Computing All the Best Swap Edges Distributively*

P. Flocchini† L. Pagli‡ G. Prencipe§ N. Santoro ¶
P. Widmayer‖

Abstract

Recently great attention has been given to *point-of-failure swap rerouting*, an efficient technique for routing in presence of transient failures. According to this technique, a message follows the normal routing table information unless the next hop has failed; in this case, it is redirected towards a precomputed link, called *swap*; once this link has been crossed, normal routing is resumed. The amount of precomputed information required in addition to the routing table is rather small: a single link per each destination. Several efficient serial algorithms have been presented to compute this information; none of them can unfortunately be efficiently implemented in a distributed environment. In this paper we present protocols, based on a new strategy, that allow the efficient distributed computation of all the optimal swap edges under several optimization criteria. In systems allowing long messages, we develop solution protocols based on the same strategy that use only $O(n)$ messages without increasing the total amount of transmitted data items.

*Research partially supported by NSERC Canada, TECSIS Co., and the Swiss BBW 03.0378-1 for EC contract 001907 (DELIS), and "Progetto ALINWEB: Algoritmica per Internet e per il Web", MIUR, Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale. A preliminary version of this paper has been presented at the 8th International Conference on Principles of Distributed Systems (OPODIS 2004).

†University of Ottawa, Canada, flocchin@site.uottawa.ca

‡Università di Pisa, Italy, pagli@di.unipi.it

§Corresponding author: Università di Pisa, Dipartimento di Informatica, Largo Bruno Pontecorvo, 3, 5600 Pisa, Italy, email: prencipe@di.unipi.it, Tel: +39 050 2213148, Fax: +39 050 2212726

¶Carleton University, Canada, santoro@scs.carleton.ca

‖ETH, Zurich, Switzerland, widmayer@inf.ethz.ch

1 Introduction

In systems using shortest-path routing tables, a single link failure is enough to interrupt the message transmission by disconnecting one or more shortest-path spanning trees. The on-line recomputation of an alternative path or of the entire new shortest path trees, rebuilding the routing tables accordingly, is rather expensive and causes long delays in the message's transmission [6, 11].

****AGGIUNGERE***** New applications, such as Voice-over-IP, have strict requirements in terms of trans- mission time, and it is very difficult to achieve sufficiently fast reaction time. ********** Hopefully, some of the recomputation costs will be reduced if the serial algorithms for dynamic graphs (e.g., those of [2]) could be somehow employed; this cost can be further reduced if the distributed dynamic algorithm is applied [1].

An alternative approach is to precompute additional information and use it to augment the shortest-path routing tables so to make them operate when a failure occurs. Examples of this approach are techniques (e.g., see [5]) of precomputing several edge-disjoint spanning trees for each destination. However, the alternative routes do not satisfy any optimization criterion (such as shortest path) even in the case when, at any time, only one link (not necessarily the same at all times) might be down.

A new strategy has been recently proposed [3, 6, 8, 9, 12]. It starts from the idea of precomputing, for each link in the tree, a single non-tree link (the *swap edge*) able to reconnect the network should the first fail. The strategy, called *point-of-failure swap rerouting* is simple: normal routing information will be used to route a message to its destination. If, however, the next hop is down, the message is first rerouted towards the swap edge; once this is crossed, normal routing will resume. Experimental results [12] show that the tree obtained from the swap edge is very close to the new shortest-path spanning tree computed from scratch,

*****AGGIUNGERE****** and the traveling messages can switch to the swap edge, in case of failure, almost immediately after the fault detection. ********** Clearly, some swap edges are preferable to others. In [9], four main objective functions were defined, giving rise to four different problems. These optimization functions describe the goal to find a new tree that minimizes, respectively, the distance between the point of failure and the root (F_{dist}); the sum of distances (F_{sum}), the largest increment in the distance (F_{incr}), and the largest distance (F_{max}) of all nodes below the point of failure to the root.

*****AGGIUNGERE****** The distance from the point of failure to the root is a good function to minimize if in the considered application the failure are not permanent and last for short time; hence the delivery of the messages arrived to the the failed edge is completed on the recovered path. If the failures last for longer, we must consider all the nodes in the subtree disconnected by the failure, that might send messages, not only the messages already arrived to the point-of-failure. In this case the optimization function are computed taking into account all the nodes of the subtree. *********

In [9] it was shown that these problems can be solved sequentially with different complexities: F_{dist} and F_{incr} in time $O(m \cdot \alpha(m, n))$, F_{sum} in $O(n^2)$, and F_{max} in $O(n\sqrt{m})$,

where $\alpha(m,n)$ is the functional inverse of Ackermann's function. These bounds are achieved using Tarjan's sophisticated technique of *transmuters* [13]. Unfortunately, there is currently no efficient distributed implementation of this sequential technique. From a distributed point of view, only the first of those problems, F_{dist}, has been investigated and solved. A simple but non-optimal solution has been developed in [6]. An efficient optimal solution has been recently proposed [4]. No efficient distributed solution exists to date for the problems F_{sum}, F_{incr}, and F_{max}. These problems appear to be rather important, since they minimize the average, the additional and the maximum delivery time of a message issued at any node. In this paper, we will be able to solve efficiently all three problems.

We propose two general distributed strategies, each solving the three problems with simple modifications. The first scheme uses $O(n_r^*)$ short messages, where n_r^* is the size of the transitive closure of $T_r \setminus \{r\}$; note that $0 \leq n_r^* \leq (n-1)(n-2)/2$. In the second scheme the number of messages decreases to $O(n)$ if long messages are allowed. Both schemes use an overall data complexity of $O(n_r^*)$.

Note that, unlike the sequential case where the algorithms have different costs for different optimization functions, the proposed distributed solutions have the same complexity for all functions.

2 Terminology and Problems

We will use some standard terminology and definitions on graphs as in [9]. Let $G = (V, E)$ be the 2-edge-connected undirected graph, with $n = |V|$ vertices and $m = |E|$ edges, describing the communication topology of the system. A non negative real number $w(e)$, called *length*, is associated to each edge $e \in E$. The length of a path is the sum of the lengths of all the edges in the path, and the distance between two vertices x and y is the length of a shortest path between them.

