\}$ between a peripheral node p and his corresponding owner c changes, that is $w^{t_i}(p, c) \neq w^{t_{i+1}}(p, c)$, then p can update its own routing table toward each node of the network $x \in V$ simply by computing:

$$d^{t_{i+1}}(p, x) = d^{t_i}(p, x) + w^{t_{i+1}}(p, c) - w^{t_i}(p, c), \tag{1}$$

$$via^{t_{i+1}}(p, x) = \{c\}. \tag{2}$$

In a similar way, if a generic node of the network $x \in V$, between the time instants t_i and t_{i+1}, receives an update about a weight change in the path toward a generic central node c (that is, $d^{t_{i+1}}(x, c) \neq d^{t_i}(x, c)$), then the nodes involved in the

Event: node v invokes procedure UpdatePeripherals($s, \mathtt{D}_{old}[v,s]$)
Procedure: UpdatePeripherals($s, \mathtt{D}_{old}[v,s]$)
1 **if** $\mathtt{D}[v,s] \neq \mathtt{D}_{old}[v,s]$ **then**
2 **foreach** $k \in CT_v[s]$ **do**
3 $\mathtt{D}[v,k] := \mathtt{D}[v,k] + \mathtt{D}[v,s] - \mathtt{D}_{old}[v,s]$
4 $\mathtt{VIA}[v,k] := \mathtt{VIA}[v,s]$
5 update any specific data structures of **A**

Fig. 3. Pseudocode of procedure UpdatePeripherals.

change are x, c and the peripheral neighbors of c, if they exists. These nodes, by Lemma 1, can update their estimated routing tables simply by computing, for all peripheral nodes p with owner c:

$$d^{t_{i+1}}(x,p) = d^{t_i}(x,p) + d^{t_{i+1}}(x,c) - d^{t_i}(x,c), \tag{3}$$

$$via^{t_{i+1}}(x,p) = via^{t_{i+1}}(x,c). \tag{4}$$

Distributed Leafs Pruning. The application of DLP to a distance vector algorithm **A** induces a new algorithm denoted as **A**-DLP. The global behavior of **A**-DLP can be summarized as follows. While in a classic routing algorithm every node performs the same code thus having the same behavior, in **A**-DLP central and peripheral nodes have different behaviors. In particular, central nodes detect changes concerning both central and peripheral edges while peripheral nodes detect changes concerning only peripheral edges. If the weight of a central edge $\{u,v\}$ changes, then node u (v, resp.) performs the procedure provided by **A** for that change only with respect to central nodes for the distributed computation of the shortest paths between all the pairs of central nodes. During this computation, if u needs information by its neighbors, it asks only to neighbors in the core (see Fig. 2(a)). Once u (v) has updated its own routing information, it propagates the variation to all its neighbors through the *update*, *increase* or *decrease* messages of **A** (Fig. 2(b)). When a generic node x receives an *update*, *increase* or *decrease* message it stores the current value of $\mathtt{D}[u,s]$ in a temporary variable $\mathtt{D}_{old}[u,s]$. Now, if x is a central node, then it handles the change and updates its routing information by using the proper procedure of **A** (Update, Increase, or Decrease) and propagates the new information to its neighbors (see Fig. 2(c)). Otherwise, x handles the change and updates its routing information toward s by using Lemma 1 and the data received from its owner. At the end, x calls the procedure UpdatePeripherals reported in Fig. 3 using s and $\mathtt{D}_{old}[p,s]$ as parameters. If the routing table entry of s is changed (line 1), then the information about the peripheral neighbors of s, if they exist, is updated by using Equations 3 and 4 (lines 3–4).

If a weight change occurs in a peripheral edge $\{u,p\}$, then the central node u sends a $p\_change(p, w(u,p), u)$ message to each of its neighbors (Fig. 4(a)), while node p sends a $p\_change(p, w(u,p), u)$ message to its owner u, without starting any distributed computation. When a generic node x receives message p_change,

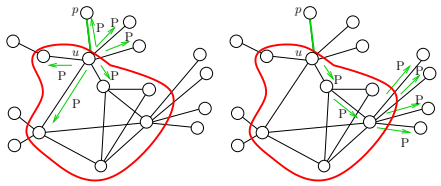

Fig. 4. (a) Node u, as a consequence of a weight change on edge $\{u, p\}$, sends message *p_change* (P) to all its neighbors; (b) Node x receiving a message P, propagates it to the whole network.

