

# A Robust Spanning Tree Topology for Data Collection and Dissemination in Distributed Environments

Darin England, *Member, IEEE,*

Bharadwaj Veeravalli, *Senior Member, IEEE, IEEE-CS*

and Jon Weissman, *Senior Member, IEEE*

Darin England and Jon Weissman are with the Department of Computer Science and Engineering, University of Minnesota, Twin Cities. Bharadwaj Veeravalli is with the Department of Electrical and Computer Engineering, The National University of Singapore.

## Abstract

Large-scale distributed applications are subject to frequent disruptions due to resource contention and failure. Such disruptions are inherently unpredictable and therefore *robustness* is a desirable property for the distributed operating environment. In this work we describe and evaluate a robust topology for applications that operate on a spanning tree overlay network. Unlike previous work that is adaptive or reactive in nature, we take a proactive approach to robustness. The topology itself is able to simultaneously withstand disturbances and exhibit good performance. We present both centralized and distributed algorithms to construct the topology, and then demonstrate its effectiveness through analysis and simulation of two classes of distributed applications: data collection in sensor networks and data dissemination in divisible load scheduling. The results show that our robust spanning trees achieve a desirable trade-off for two opposing metrics where traditional forms of spanning trees do not. In particular, the trees generated by our algorithms exhibit both resilience to data loss and low power consumption for sensor networks. When used as the overlay network for divisible load scheduling, they display both robustness to link congestion and low values for the makespan of the schedule.

## Index Terms

Robustness, Distributed computing, Graph theory, Fault tolerance, Wireless sensor networks, Divisible load scheduling

## I. INTRODUCTION

The design and implementation of distributed computing systems has historically been carried out with performance being the dominant goal. Typically the objective is to optimize a criterion such as response time, makespan, or hit rate. Furthermore, the performance metrics are usually viewed from an individual perspective and may not correspond to the social optima. In order to realize the benefits from such performance-oriented designs, the distributed environment in which the application is deployed must be somewhat predictable. That is, calculation of the *optimal* schedule often requires accurate and *á priori* knowledge of system load, communication latencies, and execution times of individual tasks. With the current trend toward large-scale, geographically separated systems with shared computational resources, the assumption of exact knowledge of system parameters is unrealistic. Hence there is a need to incorporate robustness into the design of distributed systems. Robust systems perform well across a wide range of operating conditions and exhibit graceful degradation under anomalous conditions [1]. In this work we present a

technique for improving the robustness of a distributed system for applications that operate on a spanning tree overlay network. Spanning trees are widely used in communication networks as a means to disseminate information from one node to all other nodes and/or to collect information at a single designated node. The defining characteristic of our spanning tree topology when compared to other types of commonly seen spanning trees is that the resulting trees perform well for multiple, conflicting metrics; the trees are robust.

The importance of robustness in the design of complex and distributed systems is well-established [1]–[6]. Biological systems naturally form robust topologies that are resilient to attack [4], [5]. Social networks exhibit the small-world phenomenon in which any two members are separated by just a few acquaintances. This phenomenon can be viewed as a form of robustness since information reaches every person in the network very quickly despite the fact that some people will not pass on the information. More pertinent to our interests is the ability of distributed computing systems to maintain performance despite the presence of various perturbations. This should be a fundamental concept in the design of distributed systems. Particularly when network bandwidth and computational resources are shared, robustness is a desirable system property because communication delays and execution times are inherently difficult to predict. Nonetheless, whenever we have some choice in the design of the system, e.g. the topology of an overlay network, then we can influence the system’s response to various disturbances. There is often a trade-off for incorporating robustness into a system and our work is no exception in this regard. However, we will show that the price paid in performance loss is well worth that gained when the operating environment is unpredictable.