Let $r \in V$ be a distinguished node called *source*, and let $T = (V, E(T))$ be a given (shortest-path) spanning tree of G rooted in r. Let $d_T(u,v)$ (shortly $d(u,v)$) denote the distance between nodes u and v in T. The *weight* of T, denoted by $W(T)$, is defined as the sum of the distances from all the nodes to the root; i.e., $W(T) = \sum_{x \in V} d(x,r)$. Given a node $q \in V$, let $T_q = (V(T_q), E(T_q))$ denote the subtree of T rooted in q; let $n(T_q) = |V(T_q)|$ denote the number of nodes in T_q, and let $W(T_q) = \sum_{u \in V(T_q)} d(u,q)$ denote the *weight* of T_q. For a node $u \in V$, we denote by $C(u,T_q)$, $p(u,T_q)$, and $A(u,T_q)$ the set of children, the parent and the set of ancestors of u in T_q, respectively; if $q = r$ (i.e., $T_q = T$), we will simply write $C(u)$, $p(u)$ and $A(u)$.

Consider an edge $e = (x,y) \in E(T)$, and w.l.g. let $y = p(x)$. If such an edge is removed, the tree is disconnected in two subtrees: T_x and $T \setminus T_x$. A *swap* edge for $e = (x,y)$ is any edge $e' \in E \setminus \{e\}$ that connects the two subtrees; its use instead of e forms a new tree $T_{e/e'}$, called swap tree. For two nodes $u, v \in N$, let $d_{T_{e/e'}}(u,v)$ (shortly $d_{e/e'}(u,v)$) denote their distance in $T_{e/e'}$. Let $\mathcal{S}_e$ denote the set of all possible swap trees with respect to e.

Depending on the goal of the swapping algorithm, some swap edges are preferable to

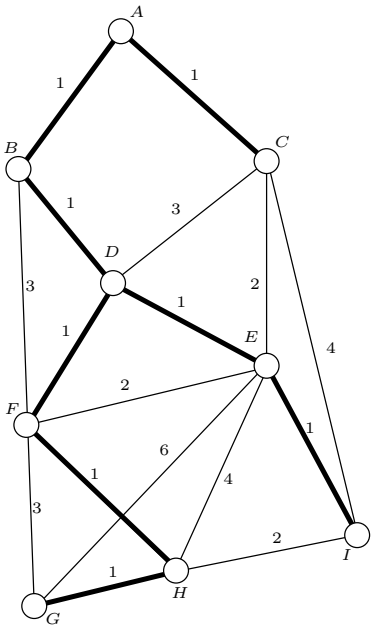

Figure 1: An example: the thick line represents the starting SPT, rooted in A.

others. Given an objective function F over $\mathcal{S}_e$, an *optimal* or *best* swap edge for a link $e = (x, y)$ is a swap edge e' such that $F(T_{e/e'})$ is minimum. For each choice of F we have a different optimization problem. *********AGGIUNGERE******* In [4] the problem of finding a new tree that minimizes the distance between the point of failure and the root (F_{dist}) has been studied. *********AGGIUNGERE******* Hence, here we consider the main problems studied in [9] not yet examined in a distributed setting:

1. (F_{sum} **problem**) *Choose a swap edge e' that minimizes the sum of the distances* from all nodes in T_x *to r when replacing e with e' :*
   $\min_{T_{e/e'} \in \mathcal{S}_e}\{F_{sum}(T_{e/e'})\}$, where $F_{sum}(T_{e/e'}) = \sum_{u \in V(T_x)} d_{e/e'}(u, r)$

2. (F_{incr} **problem**) *Choose a swap edge that minimizes the maximum increment of the distance from r to any node in T_x when replacing e with e' :*
   $\min_{T_{e/e'} \in \mathcal{S}_e}\{F_{incr}(T_{e/e'})\}$ where $F_{incr}(T_{e/e'}) = \max_{u \in V(T_x)}(d_{e/e'}(u, r) - d(u, r))$.

3. (F_{max} **problem**) *Choose a swap edge that minimizes the maximum distance from the nodes in T_x to r when replacing e with e' :*
   $\min_{T_{e/e'} \in \mathcal{S}_e}\{F_{max}(T_{e/e'})\}$ where $F_{max}(T_{e/e'}) = \max_{t \in V(T_x)} d_{e/e'}(t, r)$.

As an example, consider the 2-edge-connected graph and its shortest-path spanning-tree rooted in A shown in Figure 1. The best swap edge for link (D, B) is (E, C) when considering F_{sum} or F_{incr}; (F, B) (D, C) and (E, C) are best swap edges if using F_{max}.

3 Solution Protocols

3.1 Algorithmic Shell and Computational Tools

3.1.1 A Generic Algorithm

Consider the problem of computing the best swap edge for link $e = (x, p(x)) \in E(T)$, where $p(x)$ denotes the parent of x in T. We now present a generic distributed algorithm to perform this computation; the details of its modules depend on the objective function F and will be described later.

The algorithm is started by x; during its execution, each node $z \in V(T_x)$ will determine the best local swap edge (z, z') for $(x, p(x))$, according to the objective function. Among the local swap edges of all nodes, a swap edge yielding the global minimum cost will be then selected. More precisely, we define:

PROCEDURE BSE$(F, (x, p(x))$

1. Node x determines, among its local swap edges for $(x, p(x))$, the one that minimizes F. As we will see, x is the only node that can do so without any additional information.
2. After this, x sends to each child the *enabling information* the child needs to compute the best among its local swap edges for $(x, p(x))$.
3. Upon receiving the enabling information from its parent, a node computes the best among its local swap edge for $(x, p(x))$; it then sends enabling information to its children. This process terminates once the leaves of T_x are reached.
4. The leaves then start a *minimum finding* process to determine, among the swap edges chosen by the nodes in T_x, the one that minimizes the objective function F.
5. The optimal swap edge for $(x, p(x))$ is thus determined at node x.

This procedure finds the best swap edge for link $(x, p(x))$ (according to F). Thus, the generic algorithm to find all the best swap edges is

ALGORITHM BEST F-SWAP

1. PRE-PROCESSING(F)
2. $\forall x \neq r$: BSE$(F, (x, p(x)))$

where PRE-PROCESSING(F) is a preliminary process to be executed only if the nodes do not have the required initial information.