Event: node v receives the message p_change$(p, w(u,p), u)$ from u
Procedure: PERIPHERALCHANGE$(y, w(x,y), x)$

```
1  if w(x,y) ≠ (D[v,y] − D[v,x]) then
2      if v ≡ y then
3          foreach s ∈ V do
4              D[v,s] := D[v,s] − D[v,x] + w(x,y)
5              VIA[v,y] := x
6              update any specific data structures of A
7      else
8          D[v,y] := D[v,x] + w(x,y)
9          VIA[v,y] := VIA[v,x]
10         update any specific data structures of A
11         foreach k ∈ {x : x ∈ N(v) ∧ x ≠ u ∧ x ≠ y} do
12             send p_change(y, w(x,y), x) to k
```

Fig. 5. Pseudocode of procedure PERIPHERALCHANGE.

it simply performs procedure PERIPHERALCHANGE of Fig. 5, which is a modified flooding algorithm to forward the message over the network (Fig. 4(b)). Procedure PERIPHERALCHANGE first verifies at line 1 whether the message was already received by applying Lemma 1. If the message does not provide updated information, it is discarded. Otherwise, the procedure needs to update the data structures at x. We distinguish two cases: if x coincides with p, then the procedure updates the routing table for all the nodes $s \in V$, using Equations 1 and 2 (see Lines 2-6). Otherwise ($x \neq p$), the procedure simply updates the routing table entry concerning p by using Lemma 1 (Lines 8-10). At this point, the procedure propagates the information about the change, forwarding message *p_change* to all the neighbors, except to nodes u and possibly p (Lines 11–12).

4 Combination of DLP with distance-vector algorithms

Combination of DLP with DUAL. In DUAL, each node v maintains, for each destination s a set of neighbors called the feasible successor set $F[v,s]$, and for each $u \in N(v)$, the distance $\mathtt{D}[u,s]$ from u to s. $F[v,s]$ is computed as follows: node $u \in N(v)$ is in $F[v,s]$ if the estimated distance $\mathtt{D}[u,s]$ from u to s is smaller than the estimated distance $\mathtt{D}[v,s]$ from v to 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, i.e. the neighbor to be used to forward data, which is stored as $\mathtt{s}[v,s]$. If a weight change occurs in an adjacent edge, then node v executes procedure WeightChange. This procedure updates the data structures and, if the current successor, after the update, is no more in $F[v,s]$, it selects an alternative successor. If $F[v,s]$ becomes empty, then v turns in active state and starts a synchronous update procedure, known as a *diffuse computation*, to recompute the set, by exchanging *query* and *reply* messages with neighbors. Node v cannot choose its new successor to s until the diffuse computation terminates. When a neighbor u receives the queries, it updates $F[u,s]$. If u has a successor to s after such update, it replies to the query by sending message *reply* containing its own distance to s. Otherwise, u becomes active in turn and, before replying to v's original query, it forwards the diffuse computation by sending out queries and waiting for the replies from its neighbors. When a node receives messages *reply* by all its neighbors, it turns back in passive state, sends out all the replies, updates its data structures and finishes the diffuse computation. At the end of each diffuse computation, v sends message *update* containing the new computed distance to its neighbors. To handle concurrent weight changes and updates, each node implements, by using a large set of data structures, a finite state machine.

DUAL can be combined with DLP as described in Section 2. In addition, the generic procedures reported in Fig. 3 and 5, are modified, by using the data structures of DUAL, to generate two specific procedures, called DUAL-PeripheralChange and DUAL-UpdatePeripherals. The main changes can be summarized as follows: (i) in Procedure PeripheralChange (at Lines 5 and 9) and in Procedure UpdatePeripherals (at Line 4) the data structure `VIA` is replaced by the data structure `s`; (ii) in Procedure PeripheralChange (at Lines 6 and 10) and in Procedure UpdatePeripherals (at Line 5) the specific data structures of DUAL, described above, are updated by using Lemma 1. The correctness of DUAL-DLP directly follows from Lemma 1 and from the correctness of DUAL [2].

