University of New Mexico
UNM Digital Repository

Electrical & Computer Engineering Faculty Publications

Engineering Publications

1-1-2008

Responsive Algorithms for Defending Recon gurable Networks

Chaouki T. Abdallah

I-Ching Boman

Jared Saia

Edl Schamiloglu

Follow this and additional works at: https://digitalrepository.unm.edu/ece_fsp

Recommended Citation

Abdallah, Chaouki T.; I-Ching Boman; Jared Saia; and Edl Schamiloglu. "Responsive Algorithms for Defending Recon gurable Networks." (2008). https://digitalrepository.unm.edu/ece_fsp/125

This Article is brought to you for free and open access by the Engineering Publications at UNM Digital Repository. It has been accepted for inclusion in Electrical & Computer Engineering Faculty Publications by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Responsive Algorithms for Defending Reconfigurable Networks

I-Ching Boman * Jared Saia* Edl Schamiloglu Chaouki T. Abdallah

Abstract

We present algorithms to self-heal reconfigurable networks when they are under attack. These algorithms reconfigure the network during attack to protect two critical invariants. First, they insure that the network remains connected. Second, they insure that no node increases its degree by more than $O(\log n)$. We show both theoretically and empirically that our algorithms can successfully maintain these invariants even for large networks under massive attack by a computationally unbounded adversary.

1 Introduction

In this paper, we design *responsive* algorithms that provably protect critical invariants in reconfigurable networks. Our algorithms are responsive in the sense that they adaptively fix the network *while an attack is occuring*. This is in contrast to the standard, non-responsive approach that designs a network that is robust from the very start to the worst conceivable attack. The non-responsive approach assumes that a massive adversarial attack occurs instantaneously and that the network can do nothing while the attack is occurring. In contrast, the responsive approach assumes that the network can respond during the course of an attack.

There are many critical invariants that one would like to maintain while a network is under attack, including: keeping the network connected, keeping node degrees low, keeping the diameter low, and preserving the ability to route quickly. In this paper, we focus only on keeping the network connected and keeping node degrees low. We present responsive algorithms that can maintain these invariants *even when the network is under attack by an omniscient and computationally unbounded adversary.*

Responsive algorithms are useful for networks that are dynamically reconfigurable: networks that can change their topology during an attack. Following are three categories of dynamically reconfigurable networks and the types of attacks that can occur on them.

- *Reconfigurable networks of computational devices:* These include peer-to-peer, ad-hoc and sensor networks. Here the nodes are the devices, there is an edge between two nodes if a communication link is established between the computational devices. An adversary may remove nodes by, for example, launching EMP attacks against the computational devices. After an attack, we want to reconfigure the network to ensure that the computation or data aggregation the network is performing can continue uninterrupted.
- *Infrastructure networks:* An example dynamically reconfigurable infrastructure network is the following air transportation network. Each node represents an airport and there is an edge between two nodes if there are direct flights between the airports these nodes represent. An attacker may remove nodes by incapacitating an airport. After an attack, we want to reconfigure the edges to maintain invariants such as connectivity and low diameter.

*Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386; email: saia@cs.unm.edu. This research was partially supported by NSF grant CCR-0313160 and Sandia University Research Program grant No. 191445.

- *Social networks:* In these networks, each node represents a person in an organization. There is an edge between two nodes if, for example, information can be spread directly between the two people represented by the nodes. An attacker may remove nodes by incapacitating individuals. After an attack, we want to reconfigure the edges of the network to ensure that information can still flow efficiently through the network.

There are two limitations of the responsive approach. First, it can only be used for networks that are dynamically reconfigurable i.e. it must be possible to quickly add extra edges to the network. While the networks mentioned above all have this property, not all networks do. For example, networks where the edges are physical wires are not dynamically reconfigurable. Second, the responsive approach only works in situations where there is time to respond to the attack. In particular, the approach will not work in the case of a massive, instantaneous attack.

When the responsive approach can be used, however, it has several significant advantages over the non-responsive approach. First, we can guarantee better protection than with the non-responsive approach. For example, using the responsive approach, we can guarantee that *all* nodes in the network stay connected even if a constant fraction of the nodes are deleted by an adversary. It is easy to show that this type of guarantee is not possible for non-responsive networks. Second, the responsive approach is more resource efficient than the non-responsive approach. In particular, the non-responsive approach requires the network to be designed from the start to resist the worst conceivable adversarial attack. In contrast, the responsive approach adjusts the amount of resources it commits to network protection based on the severity of the current attack. Finally, the responsive approach works for any type of initial network topology. The non-responsive approach requires that the network topology be carefully designed from the start to be robust to attack.

Our Model: We now describe our model of attack and network response. We assume that the network is initially a connected graph over n nodes. We consider the case of both directed and undirected graphs. We assume that 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 assume that there is an adversary that is attacking the network. This adversary knows the network topology and our algorithms, and it 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 assume that these edges can 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.

