Algorithms for Determining Network Robustness

Heath J. LeBlanc
Department of Electrical & Computer Engineering and Computer Science
Ohio Northern University
Ada, OH, USA
h-leblanc@onu.edu

Xenofon Koutsoukos
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN, USA
xenofon.koutsoukos@vanderbilt.edu

ABSTRACT

In this paper, we study algorithms for determining the robustness of a network. Network robustness is a novel graph theoretic property that provides a measure of redundancy of directed edges between all pairs of nonempty, disjoint subsets of nodes in a graph. The robustness of a graph has been shown recently to be useful for characterizing the class of network topologies in which resilient distributed algorithms that use purely local strategies are able to succeed in the presence of adversary nodes. Therefore, network robustness is a critical property of resilient networked systems. While methods have been given to construct robust networks, algorithms for determining the robustness of a given network have not been explored. This paper introduces several algorithms for determining the robustness of a network, and includes centralized, decentralized, and distributed algorithms.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: Fault tolerance

General Terms

Algorithms, Security, Theory

Keywords

Network Robustness; Resilience; Distributed Algorithm; Adversary

1. INTRODUCTION

Network connectivity has long been the key metric in the analysis of fault-tolerant and secure distributed algorithms [3]. This is because (strong) connectivity formalizes the notion of redundant information flow across a network through independent paths. Thus, for algorithms that seek to relay or encode information across multiple hops in the network, connectivity precisely captures the necessary property for analysis [3,5]. More generally, for tasks that require nonlocal information, such as detection of adversary nodes, connectivity is the central property for characterizing the necessary topologies [11, 15]. However, whenever purely local strategies are employed, connectivity is no longer the key metric.

For algorithms that use purely local information, the nodes make decisions and act based on their sensor measurements, calculations, dynamics, and direct interactions with neighbors in the network. No global information is shared or assumed to be known. Instead, information is disseminated within components of the network in an iterative or diffusive manner, rather than being relayed or routed across the network. For these reasons, purely local algorithms are well suited to large-scale dynamic networks. Indeed, purely local strategies are employed in biology and nature [13]; e.g., flocking of birds and schooling of fish are postulated to arise from local interaction rules [13]. From an engineering perspective, examples of algorithms that are explicitly designed to use purely local strategies include iterative function calculation and iterative consensus algorithms [14], as well as gossip-based algorithms [2, 6].

Edge reachability and network robustness are important properties for analyzing algorithms that use purely local strategies [7, 9, 17]. It has been shown that any nontrivially robust network has a directed spanning tree [17]. In fact, 1-robustness[1] is equivalent to the existence of a directed spanning tree [17]. Moreover, for iterative consensus algorithms in a time-invariant network, existence of a directed spanning tree is a necessary and sufficient condition for achieving agreement among the nodes [12]. However, the full utility of edge reachability and network robustness is realized only when considering fault-tolerant and resilient dissemination of information in a network through purely local strategies. This is because edge reachability – which is defined for a nonempty set – captures the requirement that enough nodes inside the set are sufficiently influenced from outside the set. There are two forms of redundancy present in the definition of edge reachability: redundancy of incoming links from outside and redundancy of such nodes with redundant incoming links from outside. This dual redundancy enables resilience against faulty information produced by either a sufficiently small number of neighboring nodes from outside the set or from a sufficiently small number of nodes from within the set. Robustness is a network-wide property that stipulates a lower bound on the edge reachability properties of a sufficiently large number of subsets of nodes. Just as resilient distributed algorithms using nonlocal information utilize the redundancy of independent paths afforded by sufficient connectivity [11, 15], resilient distributed algorithms using purely local information utilize the redundancy of local information present in robust networks [7, 9, 17].

In previous work, the utility of edge reachability and network robustness as metrics for analysis of resilient distributed algorithms that use only local information has been demonstrated [8, 9, 17]. In

[1] See Section 2 for the formal definition of robustness.

HiCoNS'13, April 9–11, 2013, Philadelphia, Pennsylvania, USA.

particular, it is shown in [9] that $(F+1, F+1)$-robustness is the necessary and sufficient condition for achieving resilient asymptotic consensus in a time-invariant network in the presence of up to F malicious adversary nodes that seek to disrupt consensus. Hence, determining the robustness of a network is important for determining whether resilient distributed algorithms can succeed. A growth model for constructing large robust networks from small ones has been given in [17] and extended in [9]. This growth model entails the preferential attachment model of scale-free networks [1], which implies that many scale-free networks are nontrivially robust. In [16], it has been shown that the threshold function of random graphs coincide for robustness and connectivity, which implies that random graphs with high connectivity are also highly robust. These results imply that many complex networks are in fact robust. However, as of yet, no algorithms have been given for determining the robustness of a network.