3.1.2 Identifying Swap Edges

Before proceeding with the instantiation of the generic algorithm for each of the objective functions, we describe a tool that allows a node u to distinguish, among its incident

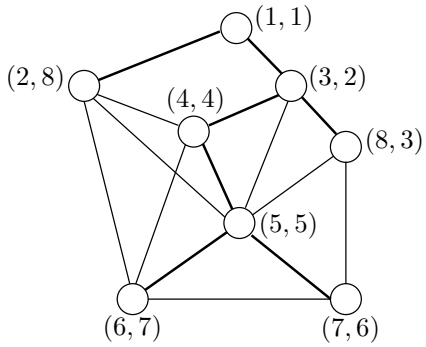

Figure 2: Labelling of the nodes.

edges, the ones that are swap edges for $(x, p(x))$. This tool, introduced in [4], will be used regardless of the objective function, and it is based on the fact that, in a rooted tree, the partial order induced by the relation "parent" has dimension at most 2, as shown below.

Consider the following labeling $\lambda : V \to \{1, \ldots, n\}^2$ of the nodes: for $u \in V$ let $\lambda(u) = (a_u, b_u)$, where a_u is the numbering of u in the *preorder* traversal of T, and b_u is the numbering of u in the *inverted preorder* traversal of T (i.e., when the order of the visit of the children is inverted).

Consider the dominance relationship $\geq$ among pairs: let $\lambda(u) = (a_u, b_u)$ and $\lambda(v) = (a_v, b_v)$, then $\lambda(u) \geq \lambda(v)$ if $a_u \geq a_v$ and $b_u \geq b_v$). The pairs of numbers given by the labeling under the dominance relationship $\geq$ form a partial order $(\lambda, \geq)$. The "dominance" relationship between these pairs completely characterizes the relationship "descendant" in the tree:

Property 1 *A node v is descendant of a node u in T* if and only if $\lambda(u) \geq \lambda(v)$.

Based on Property 1, we can now see how the labeling can be used by a node u to recognize its incident swap edges for a given link $(x, p(x))$ (refer to Figure 2).

Property 2 *An edge $(u, v) \in E \setminus E(T)$ is a swap for $(x, p(x)) \in E(T)$* if and only if *only one of u and v (but not both) is a descendant of x in T.*

Thus, node $u \in T_x$ will be able to tell whether its incident edge (u, v) is a swap edge for $(x, p(x))$ simply by comparing $\lambda(v)$ with $\lambda(x)$: (u, v) is *not* a swap edge for $(x, p(x))$ if and only if $\lambda(v) \geq \lambda(x)$.

In our algorithms, we will assume that this labeling is available to the nodes, and that every node knows what are its incident swap edges. If not available, such a labeling will be given to the tree in the preprocessing phase. Given the labeling, the information of which incident links are swap, if not available, can be easily acquired by having each node exchange the information with its neighbors. The cost of the entire preparation phase is at most $O(|E|)$ messages.

3.2 The F_{sum} problem.

In Problem F_{sum}, the *optimal swap edge* for link $e = (x, p(x))$ is one which minimizes the sum of the distances from all nodes in T_x to the root r, in the new spanning tree $T' = T_{e/e'}$. Note that any swap edge (u, v) solving F_{sum} will also minimize the *average distance* of all the nodes belonging to T_x from the root r, since the size of T_x is the same for all the swap edges for x.

3.2.1 Enabling Information

To solve the F_{sum} problem (known also as *average stretch factor* [3]), we require each node z to possess the following a-priori information: its distance $d(z, r)$ from the root; the number of nodes $n(T_q)$ in T_q for each of its children q; and the sum of the distances of all nodes in T_q to z for each of its children q. If this information is not initially available, it can be easily acquired by the nodes in a pre-processing phase, composed by the following simple convergecast in T, executed only once at the beginning of the algorithm.

Given a subtree T_z and an edge (u, v), with $u \in V(T_z)$ and $v \in V \setminus V(T_z)$, let $sum(T_z, (u, v))$ denote the sum of distances in $T_z \cup (u, v)$ from all nodes of T_z to v.

PRE-PROCESSING(F_{sum})

1. The root r sends down a message to each child q containing a `request-for-sum` and a value $k = w(r, q)$.
2. The message is propagated down to the leaves (adding to k the length of each traversed edge so that each node z in the tree knows its distance $d(z, r)$ to the root).
3. When a leaf l receives the message, it starts a convergecast up to the root to propagate the needed information: it sends to its parent $p(l)$ the values $sum(T_l, (l, p(l))) = w(l, p(l))$ and $n(T_l) = 1$. *****AGGIUNGERE***** Note that, since l is a leaf, $W(T_l) = 0$.*****AGGIUNGERE
4. An internal node z, after receiving the values $W(T_v)$ and $n(T_v)$ from each child v,, will compute the values

$$n(T_z) = \sum_{v \in C(z)} n(T_v) + 1, \text{ and } sum(T_z, (z, p(z))) \quad = \quad W(T_z) + n(T_z) \cdot w(z, p(z)),$$

and will send them to its parent $p(z)$.

The correctness of the pre-processing is proven by the following:

Lemma 1 *Let z be a node in T.*

1. *The total number of nodes in T_z is: $n(T_z) = \sum_{v \in C(z)} n(T_v) + 1$.*

2. The sum of the distances from all nodes in T_z to $p(z)$ is:

$$sum(T_z, (z, p(z))) = W(T_z) + n(T_z) \cdot w(z, p(z)).$$

Proof. Part 1. is obvious. Let us consider Part 2. By definition, $sum(T_z, (z, p(z))) = \sum_{u \in V(T_z)} d(u, p(z))$. Thus,

$$sum(T_z, (z, p(z))) = \sum_{u \in V(T_z)} d(u, z) + \sum_{u \in V(T_z)} w(z, p(z)) = W(T_z) + n(T_z) \cdot w(z, p(z)).$$

■

****AGGIUNGERE****** Note that $W(T_z) = \left(\sum_{v \in C(z)} W(T_v) + w(v, z)\right)$; that is, $W(T_z)$ can be computed with the $W(T_v)$ it receives from its children.*****AGGIUNGERE***** Once all the information is available to the nodes, each node will exchange its local information with the neighbors in G. The number of messages exchanged during the preprocessing phase is then: $O(|E|)$.