Combination of DLP with DUST. DUST maintains only the routing table described in Section 2 and, for each node v and for each source s, $\mathtt{VIA}[v,s]$ contains the set $\mathtt{VIA}[v,s] \equiv \{v_i \in N(v) \mid \mathtt{D}[v,s] = w(v,v_i) + \mathtt{D}[v_i,s]\}$. Algorithm DUST starts every time an operation c_i on edge (x_i,y_i) occurs. 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 performs procedure WeightIncrease (WeightDecrease) that sends message *increase*(x_i,s) (*decrease*$(x_i,s,\mathtt{D}[x_i,s])$) to y_i for each $s \in V$. Node y_i has the same behavior of x_i. If a node v receives message *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 with respect to 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 removes u from $\mathtt{VIA}[v, s]$. 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 distance, by sending message *get-dist*(v, s) to v_i. When v_i receives *get-dist*(v, s) by v, it performs procedure SENDDIST which sends $\mathtt{D}[v_i, s]$ to v, unless one of the following two conditions holds: (i) $\mathtt{VIA}[v_i, s] \equiv \{v\}$; (ii) v_i is updating its routing table with respect to destination s. In this case 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. If the estimated distance is 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 $\mathtt{D}[v_j, s]$.

DUST can be combined with DLP, by modifying its behavior as described Section 2. In addition, the generic procedures reported in Fig.s 3 and 5, are modified, by using the data structures of DUST, to generate two specific procedures, called DUST-PERIPHERALCHANGE and DUST-UPDATEPERIPHERALS. The main changes can be summarized as follows: (i) in Procedure PERIPHERALCHANGE (at Lines 5 and 9) and in Procedure UPDATEPERIPHERALS (at Line 4) the data structure VIA is modified to be a set instead of a single value variable; (ii) since DUST does not use any additional data structures, in Procedure PERIPHERALCHANGE Lines 6 and 10 are removed and in Procedure UPDATEPERIPHERALS Line 5 is removed. The correctness of DUST-DLP directly follows from Lemma 1 and from the correctness of DUST [11].

5 Experimental analysis

In this section we report the results of our experimental study on algorithms DUAL, DUST, DUAL-DLP, and DUST-DLP. The experiments have been carried out on a workstation with an Intel Core2 2.66 GHz processor and 8Gb RAM and consist of simulations within the OMNeT++ 4.0p1 environment [16].

Executed tests. For the experiments we used both real-world and artificial instances of the problem. In detail, we used the *CAIDA IPv4 topology dataset* [14] and a class of random power-law networks generated by the *Barabási-Albert* algorithm [15].

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. The monitors collect data by sending probe messages continuously to destination IP

addresses. Destinations are selected randomly from each routed IPv4/24 prefix on the Internet such that a random address in each prefix is probed approximately every 48 hours. The current prefix list includes approximately 7.4 million prefixes. For each destination selected, the path from the source monitor to the destination is collected, in particular, data collected for each path probed 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 (both source/destination hosts and intermediate hops), edges represent links among hops and weights are given by Round Trip Times. As the graph G_{IP} consists of almost 35000 nodes, we cannot use it for the experiments, as the amount of memory required to store the routing tables of all the nodes is $O(n^2 \cdot maxdeg)$ for any implemented algorithm. Hence, we performed our tests on connected subgraphs of G_{IP}, with a variable number of nodes and edges, 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} (we denoted each n nodes subgraph of G_{IP} with G_{IP-n}) and a set of k concurrent edge updates, where k assumes values in $\{5, 10, \ldots, 100\}$ or in $\{105, 110, \ldots, 200\}$. An edge update consists of multiplying the weight of a random selected edge by a percentage value randomly chosen in $[50\%, 150\%]$. For each test configuration (a dynamic graph with a fixed value of k) we performed 5 different experiments (for a total amount of runnings equal to 100) and we report average values.

A Barabási–Albert topology is generated by iteratively adding one node at a time, starting from a given connected graph with at least two nodes. A newly added node is connected to any other existing nodes with a probability that is proportional to the degree that the existing nodes already have. We randomly generated a set of different tests, where a test consists of a dynamic graph characterized by a n nodes Barabási–Albert random graphs, denoted as G_{BA-n} and a set of k concurrent edge updates, where k assumes values in $\{5, 10, \ldots, 100\}$ or in $\{105, 110, \ldots, 200\}$. Edge weights are non-negative real numbers randomly chosen in $[1, 10000]$. Edge updates are randomly chosen as in the CAIDA tests. For each test configuration (a dynamic graph with a fixed value of k) we performed 5 different experiments (for a total amount of runnings equal to 100) and we report average values.