In this paper we propose algorithms to determine the robustness of a given network. In particular, two centralized algorithms are introduced. The first algorithm checks for a given amount of robustness and the second one determines the robustness of any network, regardless of its connectedness properties. These algorithms assume the topology of the network is given as input to the algorithm (encoded by the adjacency matrix). A decentralized algorithm is proposed that enables the individual nodes of an undirected, connected network to compute the robustness of the network in a decentralized manner by broadcasting information about their neighborhood in order to locally reconstruct the network topology. The centralized algorithm is then used at each node to determine the overall robustness. A modification to this decentralized algorithm is proposed in which each individual node only checks the edge reachability conditions for subsets in which it is *not* included, thus resulting in a truly distributed algorithm. For these algorithms, we analyze their complexity and examine the improvement gained by the distributed algorithm.

The rest of the paper is organized as follows. Section 2 reviews the definition of edge reachability and network robustness, and recalls several useful properties of robust networks. Section 3 describes the nature of the problem of determining network robustness and proposes centralized, decentralized, and distributed algorithms to do so. Section 4 summarizes the work and provides directions for future work.

2. NETWORK ROBUSTNESS

Network robustness is a property of a network that formalizes the notion of sufficient redundancy of directed information flow between subsets of nodes in the network. Therefore, this property is generally defined for a finite, simple directed graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$. Without loss of generality, $\mathcal{V} = \{1, 2, \ldots, n\}$ is the set of nodes in the network and each directed edge $(i, j) \in \mathcal{E}$ indicates that node i is capable of transmitting information to node j. In this case, node i is an *in-neighbor* of node j and node j is an *out-neighbor* of node i. The set of in-neighbors of node i is denoted $\mathcal{N}_i^{\text{in}} = \{j \in \mathcal{V} \colon (j, i) \in \mathcal{E}\}$ and the *in-degree* of node i is denoted $d_i^{\text{in}} = |\mathcal{N}_i^{\text{in}}|$. In order to define network robustness, we require the following definition [7, 9].

DEFINITION 1 ((r, s)-EDGE REACHABLE SET). *Given a nontrivial digraph $\mathcal{D}$ and a nonempty subset of nodes $\mathcal{S}$, we say that $\mathcal{S}$ is an (r, s)-**edge reachable set** if there are at least s nodes in $\mathcal{S}$ with at least r in-neighbors outside of $\mathcal{S}$, where $r, s \in \mathbb{Z}_{\geq 0}$; i.e., given $\mathcal{X}_{\mathcal{S}}^r = \{i \in \mathcal{S} \colon |\mathcal{N}_i^{in} \setminus \mathcal{S}| \geq r\}$, then $|\mathcal{X}_{\mathcal{S}}^r| \geq s$.*

A general illustration of an (r, s)-edge reachable set of nodes is shown in Figure 1. The parameter s in the definition of (r, s)-edge reachability quantifies a lower bound on the number of nodes in the set with at least r in-neighbors outside $\mathcal{S}$. Hence, the parameter r quantifies the redundancy of directed edges coming from outside and the parameter s quantifies the redundancy of nodes with sufficient outside influence.

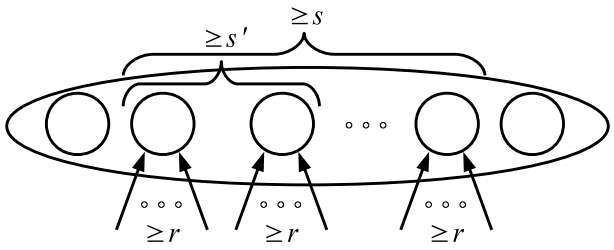

Figure 1: Illustration of an (r, s)-edge reachable set of nodes.

Observe that, in general, a set is (r, s')-edge reachable, for $s' \leq s$, if it is (r, s)-edge reachable. At one extreme, whenever there are no nodes in $\mathcal{S}$ with at least r in-neighbors outside of $\mathcal{S}$, then $\mathcal{S}$ is only $(r, 0)$-edge reachable. At the other extreme, $\mathcal{S}$ can be at most $(r, |\mathcal{S}|)$-edge reachable. Edge reachability is used to define the global property of robustness [7, 9].

DEFINITION 2 ((r, s)-ROBUSTNESS). *A nonempty, nontrivial digraph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ on n nodes ($n \geq 2$) is (r, s)-**robust**, for nonnegative integers $r \in \mathbb{Z}_{\geq 0}$, $1 \leq s \leq n$, if for every pair of nonempty, disjoint subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ of $\mathcal{V}$ at least one of the following holds (recall $\mathcal{X}_{\mathcal{S}_k}^r = \{i \in \mathcal{S}_k \colon |\mathcal{N}_i^{in} \setminus \mathcal{S}_k| \geq r\}$ for $k \in \{1, 2\}$):*

(i) $|\mathcal{X}_{\mathcal{S}_1}^r| = |\mathcal{S}_1|$;

(ii) $|\mathcal{X}_{\mathcal{S}_2}^r| = |\mathcal{S}_2|$;

(iii) $|\mathcal{X}_{\mathcal{S}_1}^r| + |\mathcal{X}_{\mathcal{S}_2}^r| \geq s$.

By convention, if $\mathcal{D}$ is empty or trivial ($n \leq 1$), then $\mathcal{D}$ is (0,1)-robust. If $\mathcal{D}$ is trivial, $\mathcal{D}$ is also (1,1)-robust.[2]

Note that an $(r, 1)$-edge reachable set is abbreviated as r-edge reachable, and an $(r, 1)$-robust digraph is abbreviated as r-robust.