3.2.2 The BSE-F_{sum} Algorithm.

Let z be a node in T_x that needs to compute the cost of a candidate swap edge $e' = (z, z')$ for e (see Figure 3). Let $T' = T_{e/e'}$.

Lemma 2 *The sum of the distances in T' from all nodes in T_x to r is:*

$$F_{sum}(T') = sum(T'_z, (z, z')) + n(T_x) \cdot d(z', r) = W(T'_z) + n(T'_z) \cdot w(z, z') + n(T'_z) \cdot d(z', r).$$

Proof. By definition we know that

$$F_{sum}(T') = \sum_{t \in V(T_x)} d_{e/e'}(t, r) = \sum_{t \in V(T_x)} [d_{e/e'}(t, z') + d(z', r)] = \\ = \sum_{t \in V(T_x)} d_{e/e'}(t, z') + \sum_{t \in V(T_x)} d(z', r)$$

which is equal to $sum(T'_z, (z, z')) + n(T_x) \cdot d(z', r)$. Noticing that

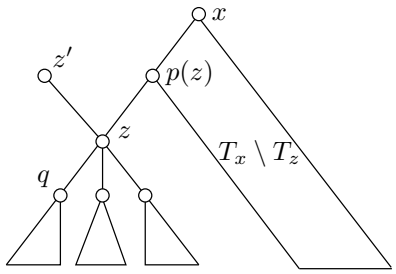

Figure 3: Structure of the subtree T_x with respect to the swap edge (z, z').

$$sum(T'_z, (z, z')) = W(T'_z) + n(T'_z), \cdot w(z, z')$$

*****AGGIUNGERE and that $n(T'_z) = n(T_x)$, *****AGGIUNGERE*****the lemma follows. ■

Notice that $W(T'_z) = W(T_z) + sum(T_x \setminus T_z, (p(z), z))$ and $n(T'_z) = n(T_z) + n(T_x \setminus T_z)$ (see Figure 3). Thus, of the information required to compute the cost of the candidate swap edge (z, z'), there are two components that a node z ($z \neq x$) does not have locally available: $sum(T_x \setminus T_z, (p(z), z))$ and $n(T_x \setminus T_z)$. Only x has all the information immediately available and can locally compute the cost of its candidate swap edges; any other node z in T_x requires this additional information.

To instantiate the generic algorithm BSE for F_{sum} we have to specify what is the *enabling information* to be propagated. On the basis of the above reasoning, the enabling information that any node z has to send down to its child q is composed of: the sum $sum(T_x \setminus T_q, (z, q))$ of the distances from q to the nodes in the subtree $T_x \setminus T_q$; and the number $n(T_x \setminus T_q)$ of nodes in this subtree.

The algorithm for finding the best swap edge for $(x, p(x))$ according to F_{sum} is as follows:

BSE($F_{sum}, (x, p(x))$)

(* Algorithm for node z *)

1. *If* $z = x$
   (a) Compute the cost of each local candidate swap edge: for each $e' = (x, x')$, $F_{sum}(T_{e/e'}) = ***** AGGIUNGERE ***** sum(T'_x, (x, x')) + n(T_x) \cdot d(x', r) = ***** AGGIUNGERE ***** sum(T_x, (x, x')) + n(T_x) \cdot d(x', r)$.
   (b) Select the best candidate
   (c) For each child q: compute the enabling information $sum(T_x \setminus T_q, (x, q))$ and $n(T_x \setminus T_q)$ and send it to q. *It will be shown that x can compute locally this information*
   (d) Wait for the results of *minimum finding* from all children; determine the best swap edge for $(x, p(x))$.
2. *Else* (* $z \neq x$ *) *Receiving enabling info* $< s, n >$ *for* $(x, p(x))$ *from* $p(z)$
   (a) Compute cost of each local candidate swap edge: for each $e' = (z, z')$, $F_{sum}(T_{e/e'}) = *****AGGIUNGERE ******s + W(T_z) + (n + n(T_z)) \cdot (w(z, z') + d(z', r)) = ******* AGGIUNGERE ******s + sum(T_z, (z, z')) + (n + n(T_z)) \cdot d(z', r) + n \cdot w(z, z')$. *It will be shown that this information can be computed locally*
   (b) Select best candidate.
   (c) If I am a leaf: start *minimum finding*.
   (d) If I am not a leaf
      i. for each child q: compute the enabling information $sum(T_x \setminus T_q, (z, q))$ and $n(T_x \setminus T_q) >$ and send it to q.
      ii. participate in *minimum finding* (wait for info from all children, select the best and send to parent).

Lemma 3 *Let $e = (x, p(x))$. Each node $z \in T_x$ can correctly compute:*

1. *the best local swap edge for e,*
2. *the value $sum(T_x \setminus T_q, (z, q))$ for each $q \in C(z)$,*
3. *the value $n(T_v \setminus T_q)$ for each $q \in C(z)$.*

Proof. First observe that, after the preprocessing phase, a node z has available: the labeling $\lambda(y)$ of each of its neighbors y; the distance $d(y, r)$ to r from each of its neighbors y; the sum of the distances $sum(T_q, (q, z))$ of all nodes in T_q to itself and the number of nodes $n(T_q)$ in T_q for each of its children q. The proof of the lemma is by induction on the number of nodes in the path from z to of x.

Basis. $z = x$; i.e., the link to be swapped is $(z, p(z))$. By Lemma 2 we know that, for each swap edge (x, x'), $\sum_{u \in V(T_x)} d_{e/e'}(u, r) = sum(T_x, (x, x')) + n(T_x) \cdot d(x', r)$. Since x is the root of T_x, all the needed information is available at x after the preprocessing phase. Thus, x can locally compute all the swap edges and choose the minimum.

Moreover x can compute, by using only local information, $sum(T_x \setminus T_q, (x,q))$ and $n(T_x \setminus T_q)$ for each $q \in C(x)$.