In the next section we compare and contrast related work in the areas of system design and scheduling. We then discuss in Section III the application model for which our technique is appropriate and present both centralized and distributed algorithms for constructing the robust topology. The form of the resulting spanning tree is compared to other spanning trees that are commonly found in the literature. Through analysis and simulation we then evaluate the effectiveness of the technique on two different distributed applications. In section IV we consider an application to sensor networks wherein data is forwarded up the tree and collected at a single node. The results show that the trees effected by our algorithm admit near-optimal resilience to node failures (we prove the optimal case) and at the same time are very efficient with respect to power consumption. In section V we present an application to divisible load scheduling in which

work originates at a single node and is distributed over a spanning tree network for the purpose of minimizing the makespan via parallel execution. The focus of this particular application is not optimality of the schedule per se, but rather it is to show that our technique can be used to easily construct robust solutions that achieve a desired trade-off: low execution times and robustness to network congestion. A discussion of future work and the conclusion is given in Section VI.

## II. RELATED WORK

### A. System Design

Several researchers have argued that robustness is a crucial property in the design and operation of distributed systems [1]–[3], [6], [7]. Gribble [1] suggests using techniques such as admission control, system introspection, and adaptive control to achieve robustness in distributed applications. These techniques are all adaptive in nature. Our approach is different in that we take a proactive approach toward robustness and therefore adaptation is not required. Thus our work is most appropriate in situations where an immediate change in the network topology is undesirable. An example of such a situation is discussed in Section IV. In the physics community, Carlson and Doyle have introduced a concept for the incorporation of robustness into the design of complex systems. They have named this concept *highly optimized tolerance* (HOT) [6], [8]. Using examples from biology and engineering, they show how high-performance (or high-yield) systems can be made robust against disturbances for which they were designed to handle, yet can fail catastrophically when subjected to unanticipated disturbances. This is the trade-off between the need for high-performance and the increased sensitivity to randomness. In a similar fashion, our approach strikes a trade-off between performance and resilience to network disruptions.

### B. Scheduling

In [9] Barlas and Veeravalli present a technique for the delivery of continuous-media documents over unreliable communication links. Progress in this area is important in order to realize the widespread adoption of video-on-demand services over the Internet. The solution presented in [9] is to deliver multiple installments of the document to the client. Each installment is transferred from a different, distributed server. A proxy at the client assembles the pieces. By making a small, adjustable portion of each installment overlap with the adjoining section, the system is able to withstand a certain amount of data loss without compromising the continuous nature of

the delivery. Similar to our work, the authors of [9] assume no knowledge of where network disruptions will occur. Given the desired probability of interrupt-free playback, the authors show how to compute the appropriate amount of overlap.

With respect to the divisible load scheduling problem Ghose et. al. [10] investigate probing strategies for estimation of network parameters and subsequent use of those parameters to allocate work to processors. Thus, no previous knowledge of bandwidth nor computing speeds is required. Indeed, one strategy that was investigated employs continuous probing and can adapt to changing network parameters. However, any probing strategy requires a certain amount of overhead. Furthermore, we again stress that our work is purely proactive in nature as opposed to being adaptive or reactive. As such the overhead associated with measuring and adapting is not an issue.

Bölöni and Marinescu [11] study the robustness of scheduling meta-programs onto grid-like distributed systems. Meta-programs can be represented as directed acyclic graphs (trees) in which nodes are component programs and edges are either data, communication, or host dependencies among components. The problem addressed by the authors is that the execution times of the component programs are non-deterministic. An unexpected long execution time of a component may cause other dependent components to be late, resulting in an increase in the overall execution time of the meta-program. Their measure of the robustness of a schedule is based on the concept of a critical path, which is a path through the graph such that if any component along the path is late, then the meta-program will be late. Given the probabilities of individual components being late, one may compute the probability of any particular path in the schedule becoming critical. Then, a schedule with fewer critical paths is considered to be more robust. In our work with spanning trees we have something akin to a critical path whenever a node has many children because if that node fails then the entire subtree below it is lost.