We adopt a total order for (r, s)-robustness that gives precedence to the r parameter in determining the relative robustness of a network. The maximal s in (r, s)-robustness is then used for ordering the robustness of two r-robust digraphs with the same value of r. The reason for adopting this convention is twofold. There are digraphs in which the parameter s may take on any value, and in such cases s loses its precise meaning. On the other hand, there are properties that show the utility of the parameter r in characterizing the robustness of a given digraph [8]. For example, the following properties provide the upper bound $r \leq \min\{\delta^{\text{in}}(\mathcal{D}), \lceil n/2 \rceil\}$ on the value of r for any digraph on n nodes, where $\delta^{\text{in}}(\mathcal{D})$ is the minimum in-degree of $\mathcal{D}$.

PROPERTY 1 (MAXIMUM ROBUSTNESS [8]). *No digraph $\mathcal{D}$ on n nodes is $(\lceil n/2 \rceil + 1)$-robust. Conversely, the complete digraph, denoted $K_n = (\mathcal{V}, \mathcal{E}_{K_n})$, with $\mathcal{E}_{K_n} = \{(i, j) \in \mathcal{V} \times \mathcal{V} \colon i \neq j\}$, is $(\lceil n/2 \rceil, s)$-robust, for $1 \leq s \leq n$. Furthermore, whenever $n > 1$ is odd, K_n is the only digraph on n nodes that is $(\lceil n/2 \rceil, s)$-robust with $s \geq \lfloor n/2 \rfloor$.*

[2]The trivial graph is defined to be both (0,1)-robust and (1,1)-robust for consistency with properties of robust networks for $n > 1$ [8].

PROPERTY 2 (MINIMUM IN-DEGREE [8]). *Given an (r,s)-robust digraph $\mathcal{D}=(\mathcal{V},\mathcal{E})$, with $0 \leq r \leq \lceil n/2 \rceil$ and $1 \leq s \leq n$, the minimum in-degree of $\mathcal{D}$, $\delta^{in}(\mathcal{D})$, is at least*

$$\delta^{in}(\mathcal{D}) \geq \begin{cases} r+s-1 & \text{if } s < r; \\ 2r-2 & \text{if } s \geq r. \end{cases}$$

3. ALGORITHMS

3.1 Centralized Algorithms

In this section, we present centralized algorithms for checking and determining robustness. The direct manner to check a digraph to determine whether it is (r,s)-robust is a combinatorial problem. Because the sets in each pair considered are required to be nonempty and disjoint, the total number of pairs $R(n)$ that must be checked is

$$R(n) = \sum_{k=2}^{n} \binom{n}{k} \left(2^{k-1}-1\right), \tag{1}$$

where

- $n = |\mathcal{V}|$ is the number of nodes;
- each $k = 2,3,\ldots,n$ in the sum is the size of the k-subsets of $\mathcal{V} = \{1,2,\ldots,n\}$. Each k-subset of $\mathcal{V}$ is partitioned into exactly two nonempty parts, $\mathcal{S}_1$ and $\mathcal{S}_2$;
- $\binom{n}{k}$ is the number of k-subsets of $\{1,2,\ldots,n\}$;
- $2^{k-1}-1 = S(k,2)$ is a Stirling number of the second kind, and is the number of ways to partition a k-set into exactly two nonempty unlabeled subsets (swapping the labels $\mathcal{S}_1$ and $\mathcal{S}_2$ results in the same pair)[3].

The form of (1) implies an algorithm for checking (r,s)-robustness of a digraph, *CheckRobustness*, which is shown in Algorithm 3.1. *CheckRobustness* takes as input values of r and s[4] and the adjacency matrix of the digraph, $A(\mathcal{D}) = [a_{ij}]$, which is defined by [4]

$$a_{ij} = \begin{cases} 1 & (i,j) \in E; \\ 0 & (i,j) \notin E. \end{cases}$$

CheckRobustness returns a Boolean variable indicating whether the digraph is (r,s)-robust along with a pair of sets with which the conditions fail. If the digraph is (r,s)-robust, the sets returned are empty sets. The algorithm iterates through all possible pairs of nonempty disjoint subsets, $\mathcal{S}_1, \mathcal{S}_2 \subset \mathcal{V}$, and checks conditions (i)-(iii) of Definition 2 using the adjacency matrix. To improve the performance on digraphs that fail the test, the algorithm returns the first pair that fails.

[3]The quantity may be argued directly by noticing that for each of the k elements, we have two choices: $\mathcal{S}_1$ or $\mathcal{S}_2$. But, we have to subtract the two sequences of choices resulting in $\mathcal{S}_1 = \emptyset$ or $\mathcal{S}_2 = \emptyset$. Finally, because the labels on the sets $\mathcal{S}_1$ and $\mathcal{S}_2$ are unimportant to the uniqueness of the nonempty partition, we divide by 2.