Our Results: The main theoretical result of this paper is an responsive algorithm that we call the *Line Algorithm*, whose properties are summarized in the following theorem, which is proven in Section 4

Theorem 1. *The line algorithm has the following properties*

- *It ensures that the network is always connected*
- *It increases the degree of any vertex by at most* $\log_2 n$ *where* n *is the number of vertices in the network before attack*
- *It is* locality aware *in the sense that it adds edges only between nodes that have just had a neighbor deleted.*

This results is quite interesting in lieu of the following theorem which we prove in Section 4.

Theorem 2. *Locality aware algorithms can increase a node's degree by at least* $\log_3 n$.

The main empirical result in this paper is a comparison of the line algorithm and several heuristics we have designed on a family of power-law networks(also known as scale-free). We find that the line algorithm is the most effective algorithm at limiting maximum degree increase. However, many of the heuristics are able to keep the *average* degree increase to $O(\log n)$, where n is the number of nodes in the initial graph. Moreover, these heuristics are likely to run much faster than the line algorithm in a distributed setting.

2 Related Work

There have been various papers that discuss strategies for adding spare capacity and rerouting in anticipation of failures [12, 15, 8, 3, 14, 4]. Médard, Finn, Barry, and Gallager [9] propose constructing redundant trees to make backup routes possible when an edge or node is deleted. Anderson, Balakrishnan, Kaashoek, and Morris [1] modify some existing nodes to be RON (Resilient Overlay Network) nodes to detect failures and reroute accordingly. Some networks have enough redundancy built in so that separate parts of the network can function on their own [5]. In all these cases, the network (nodes and edges) is fixed.

Our algorithm and heuristics, on the other hand, add edges to the network as node failures occur. We guarantee that the network remains connected but do not dictate routing paths.

3 The Line Algorithm

Our main algorithm, which we call the Line algorithm, stems from two observations. First, when a graph loses connectivity, it is broken into connected components. Connecting these components in a tree will reestablish connectivity. Second, a line is a tree with minimal edges, and therefore minimal degrees. Since each connected component will contain at least one vertex that was a neighbor of the deleted vertex, connecting them in a line will limit their degree increase to one or less (lose one edge, gain at most two). In addition, the neighbors that become the two endpoint of the line do not increase their degrees.

We now define several variables to aid with the description of our algorithm. For a fixed time step we define the following:

- The actual network at that step is $G(V, E)$
- Let E' be the edges that have been added by the algorithm up to that time step. (note $E' \subseteq E$).
- Let $G' = (V, E')$. Note that G' is a forest (see Lemma 1)
- Let $N(v, G)$ be the neighbors of vertex v in graph G.
- Let $T(v, x)$ be the tree in $G' - x$ that contains v.
- Let each vertex v have a weight, $w(v)$.
- For vertices v and x, let $W(v, x) = \sum_{v' \in T(v,x)} w(v')$
- For vertex v, let $rem(v) = \sum_{u \in N(v,G')} W(u, v) - \max_{u \in N(v,G')} W(u, v) + w(v)$

The Line algorithm is defined in Figure 1; an example of how it works is given in Figure 2.

4 Theoretical Analysis

We prove the following lemmas about the Line algorithm:

Lemma 3. *The edges added by the algorithm,* E'*, form a forest.*

Proof. At the start of the algorithm, there are no cycles because E' is empty. Each vertex is in its own tree, so G' forms a forest. During any timestep in the algorithm, deleting a vertex x from the forest forms another forest. Since at most one vertex gets new edge(s) per tree in G',

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.

Figure 1: The Line Algorithm

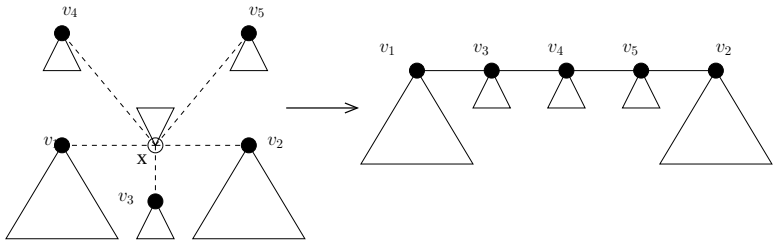

Figure 2: An example of the Line algorithm. Circles are vertices, triangles are the associated trees, and the hollow circle is the vertex to be deleted. The size of the triangles are proportional to the tree weight of the attached vertices.

the new edges do not introduce cycles within a tree. Since the new edges form a line, they also do not introduce cycles between trees. □

Lemma 4. *The algorithm increases a vertex's degree by at most one in each time step. The degree of a vertex only increases if the vertex becomes an interior point in L.*