### III. MODEL

For many distributed applications, the routing of data and messages takes place on a virtual overlay network that is constructed on top of the underlying physical network. For example, nodes in peer-to-peer systems are connected via the physical links in the Internet; however, a node forwards queries only to nodes in its own list of neighbors, thus forming an overlay network. Not surprisingly, the topology of such an overlay network plays a significant role in

the performance and efficiency of the distributed system. Herein we address distributed systems for which the overlay network is a spanning tree, i.e., a connected network that contains no cycles. Furthermore, one particular node in the network is designated as the root node. The root node acts as a collection point for data (as in a sensor network) and/or as a load origination point for the distribution of work (as in divisible load scheduling). Throughout this paper nodes are identified by indices and the root node is always labelled with the numeral one.

### A. Traditional Spanning Trees

For a moderate size network with just a few neighbors per node, there exist many possible spanning trees. For a dense network the number is enormous<sup>1</sup>. Given the numerous possibilities, the question arises as to which spanning tree is best for a particular application. The most commonly seen forms of spanning trees are the following.

**Shortest paths:** The distance in edge weights of the path from each node to the root node is minimum. Such a tree is efficiently constructed by Dijkstra's algorithm. We designate this method as SP.

**Fewest hops:** The distance in number of hops along the path from each node to the root node is minimum. This method is equivalent to SP when all edge weights are equal and therefore Dijkstra's algorithm may be employed. We designate this method by FH.

**Minimum weight:** The total sum of edge weights is minimum. Such a tree can be constructed by either Kruskal's algorithm or by Prim's algorithm [12] and does not take into consideration the location of the root node. We designate this method as MST.

Spanning trees created by FH tend to be shallow and "fat", with the average node degree being fairly large. This is because the only criterion for cost is the distance in hops from the root with no consideration of edge weights. We will show in Section IV that FH minimizes the expected value of the amount of data loss when a node or link fails. However, it is not the best choice for other performance metrics such as power consumption in sensor networks. At the other end of the spectrum, MST produces trees that are very deep and "skinny". This is natural since the only

<sup>1</sup>The exact number is determined from the Matrix Tree Theorem, a presentation of which can be found in [12], [13].

criterion is edge weight and the location of the root node is not taken into consideration. The shape of trees produced by SP are influenced by the distribution of edge weights, but they tend to be deeper and have smaller node degrees than FH trees. As we will discuss in Sections IV and V these characteristics are good for features such as power consumption in sensor networks and minimization of makespan in load scheduling.

In each of the three construction methods above, the spanning tree that results may not be unique. This fact will make no difference for our analysis and experiments. For example, we take a probabilistic approach to computing the amount of data that is lost when nodes fail. Any two MST trees of the same underlying original graph are equivalent in the sense that they both have the same expected value for the amount of data loss.

### *B. Robust Spanning Trees*

Instead of settling for one extreme or the other, we seek a method to construct spanning trees that effect good trade-offs: trees that are relatively immune to data loss when nodes or links fail and yet are able to maintain good performance. Indeed, this is the very notion of robustness. Through analysis and simulation we will show that the the spanning trees that perform best for different, and even opposing metrics are constructed by considering a weighted combination of hop count and path weight as follows.

$$\lambda \times \text{hop count} + (1 - \lambda) \times \text{path weight} \quad (1)$$

where  $0 \leq \lambda < 1$ . If more importance is placed on hop count, then the tree will tend to be fat and shallow. Alternatively, more importance on path weight means that the tree will be skinny and deep. The type of tree that performs best depends on the metric of interest. In order to construct trees that perform well under a wide variety of metrics, we attempt to make the tree fat near the root and skinny further away from the root. The intuition (with respect to data collection in sensor networks) is that the further a message has to travel to reach the root node, the more likely it is to encounter a failed parent somewhere along the way. After a message has travelled a certain distance, the network has already “invested” resources (i.e. power and bandwidth) to get the message that far. When a message gets close to the root node, we want to give it the best possible chance to make it the rest of the way so that its payload will be recorded. The weight  $\lambda$  is really a function of a node’s depth in the tree. When an edge  $(i, j)$  is being considered for