[4]Since all digraphs are 0-robust, and by Property 1 no digraph is r-robust with $r > \lceil n/2 \rceil$, it follows that one should only check $1 \leq r \leq \lceil n/2 \rceil$, $1 \leq s \leq n$.

Algorithm 3.1: CHECKROBUSTNESS$(A(\mathcal{D}), r, s)$

```
procedure RobustHolds(A(D), S1, S2, r, s)
  isRSRobust ← false , s_{r,1} ← 0, s_{r,2} ← 0
  for each k ∈ {1, 2}
  { for each i ∈ S_k
    { if Σ_{j ∈ N_i^in \ S_k} a_ji ≥ r
      then s_{r,k} ← s_{r,k} + 1
  if (s_{r,1} == |S_1|) or (s_{r,2} == |S_2|) or (s_{r,1} + s_{r,2} ≥ s)
    then isRSRobust ← true
  return (isRSRobust)

main
  isRSRobust ← true
  for k ← 2 to n
  { for each K_i ∈ K_k (i = 1, 2, ..., (n choose k))
    comment: K_k is the set of (n choose k) unique k-subsets of V
    { for each P_j ∈ P_{K_i} (j = 1, 2, ..., 2^{k-1} − 1)
      comment: P_{K_i} is the set of partitions of K_i
               with exactly two nonempty parts
      { comment: P_j = {S_1, S_2}
        if not RobustHolds(A(D), S_1, S_2, r, s)
          then { isRSRobust ← false
                 return (isRSRobust, S_1, S_2)
  S_1 ← ∅, S_2 ← ∅
  return (isRSRobust, S_1, S_2)
```

The second algorithm, called *DetermineRobustness* (and given in Algorithm 3.2), determines (r,s)-robustness of *any* digraph, regardless of the number of components of the digraph. To do this, it requires the adjacency matrix $A(\mathcal{D})$ as input. *DetermineRobustness* first initializes r and s to the maximum values possible for any digraph on n nodes (see Properties 1 and 2). Then, as in *CheckRobustness*, *DetermineRobustness* iterates through all possible pairs of nonempty disjoint subsets, $\mathcal{S}_1, \mathcal{S}_2 \subset \mathcal{V}$, and checks conditions (i)-(iii) of Definition 2. In this case, instead of terminating upon a failing condition, the value of s is first decremented. Once all values of s are checked for the given value of r, the algorithm decrements r and restores s to its maximal value of n. If the digraph has no directed spanning tree (i.e., if it is not 1-robust), then *DetermineRobustness* returns $(r=0, s=n)$. Using this approach, *DetermineRobustness* returns the maximal values r and s such that $\mathcal{D}$ is (r,s)-robust.[5]

It is clear from the form of $R(n)$ in (1) that these algorithms are not efficient. In fact, we show next that the algorithms are exponential in the square root of the size of the input (in this case, the adjacency matrix, which has size n^2). To analyze the complexity of the algorithms, we recall the following definition.

DEFINITION 3. *Given $f, g\colon \mathbb{R} \to \mathbb{R}$, then $f \in \mathcal{O}(g(x))$ if there exists $c \in \mathbb{R}_{>0}$ and $x_0 \in \mathbb{R}$ such that $|f(x)| \leq c|g(x)|$ for all $x \geq x_0$.*

We define the worst-case complexity of an algorithm as the maximal number of steps $T(m)$ required to complete the algorithm whenever the input is of size m. We say the worst-case complexity of an algorithm is $\mathcal{O}(g(m))$ if $T(m) \in \mathcal{O}(g(m))$.

[5]Recall from Section 2 that the total order on robustness compares the value of r first, and then compares the value of s for networks with the same r.

Algorithm 3.2: DETERMINEROBUSTNESS($A(\mathcal{D})$)

$r \leftarrow \min\{\delta^{\text{in}}(\mathcal{D}), \lceil n/2 \rceil\}$
comment: $\delta^{\text{in}}(\mathcal{D})$ is the minimum in-degree of $\mathcal{D}$
$s \leftarrow n$
for $k \leftarrow 2$ **to** n
  for each $K_i \in \mathcal{K}_k$ $(i = 1, 2, \ldots, \binom{n}{k})$
  comment: $\mathcal{K}_k$ is the set of $\binom{n}{k}$ unique k-subsets of $\mathcal{V}$
    for each $P_j \in \mathcal{P}_{K_i}$ $(j = 1, 2, \ldots, 2^{k-1} - 1)$
    comment: $\mathcal{P}_{K_i}$ is the set of partitions of K_i with exactly two nonempty parts
      comment: $P_j = \{\mathcal{S}_1, \mathcal{S}_2\}$
      $isRSRobust \leftarrow$ ROBUSTHOLDS$(A(\mathcal{D}), \mathcal{S}_1, \mathcal{S}_2, r, s)$
      if $(isRSRobust ==$ **false** $)$ **and** $(s > 0)$
        then $s \leftarrow s - 1$
      while $(isRSRobust ==$ **false** $)$ **and** $(r > 0)$
        do
          while $(isRSRobust ==$ **false** $)$ **and** $(s > 0)$
            do
              $isRSRobust \leftarrow$ ROBUSTHOLDS$(A(\mathcal{D}), \mathcal{S}_1, \mathcal{S}_2, r, s)$
              if not $isRSRobust$
                then $s \leftarrow s - 1$
          if $(isRSRobust ==$ **false** $)$
            then
              $r \leftarrow r - 1$
              $s \leftarrow n$
      if $r == 0$
        then return (r, s)
return (r, s)