Proof. The algorithm only connects vertices in a line. In a line, the highest degree vertices are the interior vertices, so interior vertices get the maximum degree gain. Since all vertices in the line were neighbors of x, they all lose one edge through x's deletion. The net edge gain for the interior vertices in the line is therefore $-1+2 = 1$. All other vertices have a net gain of zero. □

Lemma 5. *Whenever the algorithm increases a vertex v's degree, $rem(v)$ at least doubles.*

Proof. Let $rem(v)$ be v's rem value before x is deleted, and $rem'(v)$ be v's rem value after x is deleted and edges of L are added. To prove that $rem'(v) \geq 2rem(v)$, we will show that $rem(v) \leq W(v, x)$ and $rem'(v) \geq 2W(v, x)$.

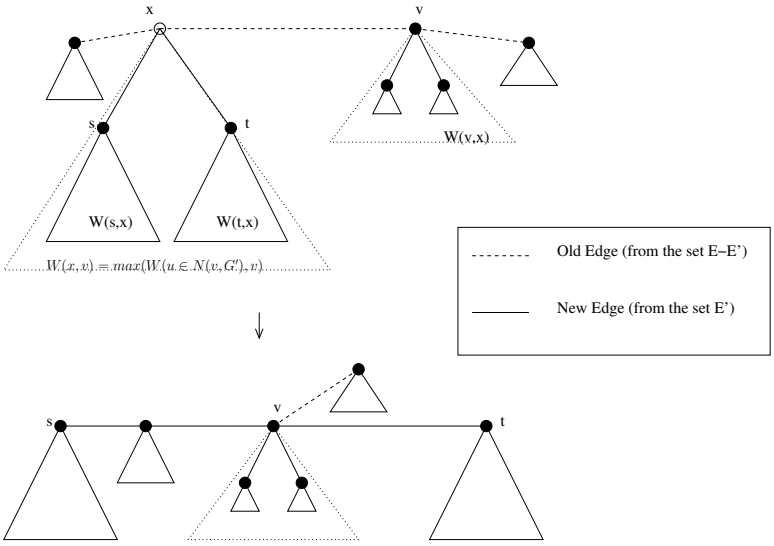

Figure 3: The dashed and solid lines belong to E, the solid lines belong only to E'. The dotted lines define subtrees. For v's degree to increase, x must have two neighbors that have $W(*, x)$ values larger than $W(v, x)$.

Before x is deleted, by definition:

$$rem(v) = \sum_{u \in N(v,G')} W(u,v) - \max_{u \in N(v,G')} W(u,v) + w(v).$$

$$\text{Thus, } \forall u' \in N(v,G'), \quad rem(v) \leq \sum_{u \in N(v,G')} W(u,v) - W(u',v) + w(v).$$

There are two cases:

- **Case 1:** $x \in N(v, G')$

$$\begin{aligned} rem(v) &\leq \sum_{u \in N(v,G')} W(u,v) - W(x,v) + w(v); \\ &= W(v,x). \end{aligned}$$

  Where the second step follows because $W(v, x)$ is the sum of $w(v)$ and all the subtrees of v except for the subtree with x before the deletion of x (see Figure 4).

- **Case 2:** $x \notin N(v, G')$
  x and v must be connected by an edge because $x \in N(v, G)$ in order for v to be eligible to be an interior point in L and gain a degree. Since $x \notin N(v, G')$, x and v must be connect by an edge in $E - E'$. Therefore

$$\begin{aligned} rem(v) &\leq \sum_{u \in N(v,G')} W(u,v) + w(v) \\ &= W(v,x) \end{aligned}$$

  Where the second step follows because no substree of x can be counted in $rem(v)$ because x is not in $N(v, G')$ or in any of the subtrees of v (see Figure 4).

For both cases, in order for v to gain degree, it needs to be an interior point in L, by Lemma 4. Let's call L's endpoints s and t (see Figure 3). In order for s and t to be endpoints of L,

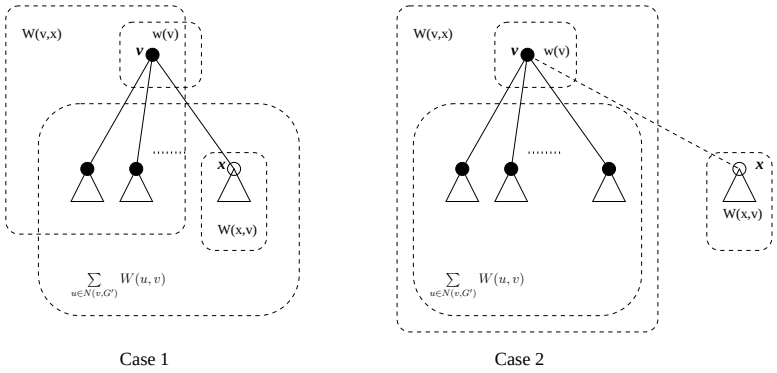

Figure 4: Lemma 3: cases before x is deleted.