inclusion in the tree and  $i$  is the new vertex not already in the tree, then

$$\lambda_i = 1 - \frac{h_i}{\epsilon_1}, \quad (2)$$

where  $h_i$  is the hop count of node  $i$  from the root and  $\epsilon_1$  is the eccentricity of the root node. The eccentricity of a node is the largest of the shortest paths from that node to all other nodes. We measure eccentricity in number of hops, not path weight. Another way to think about it is that (our measure of) eccentricity is the depth of the deepest leaf in the SP tree. However, note that the eccentricity of a node is a characteristic of the underlying graph; it is not a property of the overlay network. Using this measure of eccentricity in Equation 2 ensures that  $0 \leq \lambda_i < 1$  for all  $i$ . It also effects values for  $\lambda_i$  that are close to one when selecting nodes that are near the root, and values close to zero when selecting nodes that are further from the root. This gives the desired relative importance of hop count versus path weight in Equation 1. We now present two algorithms for constructing a robust spanning tree: a centralized version and a fully distributed version.

1) *A Centralized Algorithm:* The centralized algorithm is appropriate in situations where the node on which the algorithm runs has full knowledge of the nodes and link speeds in the underlying network. This is the case for the application to Divisible Load Scheduling discussed in Section V. Our algorithm is based on Prim's algorithm for constructing MST. Prim's algorithm begins with a single node (the root node in our case) and at each iteration the cheapest edge that incorporates a new vertex is selected for inclusion in the tree. For MST the cheapest edge is simply the one with the smallest edge weight (ties may be broken randomly). In our algorithm the cheapest edge is computed as in Equation 1.

The complete method for generating spanning trees in this way is presented as Algorithm 1 and the shorthand identifier is RB. We now discuss the computational complexity. Algorithm 1 requires that we compute the eccentricity of the root node. This can be accomplished by using Dijkstra's algorithm to compute the shortest paths from the root node to every other node and then taking the maximum distance (in number of hops) as the eccentricity. The complexity of Dijkstra's algorithm is  $O(|V|^2)$  [13]. The process of adding edges into the partial spanning tree requires at most  $|V| - 1$  cost computations and comparisons for every node in the graph (except the root node which is added initially). For this step, the worst-case time complexity occurs for complete graphs and is  $O(|V|^2)$ . However, the average time complexity is  $O(a \times |V|)$  where  $a$  is

the average node degree. Thus the overall complexity of Algorithm 1 is  $O(|V|^2)$  and is therefore *good* in the sense that it is bounded by a polynomial in the size of the graph.

**Data** : graph  $G = \{V, E\}$  with edge weights  $z_{ij}$   
 compute the eccentricity of the root node  $\epsilon_1$ ;  
 initialize the tree with the root node only;  
**while** *there are still vertices not yet added to the tree* **do**  
   **for** *every vertex  $i$  not in the tree* **do**  
     compute  $\lambda_i = 1 - (h_i/\epsilon_1)$ ;  
     compute  $\xi_i = \lambda_i \times h_i + (1 - \lambda_i) \times (\xi_j + z_{i,j})$ ;  
     store the minimum cost found so far;  
   **end**  
 add the vertex  $i$  along edge  $(i, j)$  that achieves the minimum cost;  
**end**

**Algorithm 1:** A centralized algorithm

2) *A Distributed Algorithm:* For some applications it is unrealistic to assume that any single node will have complete knowledge of the network. This is the case, for example, for wireless and ad hoc sensor networks as presented in Section IV. For such applications we require a distributed algorithm wherein each node runs the same algorithm and the tree is constructed after each node exchanges a series of messages with its neighbors. The well-known Bellman Ford algorithm can be used in this manner to construct SP and FH trees. Let  $x_i$  be node  $i$ 's estimate of its distance from the root node, either in path weight or in number of hops, and let  $z_{ij}$  be the weight on the link between nodes  $i$  and  $j$ . Then the  $k$ th iteration of the Bellman Ford algorithm has the form

$$x_i^k = \min_{j \in A(i)} (z_{ij} + x_j^{k-1}), \quad i = 2, \dots, n, \quad (3)$$