PROPOSITION 1. *Algorithms 3.1 and 3.2 have worst-case complexity $\mathcal{O}(m3^{\sqrt{m}})$ where $m = n^2$ is the size of the input (the adjacency matrix).*

PROOF. The procedure *RobustHolds* requires $\mathcal{O}(n^2)$ steps because $\mathcal{S}_k$ contains $\mathcal{O}(n)$ elements and the summation in the if-statement requires $\mathcal{O}(n)$ steps. In worst-case, there will be $R(n)$ calls to *RobustHolds* in Algorithm 3.1 and $R(n) + g(n)$ in Algorithm 3.2, where $g(n) \in \mathcal{O}(n^2)$ (since in Algorithm 3.2 there will be at most an additional $(\lceil n/2 \rceil - r)(n) + n - s$ calls to *RobustHolds* in an (r, s)-robust digraph caused by decrementing the values of r and s from their initial values). Therefore, in either case, there are $\mathcal{O}(R(n))$ calls to *RobustHolds*, and hence, $\mathcal{O}(n^2 R(n))$ steps in the worst case. Finally, to bound $R(n)$, we use the Binomial Theorem to obtain

$$\begin{aligned} R(n) &= \sum_{k=2}^{n} \binom{n}{k} (2^{k-1} - 1) \\ &\leq \sum_{k=2}^{n} \binom{n}{k} 2^k 1^{n-k} \\ &\leq \sum_{k=1}^{n} \binom{n}{k} 2^k 1^{n-k} \\ &\leq 3^n. \end{aligned}$$

Therefore, Algorithms 3.1 and 3.2 are $\mathcal{O}(m3^{\sqrt{m}})$, where $m = n^2$. □

The complexity of Algorithms 3.1 and 3.2 are typical of any algorithm that determines the robustness of a network. This is because determining robustness is an NP-hard problem [16].

3.2 Network Model

The remaining algorithms operate in a decentralized manner in a time-invariant network. To model the network, we consider the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, ..., n\}$ is the *node set* and $\mathcal{E} \subset \binom{\mathcal{V}}{2}$ is the *edge set*. Each edge $\{i, j\} \in \mathcal{E}$ indicates that nodes i and j can exchange information. Each node $i \in \mathcal{V}$ is aware of its own identifier $i \in \mathcal{V}$ and its neighbor set $\mathcal{N}_i$. The diameter of the graph is denoted $diam$. Additionally, all nodes are normal; i.e., $\mathcal{V} = \mathcal{N}$. The network is assumed to be connected and fully synchronous with reliable communication. The execution of the distributed algorithm in the synchronous network progresses in rounds mapped to the nonnegative integers, $\mathbb{Z}_{\geq 0}$. We assume multiple messages may be sent in a given round and that messages may be of arbitrary size.

3.3 A Decentralized Algorithm

In this section, we present a decentralized algorithm for determining the robustness of a connected network. The main idea of the algorithm is for the nodes to share information about their neighborhood in such a way so that each node obtains the topological information about the network. Once this information is obtained, the centralized algorithm, *DetermineRobustness*, may be used to determine the robustness. The decentralized algorithm, *DecentralDetermineRobust* is shown in Algorithm 3.3. The algorithm uses several procedures: *LeaderElectBFSTree*, *InitiateConvergecast*, *ParticipateConvergecast*, *Broadcast*, *ParticipateBroadcast*, and *DetermineRobustness*.

The first procedure is *LeaderElectBFSTree*. *LeaderElectBFSTree* takes as input the node's ID and its neighbor set, and outputs a *leader* ID, a *parent* node, and a set of *children* nodes. *LeaderElectBFSTree* elects a leader in the network using parallel executions of Breadth-First Searches (BFSs) [10]. Each node initiates a modification of the *SynchBFS* algorithm of [10]. To do this, a calculation of the maximum ID is bootstrapped to the Breadth-First Search (BFS) tree construction. The node with the maximum ID is declared the leader, and the BFS tree with the leader as the root node is the BFS tree used in the subsequent convergecast and broadcast procedures.