Analysis. In Fig. 6 we report the number of messages sent by DUST, DUST-DLP, DUAL and DUAL-DLP on subgraphs $G_{IP-5000}$ of G_{IP} having 5000 nodes and 6109 edges in the cases where the number k of modifications is in $\{5, 10, \ldots, 100\}$.

In Fig. 7 we report the number of messages sent by DUST, DUST-DLP, DUAL and DUAL-DLP on subgraphs $G_{IP-1200}$ of G_{IP} having 1200 nodes and 1443 edges in the cases where the number k of modifications is in $\{105, 110, \ldots, 200\}$.

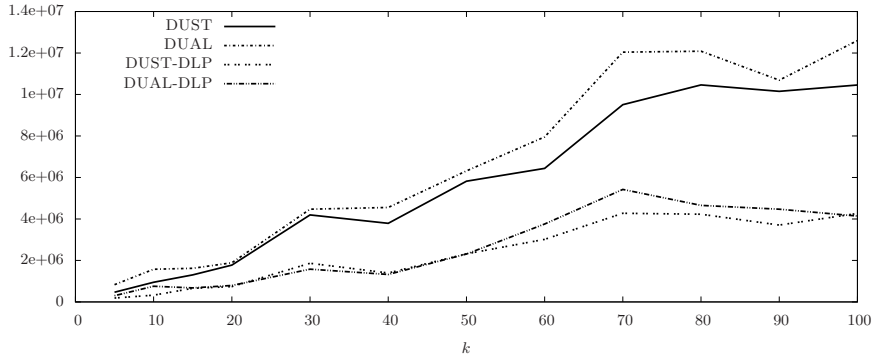

Fig. 6. Number of messages sent on a 5000 nodes subgraph $G_{IP-5000}$ of G_{IP}.

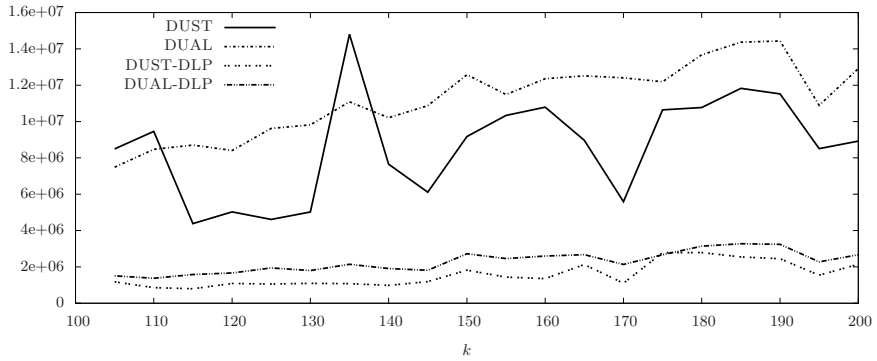

Fig. 7. Number of messages sent on a 1200 nodes subgraph $G_{IP-1200}$ of G_{IP}.

The figures show that the application of the technique DLP on both DUST and DUAL provides a significant improvement in the global number of sent messages. In the tests of Fig. 6, the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.29 and 0.48 which means that the number of messages sent by DUAL-DLP is between 29% and 48% that of DUAL. Similarly, the ratio between the number of messages sent by DUST-DLP and DUST is within 0.26 and 0.38. In the tests of Fig. 7 the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.16 and 0.23 and the ratio between the number of messages sent by DUST-DLP and DUST is within 0.07 and 0.26. In both cases, the figures show that in most of the cases DUST sends less messages than DUAL and DUST-DLP sends less messages than DUAL-DLP.

Regarding Barabási-Albert graphs, we performed similar tests as for G_{IP}. Fig. 8 reports the number of messages sent by DUST, DUST-DLP, DUAL and DUAL-DLP on a graph $G_{BA-5000}$ with 5000 nodes and 6242 edges where

Graph	Algorithm	MAX		AVG	
		Bytes	**Overhead**	**Bytes**	**Overhead**
$G_{IP-5000}$	DUAL	5 200 000	—	186 090	—
	DUST	40 032	—	40 000	—
	DUAL-DLP	5 208 508	0.16%	194 598	4.57%
	DUST-DLP	48 540	21.25%	48 508	21.27%
$G_{BA-5000}$	DUAL	5 605 000	—	187 420	—
	DUST	40 012	—	40 000	—
	DUAL-DLP	5 616 516	0.20%	198 936	6.14%
	DUST-DLP	51 528	28.78%	51 516	28.79%