either $W(s,x) \geq W(t,x) \geq W(v,x)$ or $W(t,x) \geq W(s,x) \geq W(v,x)$. Without loss of generality, assume

$$W(s,x) \geq W(t,x) \geq W(v,x) \tag{1}$$

Therefore after the deletion of x, the rem value becomes:

$$\begin{aligned} rem'(v) &= W(v,x) + \sum_{u \in N(x,G-x),\ u \neq s} W(u,x) \\ &\geq W(v,x) + W(t,x) \\ &\geq 2W(v,x) \quad \text{(by equation 1).} \end{aligned}$$

Thus, $rem'(v) \geq 2\, rem(v)$. Therefore, the rem value of a vertex at least doubles when that vertex gains a degree. □

Lemma 6. *For every vertex v, $rem(v)$ is non-decreasing over any vertex deletion where v has not been deleted.*

Proof. By Lemma 3, every vertex v in G' belongs to some tree, which we will call T_v. For every T_v in G', let the sum of the weights of all vertices in T_v be called $w(T_v)$.

By definition: $rem(v) = \sum_{u \in N(v,G')} W(u,v) - \max_{u \in N(v,G')} W(u,v) + w(v)$

Therefore: $rem(v) = w(T_v) - \max_{u \in N(v,G')} W(u,v)$

Observe first that $w(T_v)$ cannot decrease because even when no new vertices are added to T_v because the deleted vertex's weight is not "lost", but added to some member of T_v.

Initially, v does not have any subtrees, so $rem(v) = 0$. When one edge is added to v, there is only one subtree, therefore, $rem(v)$ is still zero. When two edges have been added to v, $rem(v)$ increases to some positive number.

For all subsequent steps, since $w(T_v)$ cannot decrease, $rem(v)$ would only decrease if the maximum subtree weight increases more than the tree weight's increase. Since the maximum subtree is a subset of the tree, T_v, any increases or decreases in the maximum subtree is also counted in $w(T_v)$. Therefore the maximum subtree weight cannot increase by more than the tree weight and $rem(v)$ cannot decrease. □

Lemma 7. *For all vertices, v, $rem(v)$ is always no more than n.*

Proof. Since the subtrees of a vertex are disjoint, no vertex is counted twice in a *rem* value. Since the set of subtrees cannot include more than the number of vertices remaining, the *rem* value is always no more than the sum of the weights of all undeleted vertices in G'. Let's call this value W^*.

After initialization, all vertices have weight 1. Since there are n vertices, W^* is n. Therefore the *rem* value is no more than n at initialization. At each step of the algorithm, the weight of the deleted vertex is added to one of the remaining vertices, so W^* remains constant at n. Thus the *rem* value of any vertex remains no more than n. □

We are now ready to prove Theorem 1.

Proof. It is immediate that the line algorithm is locality-aware and that it preserves connectivity. We now show that the degrees of all vertices do not increase too much. Every vertex v starts with $rem(v) = w(v) = 1$. By Lemma 6, v's *rem* value never decreases at any time step of the algorithm. By Lemma 5, whenever the degree of v increases, $rem(v)$ at least doubles. Since $rem(v)$ is always at most n by Lemma 7, the *rem* value of v can double at most $\log_2 n$ times. Therefore, vertex v can increase its degree at most $\log_2 n$ times. Each time, it increases its degree by only one (Lemma 4). Thus, the maximum degree increase is $\log_2 n$.

We next show that the latency of the line algorithm is small. Given a graph with n vertices and m edges, we run depth first search [2], which takes $O(n + m)$ time, to find disconnected neighbors of the deleted vertex. Then we connect at most n vertices in a line. Assuming it takes constant time to add one edge, it takes $O(n)$ to connect n vertices in a line. The total time is still $O(n + m)$. □

We now prove Theorem 2

Proof. We prove this by construction. We observe that whenever G is broken into three connected components, one component is forced to increase its degree. Let μ be the maximum increase in degree experienced by any vertex. The adversary can selectively delete vertices so that no matter how the components are connected, μ has to increase.

We build a graph in the following manner. Starting with a single level tree with three leaves, T_1, we connect the root node of three such trees to a new node to make a new tree, T_2, with the new node as the root. We connect the root node of three T_2 trees to another new node to make a new tree, T_3, and so forth.

For each node, v, let $\Delta(v)$ be the increase in its degree from its initial degree. The adversary first deletes nodes that were root nodes in T_1 trees, forcing a leaf node to become an interior node and gain a degree, and μ to become 1. See Figure 5 Then, the adversary deletes nodes that were root nodes in T_2 trees, forcing one of the $\Delta(v) = 1$ nodes to increase its degree and μ to increase to 2. See Figure 6.