Algorithm 3.3: DECENTRALDETERMINEROBUST(ID$_i$, $\mathcal{N}_i$)

$(Leader, Par, Chldrn) \leftarrow$ LEADERELECTBFSTREE(ID$_i$, $\mathcal{N}_i$)
if $(Leader ==$ ID$_i)$
  then
    $A(\mathcal{G}) \leftarrow$ INITIATECONVERGECAST(ID$_i$, $Chldrn$)
    BROADCAST$(A(\mathcal{G}), Chldrn)$
  else
    PARTICIPATECONVERGECAST(ID$_i$, $\mathcal{N}_i$, Par, $Chldrn$)
    $A(\mathcal{G}) \leftarrow$ PARTICIPATEBROADCAST$(Par, Chldrn)$
$(r, s) \leftarrow$ DETERMINEROBUSTNESS$(A(\mathcal{G}))$
return (r, s)

In *LeaderElectBFSTree* all variables are associated to the initiating node's ID because n parallel executions run simultaneously. There are three types of messages involved in the BFS tree construction: *search*, *respond*, and *propagate* messages. The *search* message, with node i as the initiating node, contains the sending node's ID (initially i), the maximum ID seen so far (initially i), and the initiating node's ID (also i in the first round). Each node (other than i) is initially unmarked. Whenever an unmarked node receives a *search* message (or possibly multiple *search* messages from different neighbors), it becomes marked. The receiving node sets the maximum ID variable as the max of the received maximum

IDs and chooses one of the senders as its *parent*. It then sends a *respond* message to each neighbor from which it received a *search* message. The *respond* message contains its ID, a binary variable indicating whether the node was selected as *parent*, and the ID of the initiating node i. The next round, the marked node sends a *search* message to all of its neighbors to continue the construction of the BFS tree. Whenever a marked node receives a *respond* message, it checks to see if it is selected as the node's parent, and if so, it adds the node's ID to its *children* list. After each node sends its *search* message, it waits to receive all *respond* messages from neighbors. Once it receives all of its *respond* messages, it knows whether it is a leaf node in the BFS tree (i.e., if none of its neighbors selects it as *parent*). If a node is *not* a leaf node, it waits to receive *propagate* messages from all of its *children*. If it is a leaf node, it sends a *propagate* message to its *parent*, which contains the maximum ID it has seen. Once a non-leaf node receives *propagate* messages from all of its *children*, it takes the max of the maximum IDs and sends a *propagate* messages to its *parent*. Eventually, the initiating node i receives *propagate* messages from all of its *children* and then knows the maximum ID in the network. The node with the maximum ID asserts itself as the *leader*. The *parent* and *children* variables returned by *LeaderElectBFSTree* correspond to the node's *parent* and *children* in the BFS tree with the *leader* as the initiating node. If i is the leader, it selects itself as the parent (or the null symbol). *LeaderElectBFSTree* requires $\mathcal{O}(diam)$ rounds and $\mathcal{O}(diam|\mathcal{E}|)$ messages for each of the n parallel executions [10]. Hence, in total, *LeaderElectBFSTree* requires $\mathcal{O}(diam)$ rounds and $\mathcal{O}(n \times diam|\mathcal{E}|)$ messages.

Once *LeaderElectBFSTree* terminates, the leader node is determined and its BFS tree is constructed, which provides an efficient mechanism for convergecast and broadcast. In *DecentralDetermineRobust*, if node i is the leader, it initiates a convergecast using *InitiateConvergecast*. If node i is not the leader it participates in the convergecast using *ParticipateConvergecast*. *InitiateConvergecast* takes as input the leader's own node ID and the *children* list (which is just the neighbor set of the leader node). *ParticipateConvergecast* takes as input the node's own ID and its neighbor set, as well as the *parent* and *children* determined by *LeaderElectBFSTree*. In *InitiateConvergecast* and *ParticipateConvergecast*, there are two types of messages: *downstream* and *upstream* messages. *Downstream* messages are sent in the direction of leaf nodes, and *upstream* messages are sent in the direction of the *leader* (root) node. The leader node starts the convergecast by sending a *downstream* message containing its node ID and neighbor set to its children of the BFS tree constructed by *LeaderElectBFSTree*. Each node is initially unmarked and waits to receive a *downstream* message from its parent. Each *downstream* message contains a list of pairs, each containing a node ID and the neighbors of the node ID. Once a *downstream* message is received from its parent, the node becomes marked. The node adds it own node ID and neighbor set to the list of pairs, and sends this list in its *downstream* message to its children. Once the *downstream* message is sent, the node waits to receive *upstream* messages from all of its children. Whenever a leaf node receives its *downstream* message, it similarly adds its pair and sends the *upstream* message to its parents. The *upstream* messages are created by consolidating the neighbor lists in the *upstream* messages received from all of the node's children. Once the leader (root) node receives the *upstream* messages from its children, it can construct the adjacency matrix $A(\mathcal{G})$, which is the quantity returned in *InitiateConvergecast*. Once each non-leader node sends its *upstream* message, it begins its *ParticipateBroadcast* procedure. The convergecast procedure requires $\mathcal{O}(diam)$ rounds and $\mathcal{O}(|\mathcal{E}|)$ messages [10].

After the convergecast procedure terminates, the leader node has the adjacency matrix $A(\mathcal{G})$. It then initiates a broadcast using the BFS tree to provide the other nodes with the adjacency matrix. Each non-leader node waits for the adjacency matrix to arrive from its parent, and then relays the information to its children. Upon sending its message, the node then calls *DetermineRobustness* to obtain the values of r and s. The broadcast operation requires $\mathcal{O}(diam)$ rounds and $\mathcal{O}(|\mathcal{E}|)$ messages [10]. By combining the message and round complexity of the procedures in *DecentralDetermineRobust*, it follows that *DecentralDetermineRobust* requires $\mathcal{O}(diam)$ rounds and $\mathcal{O}(n \times diam|\mathcal{E}|)$ messages. Of course, *DecentralDetermineRobust* also inherits the worst-case complexity of *DetermineRobustness* given in Proposition 1.