Induction step. Let the lemma hold for $p(z)$ and consider its child z in T. By Lemma 2 we know that, for each swap edge (z, z'),

$$F_{sum}(T') = sum(T'_z, (z, z')) + n(T_x) \cdot d(z', r).$$

Moreover,

$$sum(T'_z, (z, z')) = \textstyle\sum_{q \in C(z,T')} sum(T'_q, (q, z)) + \left(\sum_{q \in C(z,T')} n(T_q) + 1\right) \cdot w(z, z').$$

Notice that $C(z, T')$ consists of all the children of z in T plus the parent of z in T; i.e., $C(z, T') = C(z) \cup \{p(z)\}$. The values of $sum(T'_q, (q, z))$, and $n(T'_q)$ for $q \in C(z)$ have been computed in the preprocessing phase and are locally available. Since, by induction hypothesis, $p(z)$ has computed the locally best swap edge and the values of $sum(T_x \setminus T_z, (p(z), z))$ and $n(T_x \setminus T_z)$, and since it has sent to z these information, z can now correctly compute the cost of all its local swap edge and choose the minimum. Moreover, it can now compute $sum(T_x \setminus T_q, (z, q))$ and $n(T_x \setminus T_q)$ for each of its children $q \in C(z)$.

■

3.3 The F_{max} and F_{incr} Problems

In Problem F_{max}, the *optimal swap edge* e' for link $e = (x, p(x))$ is any swap edge such that the longest distance of all the nodes in T_x from the root r is minimized in the new spanning tree $T_{e/e'}$; in F_{incr}, it is any swap edge such that the maximum increment in the distance from the nodes in T_x to the root r is minimized in the new spanning tree $T_{e/e'}$.

The algorithm for computing the best swap edges with respect to F_{max} and F_{incr} have the same structure as the one for F_{sum}. What differs is: (i) the information propagated in the preprocessing phase, and (ii) the "enabling information" to be sent to the children during the algorithm.

3.3.1 Enabling Information.

Given a subtree T_z and an edge $(u, v) \in E(G)$, with $u \in V(T_z)$ and $v \in T \setminus T_z$, let $D(T_z, v)$ denote the maximum distance in $T_z \cup (u, v)$ from the nodes in T_z to v.

For solving the F_{max} and the F_{incr} problems we require each node z to possess the following information: its distance $d(z, r)$ from the root, and the maximum distance $D(T_q, z)$ to z from a node in T_q for each $q \in C(z)$. This will be accomplished with a basic convergecast like in the previous section. In this case, Lines 3. and 4. of protocol Pre-processing change as follows:

In the Pre-processing

3. A leaf l sends $D(T_l, p(l)) = w(l, p(l))$ to its parent $p(l)$.

4. An internal node z after receiving from each of its children q, the values $D(T_q, z)$ computes

$$D(T_z, p(z)) = \max\{D(T_q, z)\} + w(z, p(z))$$

and sends it to its parent $p(z)$.

3.3.2 The BSE-F_{max} and BSE-F_{incr} Algorithms

Let z be a node in T_x that needs to compute the cost of a candidate swap edge $e' = (z, z')$ for $e = (x, p(x))$, and let $T' = T_{e/e'}$.

Lemma 4 *The maximum distance $F_{max}(T')$ and the maximum distance increment $F_{incr}(T')$ in T' from a node z in T_x to r are:*

$$F_{max}(T') = \max_{q \in C(z,T')} \{D(T_q, z)\} + w(z, z') + d(z', r)$$

$$F_{incr}(T') = \max_{q \in C(z,T')} \{D(T_q, z) + w(z, z') + d(z', r)\} - d(z, r)$$

Proof. By definition we know that

$$F_{max}(T') = \max_{t \in V(T_x)} d_{e/e'}(t, r) = \max_{t \in V(T_x)} d_{e/e'}(t, z) + w(z, z') + d(z', r).$$

Noticing that $T_x = z \cup \{T_q\}$ (for all $q \in C(z, T')$), and by definition of $D(T_q, z)$, we have that: $\max_{q \in C(z,T')}\{D(T_q, z)\} = \max_{t \in V(T_x)} d_{e/e'}(t, z)$, and the first part of the lemma follows. The second part follows in a similar way, by definition and again by observing that $T_x = z \cup \{T_q\}$ (for all $q \in C(z, T')$).

■

To instantiate the generic algorithm of Section 3.1 for the F_{max} and the F_{incr} objective functions we have now to specify what is the *enabling information* that needs to be propagated so that all the nodes can make their local choice. As it will be shown, in both cases the enabling information that a node z has to send down to its child q is composed of the maximum distance $D(T_x \setminus T_q, q)$ of the nodes in the subtree $T_x \setminus T_q$ to q. The algorithm for node z is then the same as the one for F_{sum}, where the computation of the cost of the local candidate swap edges and the enabling information changes as follows:

CHANGES: BSE-F_{max} AND BSE-F_{incr} ALGORITHMS

1. The cost of each local candidate swap edge is computed as follows:

   - *If* $z = x$, for each $e' = (z, z')$,
     $F_{max}(T_{e/e'}) = \max_{q \in C(x)}\{D(T_q, x)\} + w(x, x') + d(x', r)$,
     $F_{incr}(T_{e/e'}) = \max_{q \in C(x)}\{D(T_q, x) + w(x, x') + d(x', r)\} - d(x, r)$.
   - Else (* $z \neq x$ *) *Receiving enabling info* m *for* $(x, p(x))$:
     $F_{max}(T') = \max\{m, \max_{q \in C(z)}\{D(T_q, z)\}\} + \{w(z, z') + d(z', r)\}$,
     $F_{incr}(T') = \max\{m, \max_{q \in C(z)}\{D(T_q, z)\} + \{w(z, z') + d(z', r)\} - d(z, r)\}$.

2. The enabling information to be sent is $D(T_x \setminus T_q, q)$.

Lemma 5 *Given* $e = (x, p(x))$*, each node* $z \in T_x$ *correctly computes:*

1. *the local best swap edges for* e*,*
2. *the value* $D(T_q, z)$ *for each* $q \in C(z)$*.*

Proof. The values $w(z, z')$ and $d(z', r)$ are locally available because they have been computed in the preprocessing phase. We know that $C(z, T') = C(z) \cup \{p(z)\}$. If $q \in C(z)$, then $max(T_q, z)$ is locally available because it has also been computed in the preprocessing phase. On the other hand, if $q = p(z)$, $max(T_q, z)$ has to be computed; but this is exactly the *enabling information* sent to z by $p(z)$ during the algorithm. ■

3.4 Correctness and Complexity

Let us first examine the correctness of the proposed algorithms.