Table 1. Space complexity - Results of a dynamic execution with $k = 100$ over $G_{IP-5000}$ and $G_{BA-5000}$

the number k of modifications is in $\{5, 10, \ldots, 100\}$, while Fig. 9 reports the number of messages sent by DUST, DUST-DLP, DUAL and DUAL-DLP on a graph $G_{BA-1200}$ with 1200 nodes and 1836 edges where the number k of modifications is in $\{105, 110, \ldots, 200\}$. In most of these executions, differently from the previous case, DUAL behaves better than DUST. However, the use of DLP again shows a clear improvement in the global number of sent messages by both algorithms. In detail, the ratio between the number of messages sent by DUAL-DLP and DUAL is within 0.30 and 0.37 and within 0.32 and 0.42 in the tests of Fig. 8, while the ratio between the number of messages sent by DUST-DLP and DUST is within 0.16 and 0.38 in $G_{BA-5000}$ and within 0.08 and 0.41 in the tests of Fig. 9.

To conclude our analysis, we analyzed the space occupancy per node, which is summarized in Tables 1 and 2. DUAL requires a node v to store, for each destination, the estimated distance given by each of its neighbors and a set of variables used to guarantee loop-freedom, while DUST 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 DUST is much smaller than that of DUAL. The space occupancy of the data structure used by DLP is not a function of the degree of the graph and is always bounded, in the worst case, by n. It follows that the application of the technique on both DUST and DUAL implies a really small memory overhead compared with the space occupancy of these algorithms. In fact, the space overhead of e.g. $G_{IP-5000}$ is only 8 508 bytes, which is very small compared, e.g., to the space required by DUAL that is about $5M$ bytes. In Table 1 and 2 the relative overheads, that are the percentage of the ratios between the space overhead and the space required by DUAL and DUST, are also reported. It is evident from Table 1 and 2 that the space occupancy overhead resulting from the application of DLP is experimentally irrelevant.

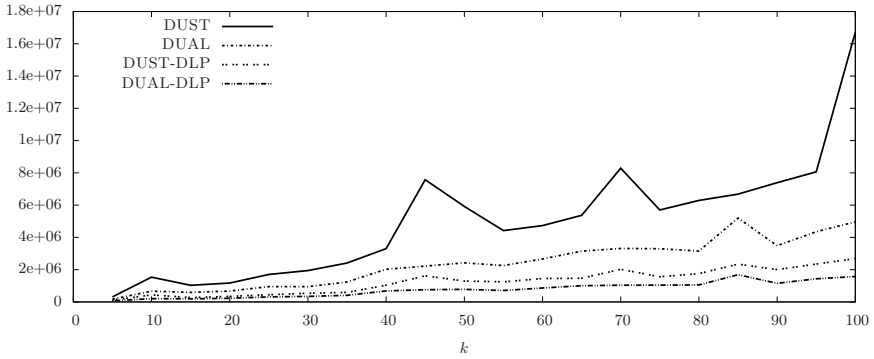

Fig. 8. Number of messages sent on a 5000 nodes subgraph $G_{BA-5000}$ of G_{BA}.

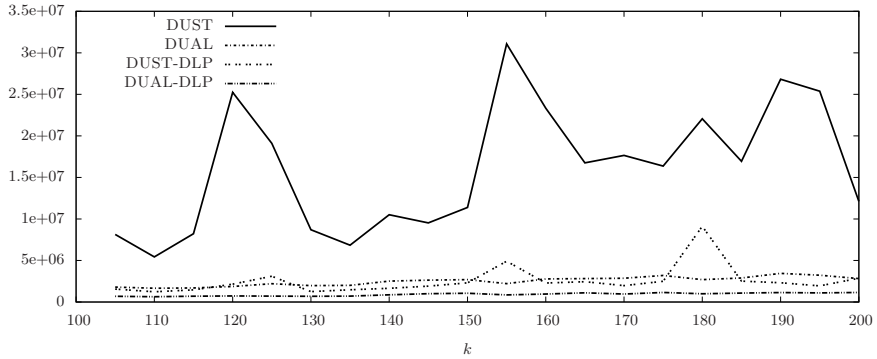

Fig. 9. Number of messages sent on a 1200 nodes subgraph $G_{BA-1200}$ of G_{BA}.