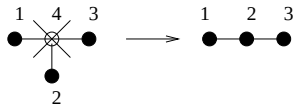

Figure 5: Max degree increase changes from zero to one

With selective trimming of nodes to position the maximal degree increase vertices, this technique of forcing the algorithm to join three nodes with the highest $\Delta(v)$ values will increase μ for each level of the tree. Since there are $\log_3 n$ levels in the tree, $\mu = \log_3 n$ eventually. Through this example, we have shown that a node can be forced to increase its degree by $\log_3 n$ by the adversary. □

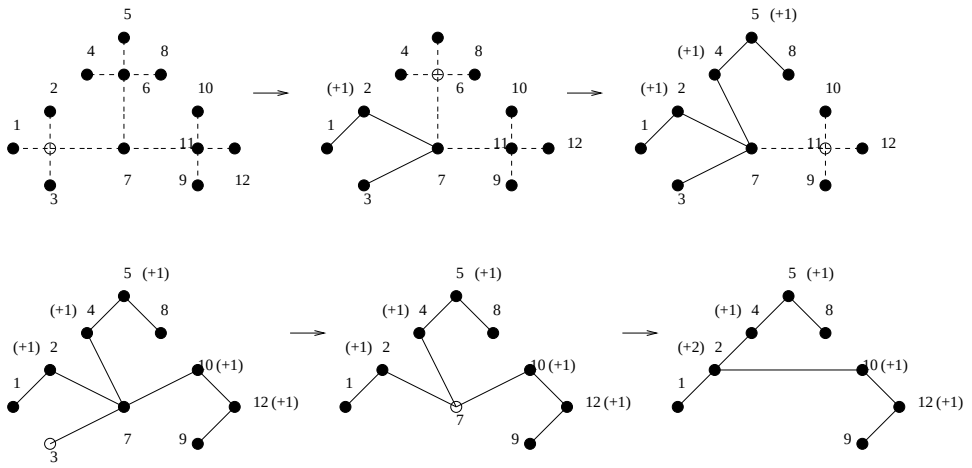

Figure 6: Max degree increase changes from one to two

5 Empirical Results

If the Line algorithm is implemented distributedly, the latency can be large. In order to find the trees in G' after deletion, a distributed depth first search would have to be done. This takes time linear in the diameter of the network. The large diameters of lines increases the latency.

Thus, in this section, we investigate what will happen if all the neighbors of the deleted vertex are connected in a line, without regard to being in the same tree or not. In this case, a depth first search is not be necessary, and the run time is independent of the diameter. We experimented with several criteria for determining endpoints, as we did with the Line algorithm.

5.1 Setup

Sets of power-law graphs with degree distribution $Ck^{-\gamma}$ where $C = 20$ and $\gamma = 3$ were randomly generated using the procedure giving in [13]. Then, an adversary deletes a vertex from the graph using one of two attack schemes and a responsive algorithm is applied. This deletion-response pattern was repeated $n-2$ times. [1] One attack scheme removes the maximum degree vertex from the remaining vertices, the other scheme removes a random vertex. The responsive algorithms are:

- *maxIncNbrs*: All neighbors of the recently deleted vertex are connected in a line unless they already share an edge. Neighbors with the most degree increase(s) are placed at the ends.
- *minIDNbrs*: All neighbors of the recently deleted vertex are connected in a line unless they already share an edge. A neighbor with minimal ID is selected as an endpoint. The minimal ID propagates its ID to all other vertices connected to it. Each vertex starts with a randomly assigned ID in a fixed range.
- *maxIncTrees*: Uses the Line algorithm described in the previous section. One neighbor from each connected component in G' (a tree) is selected. They are connected in a line. The neighbors with the most tree weight(s) are placed at the ends.
- *maxIncComps*: One neighbor from each connected component in G (not G', the forest) is selected. They are connected in a line. Neighbors with the most degree increase(s) are the endpoints.

[1] Source code is available at http://www.cs.unm.edu/homes/iching/SELFHEALING/programs.tar.gz.