Lemma 6 *After the execution of Algorithms* BSE(F_{sum})*,* BSE(F_{max})*, and* BSE(F_{incr})*,* x *finds the best swap edge for* $(x, p(x))$ *according to the corresponding objective function.*

Proof. By Lemmas 3 and 5 respectively, every node correctly computes its local best swap edge for e. By the correctness of the minimum finding, the global best swap edge will be communicated to x. ■

Thus, by executing these algorithms for each edge $(x, p(x))$ of T, we have

Theorem 1 *Algorithms* BEST F_{sum}-SWAP*,* BEST F_{max}-SWAP*, and* BEST F_{incr}-SWAP*, correctly solve problems* $\{r, \sum\}$*,* $\{r, \delta\}$*, and* $\{r, \max\}$*, respectively.*

Let us now examine the complexity of the proposed algorithms. Let n^* be the number of edges of the transitive closure of $T_r \setminus \{r\}$; the following simple properties hold by definition:

Property 3
(i) $\sum_{x \in V} |V(T_x)| - 2n + 1 = n^*$
(ii) $\sum_x |A(x)| = n^*$
(iii) $0 \leq n^* \leq \frac{(n-1)(n-2)}{2}$

Theorem 2 *The message and the data complexity of the Algorithms* BEST F_{sum}-SWAP*,* BEST F_{max}-SWAP*, and* BEST F_{incr}-SWAP *is* $O(n^*)$*.*

Proof. The preprocessing phase is executed on T once and its complexity is $O(n)$ messages. During the execution of the algorithm for $(x, p(x))$, the number of messages transmitted is $2|V(T_x)|$; thus, by Property 3, in total we have: $\sum_{x \neq r} 2|V(T_x)| = 2(n^* + 2n - 1) - 2n = 2(n^* + n - 1) = O(n^*)$. Since each message contains only a constant number of data items, the overall data complexity is of the same order of magnitude, i.e., $O(n^*)$. ■

4 Solution Protocols with $O(n)$ Messages

In this section, we present a different approach suitable when long messages are allowed. In this case, the proposed protocols use only $O(n)$ messages without increasing the data complexity.

4.1 Algorithmic Shell and Tools

The basic idea is that each node x simultaneously computes the best swap edges not only for $(x, p(x))$, but also for each $(a, p(a))$, where $a \in A(x)$ is an ancestor of x in T. At a high level, the algorithm consists simply of a *broadcast* phase started by the children of the root, followed by a *convergecast* phase started by the leaves.

BEST F-SWAP-LONG (BSL)

[*Broadcast*]

1. Each child x of the root starts the broadcast by sending to its children a list containing its name, its label, and its distance from the root.

2. Each node y, upon receiving from its parent $p(y)$ the list of ancestors $A(y)$, their labels, and their distances from the root, appends its name, its label, and $d(y, r)$ to the list and sends it to its children.

[*Convergecast*]

1. Each leaf l first computes the best local swap edge for $(l, p(l))$; then for each ancestor $u \in A(l)$, it computes the best local candidate swap for $(u, p(u))$; finally it sends the list of those computed edges to its parent $p(l)$ (if different from r).

2. An internal node y waits until it receives from each of its children the list of the computed swap edges. Based on the received information and on its local swap edges, it computes its best swap edge for $(y, p(y))$; it then computes for each ancestor $v \in A(y)$ the best candidate for $(v, p(v))$; finally, it sends the list of those edges to its parent $p(y)$ (if different from r).

To instantiate this generic algorithmic structure to solve the three different problems, we need to specify (i) how the computation of the best swap edge is performed during the convergecast phase, and (ii) the additional information to be communicated to the ancestors together with the computed swap edges.

For a node $y \in V$, we will denote by $ASL(y)$ the *ancestors' swap edges list* of y; specifically, $ASL(y)$ is a list of records $< node, edge, value, information >$, one for each ancestor $u \in A(y)$ (the *node*), indicating the best swap edge e' (the *edge*) for $e = (u, p(u))$ in T_x, the value (*value*) of the objective function in $T_{e/e'}$, and where *information* a list of parameters to be specified for the particular problem being solved; we shall denote by $ASL(y)[u]$ the record associated to $u \in A(y)$.

Let us describe in details the operations executed by node x. First of all x computes the best swap edge for $(x, p(x))$; then it computes $ASL(x)$. Recall that, during the converge cast, x receives $ASL(y)$ from each child $y \in C(x)$. Since $A(y) = A(x) \cup \{x\}$, this list includes the best swap edge in T_y for $(x, p(x))$, as well as the best swap edge in T_y for each $(u, p(u))$, $u \in A(x)$. Hence,

CONVERGECAST COMPUTATION

(* Algorithm for node x *)

1. To compute the best swap edge for $(x, p(x))$, x selects, among the local swap edges for $(x, p(x))$ and those sent by its children $y \in C(x)$ in $ASL(y)[x]$, the one that minimizes the objective function.

2. To construct $ASL(x)$: for each ancestor $u \in A(x)$, x selects, among the local swap edges for $e = (u, p(u))$ and those sent by the children $y \in C(x)$ in $ASL(y)[u]$, the edge e' that minimizes the objective function; it then sets $ASL(x)[u] =< u, e', F(T_{e/e'})$,*information*$>$.

Finally, note that the information collected in the broadcast phase is sufficient for each node to determine which of its incident edges are swap edges for itself and its ancestors. In fact, during the broadcast, each node x receives the set $A(x)$ of its ancestors (except r) and their labellings; thus (by Property 2), x can determine which of its local edges are swap edges for $(x, p(x))$ and for $(u, p(u)), u \in A(u)$.

4.2 The F_{sum} Problem

To solve the F_{sum} problem, each node z requires some additional information: the weight $W(T_z)$ of the subtree T_z; the number of nodes $n(T_z)$ in such a subtree; and its distance $d_{T_{e/e'}}(z, r)$ from the root for each swap edge e' for $e = (z, p(z))$. This is achieved by having the *information* field of each record of $ASL(x)$ contain the values (i) $W(T_x)$, (ii) $n(T_x)$, and (iii) $d_{T_{(x,p(x))/e_x}}(x, r)$ where e_x denotes the best swap edge for $(x, p(x))$.