3.4 A Distributed Algorithm

Here, we present a distributed algorithm for determining the robustness of a connected network. Instead of simply using the centralized algorithm, *DetermineRobustness*, after obtaining the adjacency matrix, as was done in Algorithm 3.3, in this case the computation required to determine robustness is reduced by only checking the edge reachability properties in subsets in which the node is *not* a member. The BFS tree construction is again used to elect a leader and provide an efficient means to broadcast information. In this algorithm, however, a second convergecast/broadcast sequence must be performed after the estimates of r and s are determined in order for the nodes to obtain the true values of r and s. *DistributedDetermineRobust* is given in Algorithm 3.4.

Algorithm 3.4: DISTRIBUTEDDETERMINEROBUST($\text{ID}_i, \mathcal{N}_i$)

$(Leader, Par, Chldrn) \leftarrow$ LEADERELECTBFSTREE($\text{ID}_i, \mathcal{N}_i$)
if $(Leader == \text{ID}_i)$
  then $\begin{cases} A(\mathcal{G}) \leftarrow \text{INITIATECONVERGECAST}(\text{ID}_i, Chldrn) \\ \text{BROADCAST}(A(\mathcal{G}), Chldrn) \end{cases}$
  else $\begin{cases} \text{PARTICIPATECONVERGECAST}(\text{ID}_i, \mathcal{N}_i, Par, Chldrn) \\ A(\mathcal{G}) \leftarrow \text{PARTICIPATEBROADCAST}(Par, Chldrn) \end{cases}$
$(\hat{r}, \hat{s}) \leftarrow$ DETERMINEPARTIALROBUST($A(\mathcal{G}), \text{ID}_i$)
if $(Leader == \text{ID}_i)$
  then $\begin{cases} (r, s) \leftarrow \text{INITIATECONVERGECAST2}(\hat{r}, \hat{s}, Chldrn) \\ \text{BROADCAST2}(r, s, Children) \end{cases}$
  else $\begin{cases} \text{PARTICIPATECONVERGECAST2}(\hat{r}, \hat{s}, Par, Chldrn) \\ (r, s) \leftarrow \text{PARTICIPATEBROADCAST2}(Par, Chldrn) \end{cases}$
return (r, s)

Before describing *DeterminePartialRobust*, we explain the difference in the second convergecast/broadcast sequence. In the second sequence, the nodes must determine the true values of r and s from their estimates. Therefore, the *downstream* and *upstream* messages of *InitiateConvergecast2* and *ParticipateConvergecast2* contain the minimum value of r seen along the downstream path, along with the minimum value of s seen for the given minimum value of r. For example, if a node's estimate is $(\hat{r} = 4, \hat{s} = 1)$ and it receives a *downstream* message containing $(\hat{r} = 3, \hat{s} = 3)$, then the pair sent in the next *downstream* message is $(\hat{r} = 3, \hat{s} = 3)$. Similarly, once the downstream paths reach leaf nodes, the *upstream* messages are determined similarly. Once the leader node receives the *upstream* messages from its children, it can determine the true values of r and s. These values are used in the second broadcast.

DeterminePartialRobust is shown in Algorithm 3.5. Because the network is assumed to be connected, the network is at least 1-robust. Therefore, all subsets *not* including node i are always

checked in *DeterminePartialRobust* when called from *Distributed-DetermineRobust* under the assumption of a connected network. Since decrementing r and s requires at most $\mathcal{O}(n^2)$ steps, this implies that all nodes will complete *DeterminePartialRobust* within $\mathcal{O}(n^2)$ steps of each other.

For the performance improvement of *DeterminePartialRobust*, observe that by eliminating the sets in which i is an element, *DeterminePartialRobust* effectively reduces the problem from size n to $n-1$. That is, there are $R(n-1)$ pairs of subsets to check in *DeterminePartialRobust*, instead of $R(n)$ pairs of subsets, as in *DetermineRobustness*. By using Pascal's Rule

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$$

we can show that the number of pairs of subsets that are avoided in *DeterminePartialRobust* is

$$R(n) - R(n-1) = \sum_{k=1}^{n-1} \binom{n-1}{k} (2^k - 1).$$

Notice that the number of pairs above is also the number of subsets in which i is a member. The difference between this number and $R(n-1)$ is

$$n - 1 + \sum_{k=2}^{n-1} \binom{n-1}{k} (2^{k-1} + 1) > R(n-1)$$

Therefore, it is more than twice as efficient to check the subsets in which i is *not* a member rather than checking only subsets in which it *is* a member. However, the worst-case complexity of the algorithm is not improved. Also, the round and message complexity for rounds in which communication is needed coincide with the round and message complexity of the decentralized algorithm.