$$x_1 = 0, \quad (4)$$

where  $A(i)$  is the neighborhood of node  $i$ . The algorithm terminates when  $x_i^k = x_i^{k-1}$  for all  $i$ . This algorithm is naturally suited to a distributed implementation since each node may carry out its iteration in parallel with the other nodes. In the synchronized version, a node may only proceed to the next iteration after it has received the updates of the previous iteration from all of

its neighbors [14]. It has been shown [14] that using the initial conditions

$$x_1^0 = 0 \tag{5}$$

$$x_i^0 = +\infty \quad i = 2, \dots, n, \tag{6}$$

the algorithm will terminate after at most  $m^*$  iterations, where

$$m^* = \max_{i=2, \dots, n} m_i \leq n - 1, \tag{7}$$

and  $m_i$  is the number of edges contained in a shortest path from  $i$  to 1. Thus the worst-case scenario is  $n - 1$  edges. Now assume that communication time dominates the complexity. That is, we assume that it takes significantly more time to transmit a message than it does to make an addition and a comparison. Each iteration of the synchronous version of the Bellman Ford algorithm requires  $2 \times |E|$  messages. Thus the serial solution time is  $O(m^*|E|)$ . Note that for no additional cost we can simultaneously construct the SP and FH trees by simply maintaining two sets of the equations (3). Upon termination of the Bellman-Ford algorithm each node will know

- 1) its parent in both the SP tree and the FH tree,
- 2) its distance to the root in edge weights (and those of its neighbor's),
- 3) its distance to the root in number of hops (and those of its neighbor's).

Now the idea is that each node  $i$  simply chooses a parent  $j$  based on the form of the weighted cost in Equation 1. It is important to note that the hop count of a node  $i$  is the number of hops to the root if node  $i$  chooses node  $j$  as its parent. Similarly, the path weight of a node  $i$  is the distance from node  $i$  to the root if node  $i$  chooses node  $j$  as its parent. Each node  $i$  scans its list of neighbors and chooses a parent  $j$  according to

$$\min_{j \in A(i)} \{ \lambda_i \times (1 + h_j) + (1 - \lambda_i) \times (z_{ij} + d_j) \}. \tag{8}$$

where  $\lambda_i$  is computed as

$$\lambda_i = 1 - \frac{h_i}{\epsilon_1}. \tag{9}$$

This method will result in a tree being constructed as long as 1) no two nodes choose each other as a parent (a simple check to implement), and 2) no cycles are constructed (impossible if all cycles have positive cost). There is one element to communication cost for our approach that must be incurred over and above that of the Bellman-Ford algorithm: each node must know

the eccentricity  $\epsilon_1$  of the root node. We accomplish this in the following way. After Bellman-Ford terminates, each node sends a message destined for the root node (using the SP tree). The message contains a counter that is incremented at each hop. After a reasonable amount of time, the root node scans the messages and determines the depth of the deepest leaf (i.e. the eccentricity). The root node then sends this information back down the tree. The worst-case number of messages will be  $n - 1$  (in each direction). Again assuming that communication time is the dominant factor, the complexity of our distributed algorithm is  $O(m*|E|)$  to run Bellman-Ford (a cost that would have to be incurred anyway to construct either SP or FH trees), plus  $O(|V|)$  to communicate the eccentricity of the root node.

To illustrate the effect of our algorithm, Figure 1 shows the results of the four different construction methods that we have discussed. The underlying graph for this figure is a 100-node random graph wherein each node has between 20 and 30 neighbors, uniformly distributed. The edge weights are uniformly distributed between .1 and 10. Figure 1(d) shows the robust spanning tree from our distributed algorithm. It is closest in form to the FH spanning tree shown in Figure 1(b); however, the distribution of node degrees is not as heavy-tailed as in FH. Hence the failure of any particular node will not result in as much disruption to the network as the loss of a highly connected node in the FH tree. The RB tree in Figure 1(d) was constructed using the centralized algorithm of section III-B.1. We mention here that both versions of our algorithm are heuristic in nature. They do not necessarily produce the exact same tree; however, they do produce trees with the same properties since they both use exactly the same cost function. For this reason, the results presented in each of the next two sections are nearly identical regardless of which algorithm is used.