With this information, z can easily compute the values $n(T_z)$ and $W(T_z)$ from the values sent by its children z_j in the convergecast. Namely: If z is a leaf, then $n(T_z) = 1$ and $W(T_z) = 0$; otherwise,

1. $n(T_z) = \sum_{z_i \in C(z)} n(T_{z_i}) + 1$

2. $W(T_z) = \sum_{z_i \in C(z)} W(T_{z_i}) + \sum_{z_i \in C(z)} n(T_{z_i}) w(z, z_i)$.

Once this information is available, z can compute the new values of F_{sum} and of $d_{T_{e/e'}}(z, r)$, as indicated by the following Lemma.

Lemma 7 *Consider an edge $e = (z, p(z))$.*

(i) Let $e' = (z, y)$ be a swap edges for e. Then $F_{sum}(T_{e/e'}) = W(T_z) + n(T_z) \cdot (w(z, y) + d_{T_{e/e'}}(y, r)$.

(ii) Let $z_i \in C(z)$ be a child of z, let $e_i = (z_i, z)$, and let the record in in $ASL(z_i)[z]$ sent by z_i to z be $< z, e'_i \neq NIL, F_{sum}(T_{z_i}), \{d_{T'}(z_i, r), W(T_{z_i}), n(T_{z_i})\} >$. Then

(a) $d_{T_{e/e'_i}}(z, r) = w(z, z_i) + d_{T_{e/e'_i}}(z_i, r)$

(b) $F_{sum}(T_{e/e'_i}) = F_{sum}(T_{e_i/e'_i}) + d_{T_{e/e'_i}}(z, r) + \sum_{j=1, j \neq i}^{h} (W(T_{z_j}) + n(T_{z_j})(w(z, z_j) + w(z, z_i) + d_{T_{e/e'_i}}(z_i, r)))$.

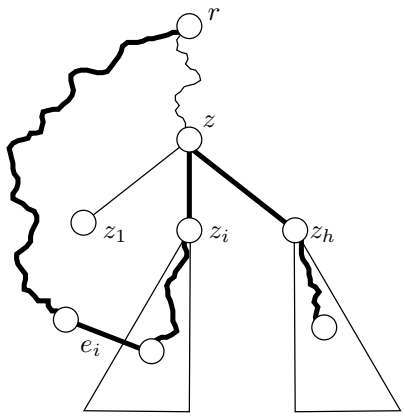

Figure 4: Case (ii) in Lemma 7: the computation of $F_{sum}(T_{(z,p(z))/e'_i})$ via the swap edge e_i. The thick line represents the path to the root via e_i.

Proof. Case (i) follows by Lemma 2. The scenario of Case (ii) is better understood looking at Figure 4. If a swap edge e_i belonging to T_{z_i} is considered, all the nodes in T_{z_i} maintain their distance from the root, hence they contribute to $F_{sum}(T_{e/e'_i})$ only for $F_{sum}(T_{e_i/e'_i})$. Node z contributes for $d_{T_{e/e'_i}}(z, r)$. All the other nodes in $T_{z_j}, 1 \leq j \leq h, j \neq i$, to get the root, follow a path through edges (z_j, z), (z, z_i), and through the swap edges e_i. ■

We finally have:

Lemma 8 *Each node $z \neq r$:*

(i) correctly computes its best swap edge for $(z, p(z))$;

(ii) determines for each ancestor $u \neq r$ the best swap edge for u in T_z.

Proof. The proof is by induction on the height $h(z)$ of the subtree T_z.

Basis. $h(z) = 0$; i.e., z is a leaf. In this case, one component contains only z, while the other contains all the other nodes. In other words, the only possible swap edges are incident on z. Thus, z can correctly compute its best swap edge by computing the value of the distance as stated in point (i) of Lemma 7, thus proving (i). It can also immediately determine the swap edges with respect to all of its ancestors and compute for them the value of the parameters as stated in point (ii) of Lemma 7, and select, for each ancestor, the best swap edge in T_z.

Induction step. Let the theorem hold for all nodes z with $0 \leq h(z) \leq k - 1$; we will now show that it holds for z with $h(z) = k$. By inductive hypothesis, z receives from each child $q \in C(z)$ the best swap edge in T_q for each ancestor of q, including z itself. Hence, based on these lists and on the local swap edges, by Lemma 7, z can correctly determine the optimal swap edge in T_z for itself and for each of its ancestors.

■

4.3 The F_{max} and F_{incr} Problems

In both problems, the value to be minimized is the maximal distance from the nodes in T_z to the root via a swap edge e_i. To achieve this goal, each node z requires some additional information: its distance $d_{T_{e/e'}}(z,r)$ from the root for each swap edge e' for $e=(z,p(z))$, and the maximal distance $D(z,q)$ from the nodes in T_q to z for each child $q \in C(z)$; let us denote by $D(z) = \max_{q \in C(z)} D(z,q)$. We will have the *information* field of each record of $ASL(x)$ contain the values (i) $D(x)$ and (ii) $d_{T_{(x,p(x))/e_x}}(x,r)$ where e_x denotes the best swap edge for $(x,p(x))$.

Let us now show how, using this information, z compute the new values of the parameters. We have:

Lemma 9 *Let $C(z) = \{z_1, ..., z_h\}$. Let $D(z) = D(z,z_k)$, $1 \leq k \leq h$; and let $D_2(z) = max_{j \neq k}(D(z,z_j)$.*

(i) Let $e' = (z,l)$ be a swap edge for $e = (z,p(z))$. Then

(a) $F_{max}(T_{e/e'}) = \max_{q \in C(z,T)}(D(q,z) + w(z,l) + d_{T_{e/e'}}(l,r))$.