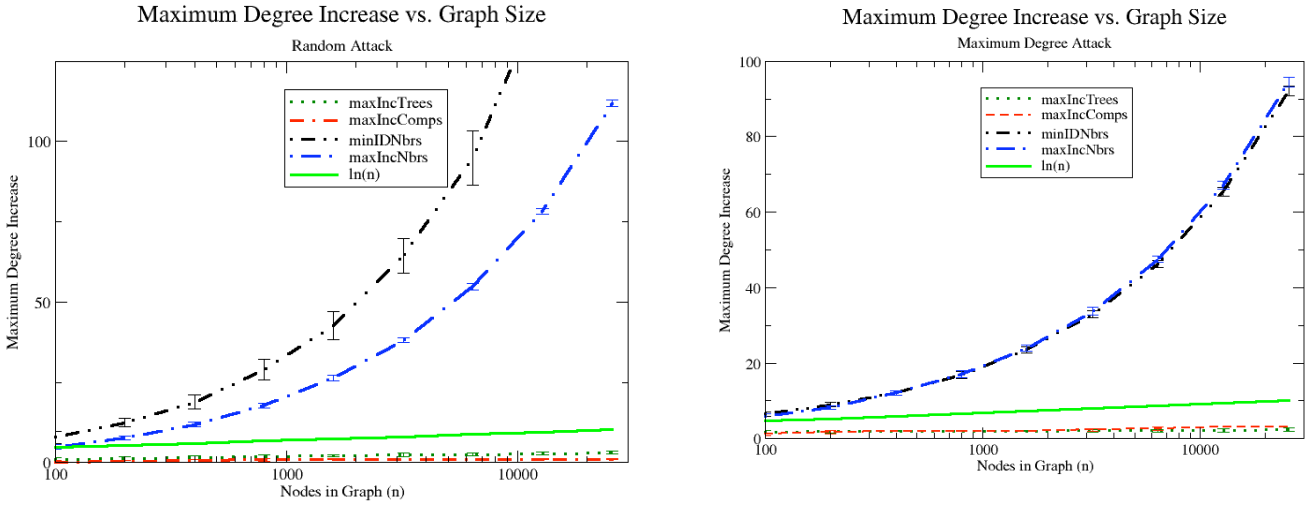

Figure 7: Maximum degree increase. The left plot shows random adversarial attack. The right plot shows adversarial attack that repeatedly deletes the node with maximum degree.

5.2 Results

We measured the following, where each vertex v_j starts with degre $v_j(0)$, and has degree $deg(v_j)$ at the end of a timestep:

- Maximum Degree Increase over time and vertices:
  $\forall$ timestep $1 \leq t \leq n-1$ and vertex j : $\max_t(\max_j(deg(v_j) - v_j(0)))$
- Average Maximum Degree Increase over all time steps:
  $\frac{1}{n}\sum_j(\max_t(deg(v_j) - v_j(0)))$

We found that connecting all neighbors leads to a much greater maximum degree increase than the logarithmic bound with the components-only algorithms (see Figure **??**). The average maximum degree increase, although still worse than the components-only algorithms, was logarithmic for the all-neighbors line algorithm. (see Figure 8). This is not unexpected, because once a node has been deleted, its degree can no longer increase. As more vertices are deleted, although some nodes keep increasing their degree and increasing the maximum degree increase of the graph, more nodes stay constant in their degree increase. An example of the Line algorithm in action is given in Figure 9.

6 Conclusions and Future Work

We have developed an algorithm, called the Line algorithm, that maintains connectivity after node deletions by adding locality aware edges. It guarantees a degree increase of at most $\log_2 n$, which we have shown to asymtotically match a lower bound on locality aware algorithms. We have shown empirically that this algorithm performs well in protecting large-scale networks.

In addition, we compared hueristics that connect all neighbors with heuristics that connect only components in a line. We found empirically that connecting all neighbors, which is likely to run much faster in a distributed setting, limits *average* degree increase to be logarithmic. On the other hand, taking the time to find the connected components that formed upon deletion of a vertex is essential if one has to limit the maximum degree increase.

The following is a description of some ideas for future work:

Modify Randomized Minimum ID Heuristic The heuristic that assigns random IDs to

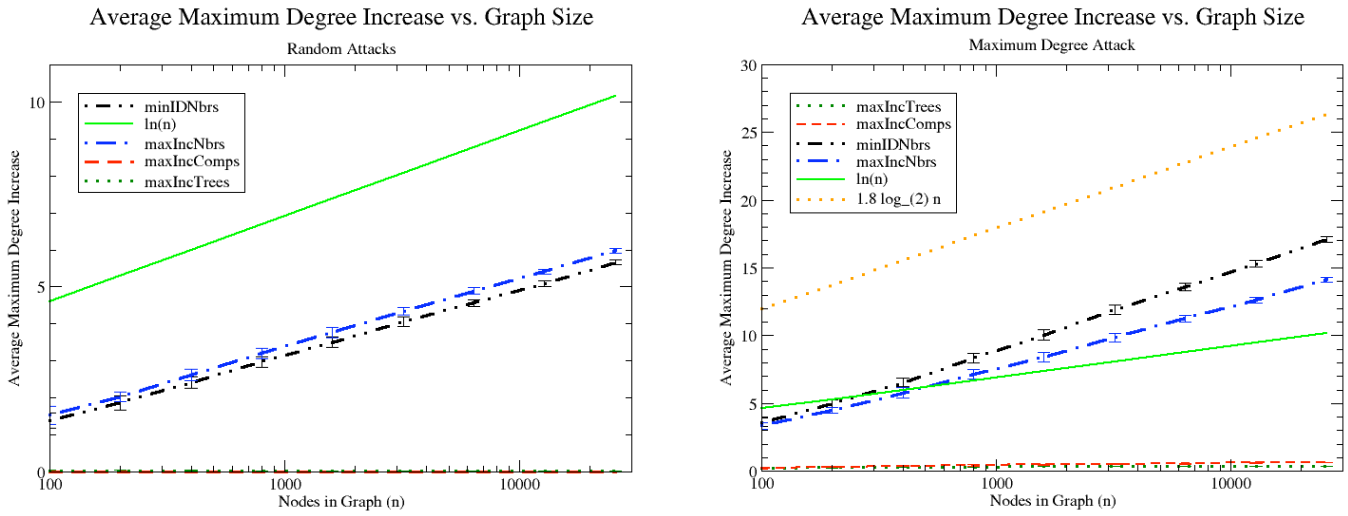

Figure 8: Average Maximum degree increase. The left plot shows random adversarial attack. The right plot shows adversarial attack that repeatedly deletes the node with maximum degree.

vertices and connects them using the minimum ID vertex as an endpoint failed to limit degree increase because the number of ID changes does not measure the number of degree increases. However, perhaps the heuristic can be modified without sacrificing its speed and simplicity. Some other method besides changing IDs may be applied that can measure the number of degree changes.

Decrease Diameter In the worst case, the diameter of the reconnected graph can become rather large, since lines have large diameters. Improving the diameter would be the next logical step. One idea is to connect the components into binary trees instead of lines. The inner nodes of the tree would experience a degree increase of two, as opposed to a degree increase of one experienced by the inner nodes of a line. On the other hand, there can be more than two nodes (leaf nodes) that do not gain any net degree. In a line of more than three nodes, only two can be spared from degree increases.

Add Edge Costs Another variable that deserves further investigation is the cost of adding edges. Currently we only add edges to nodes that are at most two edges apart before deletions. If a cost is associated with each potential edge, and each node is allocated a certain "budget" initially, we can model the problem more accurately.

Investigate Cascading Failures It would be interesting to test how our algorithm handle cascading failures. Each vertex starts with a certain capacity. When a vertex is deleted, some of its "load" (often defined as the number of shortest paths that go through the vertex) is diverted to the remaining vertices. The remaining vertices, in turn, can fail if the extra load exceeds their capacities. Studies have shown that power-law networks are vulnerable to cascading failures ([11] [7]). Motter and Lai had a strategy for preventing cascading failures with intentional removals [10]. Hayashi an Miyazaki had another strategy that also adds edges, called emergent rewirings [6]. It would be interesting to compare how the Line algorithm compares against these algorithms, and perhaps modify the algorithm to also deal with cascading failures.

References

[1] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. *SIGOPS Oper. Syst. Rev.*, 35(5):131–145, 2001.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill, second edition, 2001.

[3] Robert D. Doverspike and Brian Wilson. Comparison of capacity efficiency of dcs network restoration routing techniques. *J. Network Syst. Manage.*, 2(2), 1994.

[4] T. Frisanco. Optimal spare capacity design for various protection switchingmethods in atm networks. In *Communications, 1997. ICC 97 Montreal, 'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, volume 1, pages 293–298, 1997.

[5] Sanjay Goel, Salvatore Belardo, and Laura Iwan. A resilient network that can operate under duress: To support communication between government agencies during crisis situations. *Proceedings of the 37th Hawaii International Conference on System Sciences*, 0-7695-2056-1/04:1–11, 2004.

[6] Yukio Hayashi and Toshiyuki Miyazaki. Emergent rewirings for cascades on correlated networks. cond-mat/0503615, 2005.

[7] Petter Holme and Beom Jun Kim. Vertex overload breakdown in evolving networks. *Physical Review E*, 65:066109, 2002.

[8] Rainer R. Iraschko, M. H. MacGregor, and Wayne D. Grover. Optimal capacity placement for path restoration in stm or atm mesh-survivable networks. *IEEE/ACM Trans. Netw.*, 6(3):325–336, 1998.

[9] Muriel Medard, Steven G. Finn, and Richard A. Barry. Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs. *IEEE/ACM Transactions on Networking*, 7(5):641–652, 1999.

[10] Adilson E Motter. Cascade control and defense in complex networks. *Physical Review Letters*, 93:098701, 2004.

[11] Adilson E Motter and Ying-Cheng Lai. Cascade-based attacks on complex networks. *Physical Review E*, 66:065102, 2002.

[12] Kazutaka Murakami and Hyong S. Kim. Comparative study on restoration schemes of survivable ATM networks. In *INFOCOM (1)*, pages 345–352, 1997.