#### IV. APPLICATION TO SENSOR NETWORKS

Several different application areas are now employing wireless sensor networks [15]–[17]. Environmental monitoring for natural disasters and wildlife habitat, building automation, and military surveillance are common examples. The model of data flow in such systems is many-to-one, which naturally corresponds to a spanning tree topology. Messages are forwarded up the tree from child to parent to the root node which is typically connected to a storage device and/or a wired network. The overlay network upon which data is routed affects both the fault tolerance and the longevity (via battery life) of the system [18]. In this regard the primary characteristics

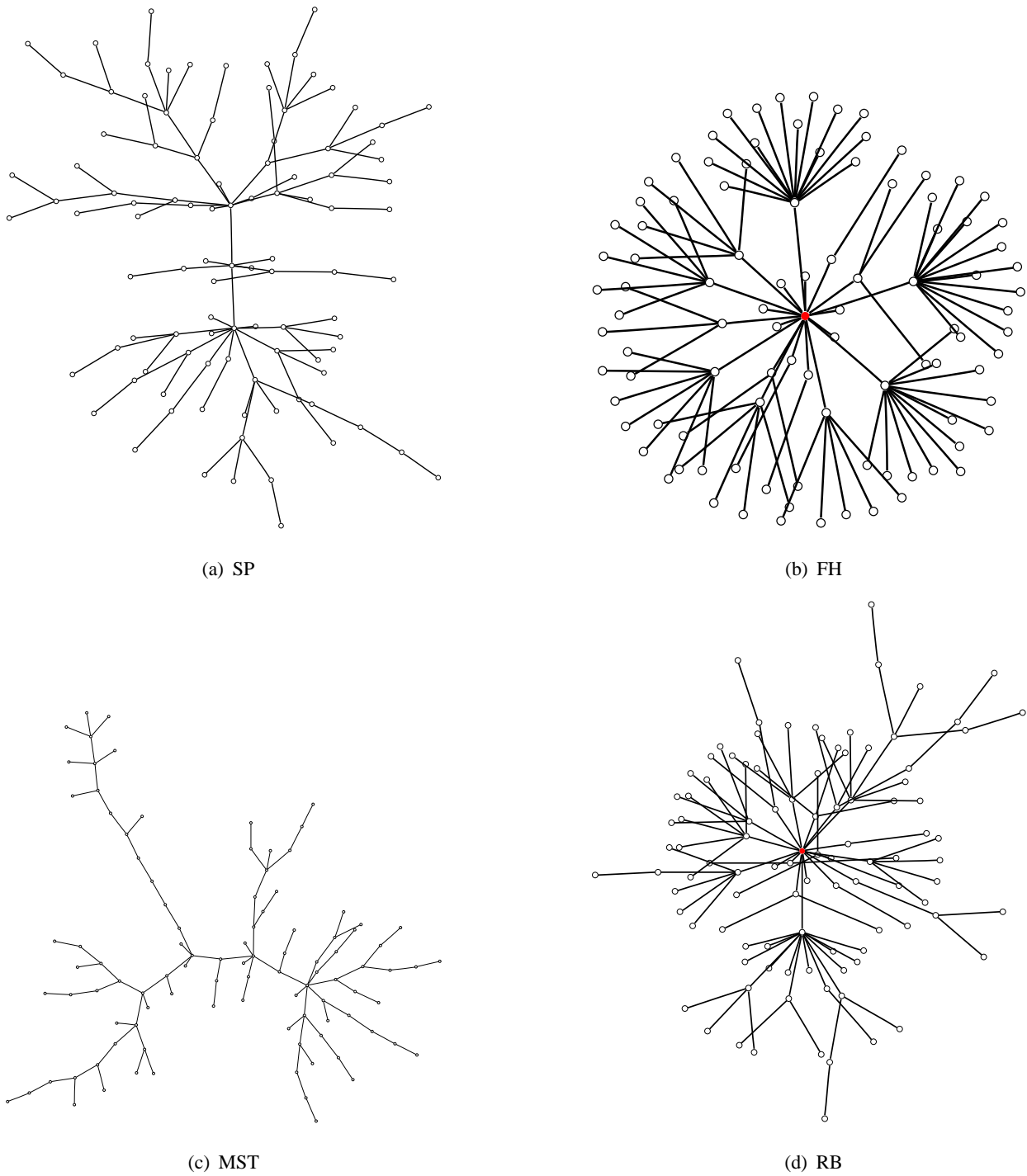


Fig. 1. Spanning trees constructed from a 100-node random graph. The node degrees of the original underlying graph are distributed uniformly between 20 and 30 neighbors. Figure 1(d) is the spanning tree constructed by Algorithm 1. All nodes are just a few hops from the root, which allows shorter paths for data transmission than the SP and MST trees of Figures 1(a) and 1(c), respectively. In addition, the relatively low number of highly connected nodes means that there is less chance of massive data loss as compared to the fewest hops spanning tree of Figure 1(b).

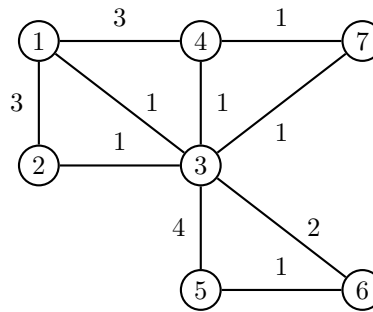


Fig. 2. A Small Sensor Network

are the distribution of node degrees and the depth of the tree. In general, nodes that transmit over longer distances or through obstructions consume more power. Also, parent nodes of large subtrees use more power since they must forward more data (minus any data aggregation). Such nodes also expose the network to the potential for massive data loss if they (or their upward links) happen to fail.

Figure 2 shows a small sensor network that consists of seven nodes. An edge between two nodes indicates that they can communicate directly. The edge weight is the amount of power required to transmit a single message between the two nodes. A larger weight indicates a greater distance or an obstruction. Node 1 is the root node. It is the collection point to which all other nodes must route their data. The SP, FH, and RB spanning trees for this network are shown in Figure 3. In this case the MST tree happens to be the same as the SP tree. Again we stress a proactive approach in our work. In particular we assume no *a priori* knowledge concerning exactly which nodes or links will fail. As such we compute the statistical expectation of the amount of data loss given the number of nodes that fail and the corresponding probabilities of failure. In the next two sections we define metrics for data loss and power consumption. In doing so we present a theorem and its proof for the form of the spanning tree that minimizes the expected value of data loss (the FH tree). Using these metrics we then compare our robust spanning tree topology RB to the SP, FH and MST topologies.

### A. Expected Data Loss

For clarity of presentation we focus on node failures as opposed to link failures. Note that for tree networks there is no loss of generality in doing so because, as far as the amount of data