6 Conclusions and future work

We have proposed a simple and practical speed-up technique, which can be combined with every distance vector routing algorithm based on shortest paths, allowing to reduce the total number of messages sent by that algorithm. We have combined the new technique with DUAL and the recent DUST. We have given experimental evidence that these combinations lead to an important gain in terms of the number of messages sent by these algorithms at the price of a little increase in terms of space occupancy per node.

Some research directions deserve further investigation: (i) we have experimentally show that the application of DLP has a better impact on the number of messages sent by DUST; it would be interesting to evaluate the impact of DLP on the convergence of DUST; (ii) it would be interesting to know how DLP is scalable to bigger networks than those considered in this paper.

Graph	Algorithm	MAX		AVG	
		Bytes	Overhead	Bytes	Overhead
$G_{IP-1200}$	DUAL	1 248 000	—	44 430	—
	DUST	9 602	—	9 600	—
	DUAL-DLP	1 250 660	0.21%	47 086	6.00%
	DUST-DLP	13 520	40.08%	12 256	27.67%
$G_{BA-1200}$	DUAL	834 000	—	48 360	—
	DUST	9 624	—	9 600	—
	DUAL-DLP	836 136	0.26%	50 496	4.42%
	DUST-DLP	12 549	30.39%	11 732	22.21%

Table 2. Space complexity - Results of a dynamic execution with $k = 200$ over $G_{IP-1200}$ and $G_{BA-1200}$

Acknowledgments

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.

References

1. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D.: Partially dynamic efficient algorithms for distributed shortest paths. Theoretical Computer Science **411** (2010) 1013–1037
2. Garcia-Lunes-Aceves, J.J.: Loop-free routing using diffusing computations. IEEE/ACM Trans. on Networking **1**(1) (1993) 130–141
3. Italiano, G.F.: Distributed algorithms for updating shortest paths. In: International Workshop on Distributed Algorithms. Volume 579 of Lecture Notes in Computer Science. (1991) 200–211
4. McQuillan, J.: Adaptive routing algorithms for distributed computer networks. Technical Report BBN Report 2831, Cambridge, MA (1974)
5. Bertsekas, D., Gallager, R.: Data Networks. Prentice Hall International (1992)
6. Moy, J.T.: OSPF: Anatomy of an Internet routing protocol. Addison-Wesley (1998)
7. Elmeleegy, K., Cox, A.L., Ng, T.S.E.: On count-to-infinity induced forwarding loops in ethernet networks. In: Proceedings IEEE INFOCOM. (2006) 1–13
8. Myers, A., Ng, E., Zhang, H.: Rethinking the service model: Scaling ethernet to a million nodes. In: ACM SIGCOMM HotNets, ACM Press (2004)
9. Ray, S., Guérin, R., Kwong, K.W., Sofia, R.: Always acyclic distributed path computation. IEEE/ACM Trans. on Networking **18**(1) (2010) 307–319
10. Zhao, C., Liu, Y., Liu, K.: A more efficient diffusing update algorithm for loop-free routing. In: 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom09), IEEE Press (2009) 1–4
11. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Maurizio, V.: A new fully dynamic algorithm for distributed shortest paths and its experimental evaluation. In: 9th Symposium on Experimental Algorithms (SEA 2010). Volume 6049 of Lecture Notes in Computer Science. (2010) 59–70

12. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Maurizio, V.: Engineering a new algorithm for distributed shortest paths on dynamic networks. Submitted for publication. http://informatica.ing.univaq.it/misc/DUST/ Prel. version in [11].
13. D'Angelo, G., Frigioni, D., Maurizio, V.: An experimental study of distributed algorithms for shortest paths on real networks. In: 12th Italian Conference on Theoretical Computer Science (ICTCS). (2010)
14. Hyun, Y., Huffaker, B., Andersen, D., Aben, E., Shannon, C., Luckie, M., Claffy, K.: The CAIDA IPv4 routed/24 topology dataset. *http* : *//www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml*
15. Albert, R., Barabási, A.L.: Statistical mechanics of complex network. Reviews of Modern Physics **74** (2002) 47–97
16. : Omnet++: the discrete event simulation environment. `http://www.omnetpp.org/`
17. Albert, R., Barabási, A.L.: Emergence of scaling in random networks. Science **286** (1999) 509–512