[13] M E J Newman, S H Strogatz, and D J Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64:026118, 2001.

[14] B. van Caenegem, N. Wauters, and P. Demeester. Spare capacity assignment for different restoration strategies in mesh survivable networks. In *Communications, 1997. ICC 97 Montreal, 'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, volume 1, pages 288–292, 1997.

[15] Yijun Xiong and Lorne G. Mason. Restoration strategies and spare capacity requirements in self-healing atm networks. *IEEE/ACM Trans. Netw.*, 7(1):98–110, 1999.

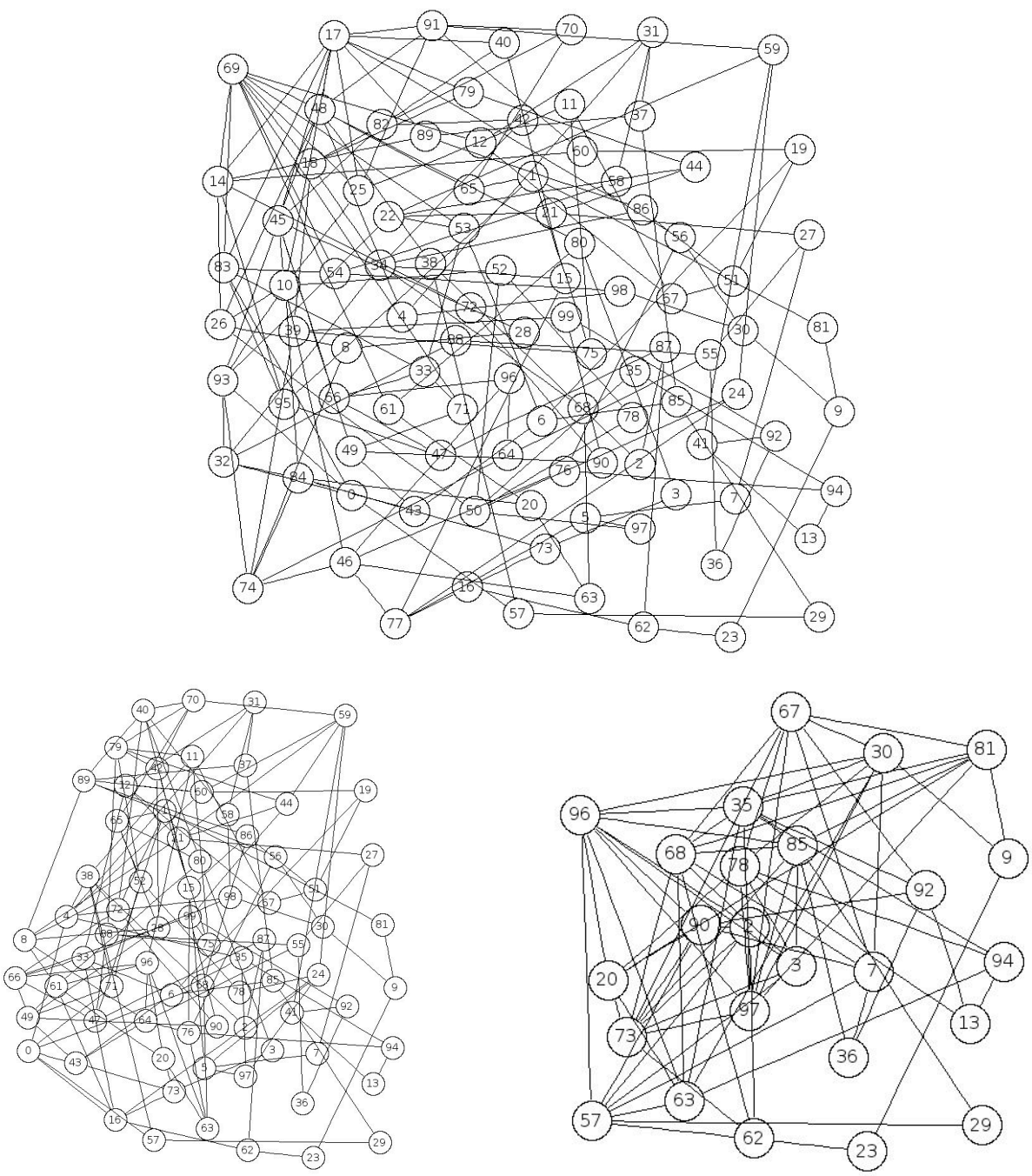

Figure 9: The top image is a power-law network with 100 nodes. The bottom left image is a snapshot of this network after 25 adversarial deletions of maximum degree nodes, healing using the Line algorithm. The bottom right image is a snapshot of the same network after 75 adversarial deletions of maximum degree nodes, healing using the Line algorithm.