(b) $d_{T_{e/e'}}(z,r) = w(z,l) + d(l,r)$, and

(ii) Let $z_i \in C(z)$ be a child of z, let $e_i = (z_i,z)$, and let the record $ASL(z_i)[z]$ sent by z_i to z be $< z, e'_i \neq NIL, F_{max}(T_{e_i/e'_i}), \{d_{T_{e_i/e'_i}}(z_i,r), D(z_i) >$. Then

(a) $d_{T_{e/e'_i}}(z,r) = w(z,z_i) + d_{T_{e/e'_i}}(z_i,r)$

(b) if $i = k$, then $F_{max}(T_{e/e'_i}) = \max\{F_{max}(T_{e_i/e'_i}), D_2(z) + d_{T_{e/e'_i}}(z,r)\}$;
otherwise $F_{max}(T_{e/e'_i}) = \max\{F_{max}(T_{e_i/e'_i}), D(z) + d_{T_{e/e'_i}}(z,r)\}$

Proof. Assume that the children of z have already terminated their computation and transmitted their lists to z. From these values z can compute the maximum distance of a node in T_z, and Case (i) follows immediately. For Case (ii), if the swap edge e_i does not belongs to T_{x_k}, the maximal distance is given by the the largest between $F_{max}(T_{e_i/e'_i})$ and $(D(z) + d_{T_{e/e'_i}}(z,r))$. Otherwise, all the nodes in T_{z_k} maintain their distance from the root; for all the nodes in T_{z_j}, $j \neq k$, called *far* nodes, to get to the root the path goes through edges (z_j,z), (z,z_k), and through the swap edge e'_i. Hence, in this case, to compute the distance of the far nodes we have to consider the maximal distance to z from the descendent of z not belonging to T_{x_k}, i.e. $D_2(z)$. ■

****AGGIUNGERE***** Thus, following an inductive argument similar to the one used to prove Lemma 8, it follows that:******AGGIUNGERE******

Lemma 10 *Each node $x \neq r$:*

(i) correctly computes the best swap edge for $(x, p(x))$ *according to* F_{max}*;*

(ii) determines for each ancestor $u \neq r$ *the best swap edge in* T_x *for* $(u, p(u))$*.*

When the function is F_{incr}, the corresponding protocol is obtained with simple modifications to the one we have described for F_{max}, and the analogous properties follow in the say way.

4.4 Correctness and Complexity

Concerning the correctness of the proposed algorithms, we state the following

Theorem 3 *Algorithms* BEST F_{sum}-Swap-Long*,* BEST F_{max}-Swap-Long*, and* BEST F_{incr}-Swap-Long *correctly solves* $\{r, \sum\}$*,* $\{r, \max\}$*, and* $\{r, \delta\}$*, respectively.*

Proof. By Lemmas 8 and 10, every node correctly computes the best swap edge for itself and all of its ancestors. Therefore, the correctness follows by the correctness of the convergecast procedure. ■

Let us now analyze the number of messages and the amount of data transferred by these new protocols.

Theorem 4 *The message complexity of Algorithms* BEST F_{sum}-Swap-Long*,* BEST F_{max}-Swap-Long*, and* BEST F_{incr}-Swap-Long *is* $O(n)$*; the data complexity is* $O(n_r^*)$

Proof. The basic structure is a broadcast followed by a convergecast, both performed in T; hence the total number of messages is $O(n)$. In the broadcast, every node x (except the root) receives a message that contains a constant amount of data items for each ancestor of x; similarly, in the convergecast, each node x (except the children of the root) sends a constant amount of data items for each ancestor of x. Thus in total, the number of data items transmitted in the algorithms is $O(\sum_x |A(x)|)$; since $\sum_x |A(x)| = n^*$ (Property 3), the theorem follows. ■

5 Conclusions

The computation of the optimal swap edges for a shortest-path tree is a crucial preliminary step to enable the *point-of-failure swap rerouting* technique proposed in the literature. Efficient algorithms for this computations existed only in the *serial* setting. In this paper, we have shown how to efficiently compute the optimal swap edges in a *distributed* setting under several optimization criteria. These protocols are based on a new strategy, different from the serial one. We have also shown how this strategy can be used to develop solution protocols that use only $O(n)$ messages in systems allowing long messages without increasing the total amount of transmitted data items.

An interesting open problem is whether the data complexity can be decreased by employing a different strategy. In other words, the open question is whether or not $\Omega(n^*)$ is a lower bound on the number of data items that must be transferred.

*******AGGIUNGERE ALCUNE REF A PAPER PIU' APPLICATIVI******

References

[1] S. Cicerone, G. Di Stefano, D. Frigioni, and U. Nanni. A fully dynamic algorithm for distributed shortest paths. *Theoretical Computer Science*, 297(1-3):83-102, 2003.

[2] D. Eppstein, Z. Galil, and G.F. Italiano. Dynamic graph algorithms. *In Mikhail J. Atallah (Editor), Algorithms and Theory of Computation Handbook*, Chapter 22, CRC Press, 1997.

[3] A. Di Salvo and G. Proietti. Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor. *Proc. of 10th Colloquium on Structural Information and Communication Complexity* (SIROCCO 2004) 2004.

[4] P. Flocchini, T. Mesa, L. Pagli, G. Prencipe, and N. Santoro. Point-of-failure shortest-path rerouting: computing the optimal swap edges distributively. *IEICE Transactions on Information and Systems*, E89-D (2):700–708, 2006.

[5] A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79:43-59, 1988.

[6] H. Ito, K. Iwama, Y. Okabe, and T. Yoshihiro. Single backup table schemes for shortest-path routing. *Theoretical Computer Science*, 333:347-353, 2004.

[7] H. Mohanty and G.P. Bhattacharjee. A distributed algorithm for edge-disjoint path problem *Proc. of 6th Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), 44-361, 1986.

[8] E. Nardelli, G. Proietti, and P. Widmayer. Finding all the best swaps of a minimum diameter spanning tree under transient edge failures. *Journal of Graph Algorithms and Applications*, 2(1):1–23, 1997.

[9] E. Nardelli, G. Proietti, and P. Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algoritmica*, 35:56–74, 2003.

[10] P. Narvaez, K.Y. Siu, and H.Y. Teng. New dynamic algorithms for shortest path tree computation *IEEE Transactions on Networking*, 8:735–746, 2000.

[11] L.L. Peterson and B.S. Davie. *Computer Networks: A Systems Approach, 3rd Edition.* Morgan Kaufmann, 2003.

[12] G. Proietti. Dynamic maintenance versus swapping: An experimental study on shortest paths trees. *Proc. 3rd Workshop on Algorithm Engineering* (WAE 2000), 207–217, 2000.

[13] R. E. Tarjan. Application of path compression on balanced trees. *Journal of ACM*, 26:690–715, 1979.