Self-Healing Algorithms for Reconfigurable Networks

Iching Boman[1], Jared Saia[1], Chaouki T. Abdallah[1], and Edl Schamiloglu[1]

University of New Mexico, Albuquerque, NM 87131, USA,
Contact Author: Jared Saia, `saia@cs.unm.edu`

Abstract. We present an algorithm to self-heal reconfigurable networks. This algorithm reconfigures the network during an attack to protect two critical invariants. First, it insures that the network remains connected. Second, it insures that no node increases its degree by more than $O(\log n)$. We prove that our algorithm can successfully maintain these invariants even for large networks under massive attack by a computationally unbounded adversary.

1 Motivation and Model

Many modern networks, such as peer-to-peer, are *reconfigurable* in the sense that their topology can change dynamically. We design self-healing algorithms that specifically exploit the reconfigurable nature of these networks. In contrast to many previous results, our algorithms: **provide more protection**: for example, we can guarantee that *all* nodes in the network stay connected instead of just almost all of the nodes; and **conserve resources**: for example, our algorithms devote no resources to defending the network until the time when an attack occurs.

Model: We assume an initially connected network over n nodes where every node knows not only its neighbors in the network but also the neighbors of its neighbors i.e. neighbor-of-neighbor (NoN) information. In particular, for all nodes x,y and z such that x is a neighbor of y and y is a neighbor of z, x knows z. We further assume that there is an *omniscient and computationally unbounded* adversary that is attacking the network. This adversary knows the network topology and our algorithms, and has the ability to delete carefully selected nodes from the network. However, we assume the adversary is constrained in that in any time step it can only delete a small number of nodes from the network. We further assume that after the adversary deletes some node x from the network, that the neighbors of x become aware of this deletion and that they have a small amount of time to react.

When a node x is deleted, we allow the neighbors of x to react to this deletion by adding some set of edges amongst themselves. We constrain these edges to only be between nodes which were previously neighbors of x. This is to ensure that, as much as possible, edges are added which respect locality information in the underlying network. We assume that there is very limited time to react to deletion of x before the adversary deletes another node. Thus, the algorithm for deciding which edges to add between the neighbors of x must be fast and localized.

2 Our Results

Our main results are summarized in the following two theorems. The theorems are proven in the full version of this paper (available at `http://www.cs.unm.edu/~saia/sss06.pdf`). We also include below a centralized version of the *Line* algorithm that is used to prove Theorem 1. We omit the discussion of how to make the algorithm distributed due to space limitations.

Theorem 1. *There exists an algorithm, which we call the* Line *algorithm with the following properties:*
- *Insures that the network is always connected*
- *Increases the degree of any vertex by at most* $\log_2 n$ *where* n *is the number of vertices in the network before attack*
- *Is* locality aware *in the sense that it adds edges only between nodes that have just had a neighbor deleted.*

Theorem 2. *Any locality aware algorithm that insures network connectivity can be forced to increase a node's degree by at least* $\log_3 n$.

2.1 The Line Algorithm

We first define several variables to aid with the description of our algorithm. For a fixed time step we define the following:
- Let $G(V, E)$ be the actual network at the given time step
- Let E' be the edges that have been added by the algorithm up to that time step. (note $E' \subseteq E$).
- Let $G' = (V, E')$. (We note, without proof here, that G' is a forest)
- Let each vertex v have a weight, $w(v)$.
- Let $T(v, x)$ be the tree in $G' - x$ that contains v.
- For vertices v and x, let $W(v, x) = \sum_{v' \in T(v,x)} w(v')$

Line Algorithm:
Initialize each vertex v to have weight $w(v) = 1$ before the first timestep. Then, for each timestep:
- Let G, G' be the graphs at a fixed timestep as defined above, and let x be the node deleted by the adversary at the timestep.
- Let $N^*(x)$ be a maximal set of neighbors of x that are unconnected in $G - x$.

1. Let v_1, v_2 be vertices in $N^*(x)$ with maximal $W(*, x)$ values, i.e. $W(v_1, x) \geq W(v_2, x)$ and $\forall j \in N^*(x)$ s.t. $v_j \neq v_1$, $W(v_2, x) \geq W(v_j, x)$
2. $w(v_1) \leftarrow w(v_1) + w(x)$.
3. Add edges to connect the vertices in $N^*(x)$ in a line, L, such that $v1$ and $v2$ are the endpoints of L.

Fig. 1. The Line Algorithm