Algorithm 3.5: DETERMINEPARTIALROBUST($A(\mathcal{D}), i$)

$r \leftarrow \min\{\delta^{\text{in}}(\mathcal{D}), \lceil n/2 \rceil\}$
comment: $\delta^{\text{in}}(\mathcal{D})$ is the minimum in-degree of $\mathcal{D}$

$s \leftarrow n$
for $k \leftarrow 2$ **to** $n-1$
  for each $K'_i \in \mathcal{K}'_k$ $(i = 1, 2, \ldots, \binom{n-1}{k})$
  comment: $\mathcal{K}'_k$ is the set of $\binom{n-1}{k}$ unique k-subsets of $\mathcal{V} \setminus \{i\}$
    for each $P'_j \in \mathcal{P}'_{K_i}$ $(j = 1, 2, \ldots, 2^{k-1} - 1)$
    comment: $\mathcal{P}'_{K_i}$ is the set of partitions of K'_i with exactly two nonempty parts
      comment: $P'_j = \{\mathcal{S}_1, \mathcal{S}_2\}$
      $isRSRobust \leftarrow$ ROBUSTHOLDS($A(\mathcal{D}), \mathcal{S}_1, \mathcal{S}_2, r, s$)
      if ($isRSRobust ==$ **false**) **and** ($s > 0$)
        then $s \leftarrow s - 1$
      while ($isRSRobust ==$ **false**) **and** ($r > 0$)
        do
          while ($isRSRobust ==$ **false**) **and** ($s > 0$)
            do
              $isRSRobust \leftarrow$ ROBUSTHOLDS($A(\mathcal{D}), \mathcal{S}_1, \mathcal{S}_2, r, s$)
              if not $isRSRobust$
                then $s \leftarrow s - 1$
          if ($isRSRobust ==$ **false**)
            then
              $r \leftarrow r - 1$
              $s \leftarrow n$
      if $r == 0$
        then return (r, s)
return (r, s)

4. CONCLUSIONS

In this paper we have presented several algorithms for checking and determining the robustness of a network. We present two centralized algorithms, a decentralized algorithm, and a distributed one. All algorithms are inefficient; they are $\mathcal{O}(m3^{\sqrt{m}})$ in the size of the input. This is to be expected because the problem of determining the robustness of a network is NP-hard [16]. In order to improve on efficiency, one must consider approximate algorithms. We are currently investigating polynomial-time approximations that provide conservative estimates of the robustness of the network.

5. ACKNOWLEDGMENTS

This work has been supported in part by the National Science Foundation (CNS-1035655, CCF-0820088), the U.S. Army Research Office (ARO W911NF-10-1-0005), and Lockheed Martin. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

6. REFERENCES

[1] R. Albert and A. L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74(1):47–97, Jan. 2002.

[2] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, June 2006.

[3] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[4] C. Godsil and G. Royle. *Algebraic Graph Theory*. Springer-Verlag New York, Inc., 2001.

[5] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard. Resilient network coding in the presence of Byzantine adversaries. In *26th IEEE International Conference on Computer Communications, INFOCOM*, pages 616–624, Anchorage, AL, May 2007.

[6] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482–491, Cambridge, MA, Oct. 2003.

[7] H. J. LeBlanc. *Resilient Cooperative Control of Networked Multi-Agent Systems*. PhD thesis, Department of EECS, Vanderbilt University, 2012.

[8] H. J. LeBlanc, H. Zhang, X. D. Koutsoukos, and S. Sundaram. Resilient asymptotic consensus in robust networks. *To appear in IEEE Journal on Selected Areas in Communications, special issue on In-Network Computation: Exploring the Fundamental Limits*, 2013.

[9] H. J. LeBlanc, H. Zhang, S. Sundaram, and X. Koutsoukos. Consensus of multi-agent networks in the presence of adversaries using only local information. In *Proceedings of the 1st International Conference on High Confidence Networked Systems (HiCoNS)*, pages 1–10, Beijing, China, 2012.

[10] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, California, 1997.

[11] F. Pasqualetti, A. Bicchi, and F. Bullo. Consensus computation in unreliable networks: A system theoretic approach. *IEEE Transactions on Automatic Control*, 57(1):90–104, Jan. 2012.

[12] W. Ren, R. W. Beard, and E. M. Atkins. Information consensus in multivehicle cooperative control. *IEEE Control Systems Magazine*, 27(2):71–82, April 2007.

[13] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21:25–34, August 1987.

[14] S. Sundaram and C. N. Hadjicostis. Distributed function calculation and consensus using linear iterative strategies. *IEEE Journal on Selected Areas in Communications*, 26(4):650–660, May 2008.

[15] S. Sundaram and C. N. Hadjicostis. Distributed function calculation via linear iterative strategies in the presence of malicious agents. *IEEE Transactions on Automatic Control*, 56(7):1495–1508, July 2011.

[16] H. Zhang and S. Sundaram. Robustness of complex networks with implications for consensus and contagion. *CoRR*, abs/1203.6119, 2012.

[17] H. Zhang and S. Sundaram. Robustness of information diffusion algorithms to locally bounded adversaries. In *Proceedings of the American Control Conference*, pages 5855–5861, Montréal, Canada, 2012.