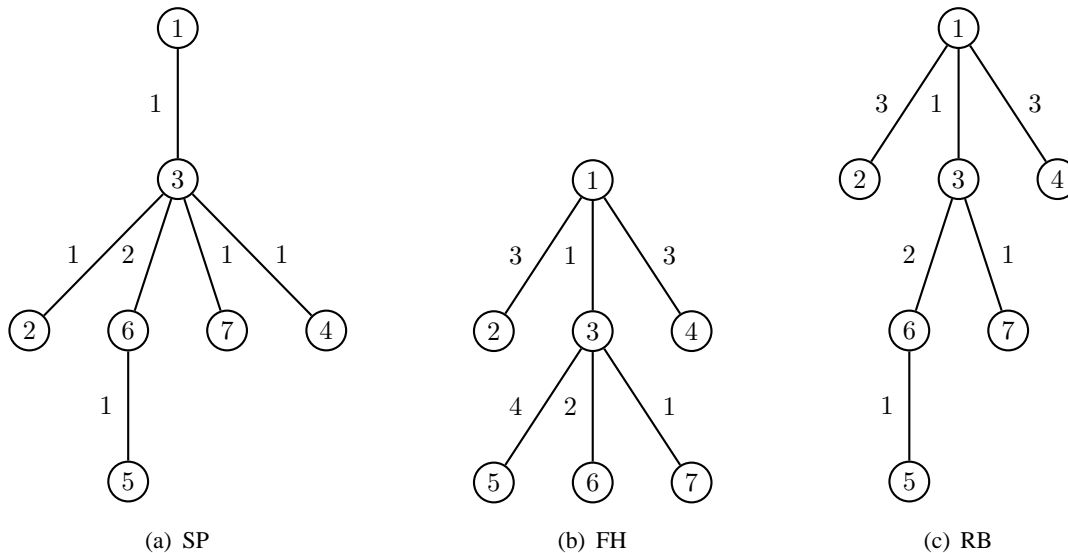


Fig. 3. Spanning Trees of the Sensor Network of Figure 2

loss is concerned, the failure of a node is equivalent to the failure of the link to the parent. This applies to every node in the tree except the root node, which we assume will not fail. The reason for omitting this possibility is that if the root node fails, then the entire data collection process is halted regardless of the topology of the overlay network. Thus we see no reason to include this possibility when comparing alternative topologies. Consider a tree  $T$  with vertex set  $V(T)$  and edge set  $E(T)$ . Let  $m_i$  be the number of nodes in the subtree rooted at node  $i$  (including node  $i$  itself) and let  $q_i$  be the probability that node  $i$  will fail. Then the expected value of data loss  $L$  given that exactly one node fails is

$$\mathbf{E}\{L \mid \text{exactly one node fails}\} = \sum_{i \in V(T)} m_i \times q_i, \quad (10)$$

where

$$\sum_{i \in V(T)} q_i = 1. \quad (11)$$

Throughout this work we assume that all nodes have an equal probability of failure. The expected value of data loss then becomes

$$\mathbf{E}\{L \mid \text{exactly one node fails with equal probabilities}\} = \frac{1}{n-1} \sum_{i \in V(T)} m_i, \quad (12)$$

where  $n = |V(G)|$  is the number of nodes in the graph. Using Equation 12 the expected data loss of the spanning tree in Figure 3(a) is

$$\mathbf{E}\{L\} = \frac{1}{6}(6 + 1 + 2 + 1 + 1 + 1) = 2.0. \quad (13)$$

Similarly for Figure 3(b),  $\mathbf{E}\{L\} = 1.5$ , and for Figure 3(c),  $\mathbf{E}\{L\} = 1.667$ . In this case the FH spanning tree admits the smallest value for expected data loss. This is intuitive since the depth of the tree is as small as possible. Indeed, Theorem 1 shows that the FH spanning tree minimizes the expected value of data loss for any graph.

**Theorem 1** *Given an arbitrary graph representing a sensor network in which each node has an equal probability of failure, the spanning tree that minimizes the number of hops from the root processor also minimizes the expected value of the amount of data loss.*

*Proof:* The proof is by induction on the number of nodes in the tree. The case of one or two nodes is trivial since there is only one spanning tree for each of those cases. Therefore our base case is a network consisting of three nodes as shown in Figure 4(a). The expected data loss of the minimum-hop spanning tree, shown in Figure 4(b), is computed as  $(1/2) \times (1 + 1) = 1$ . There are two other trees, and they are isomorphic to each other. One of these is shown in Figure 4(c), and its expected data loss is  $(1/2) \times (2 + 1) = 1.5$ . Thus the theorem is true for the base case.

Now suppose that the theorem is true for an arbitrary graph with  $n$  nodes. Let us examine what happens when we increase the size of the network by a single node, labelled  $i$ , and suppose that we have multiple choices of where to attach node  $i$  in the spanning tree. We may attach the new node  $i$  to a node  $v$  which is  $r$  hops away from the root node, or we may attach the new node  $i$  to a node  $u$  which is  $s$  hops away from the root node, where  $s < r$ . Let  $X$  be the expected amount of data loss before the new node  $i$  is added and let  $X_v$  be the expected amount of data loss if node  $i$  is attached to node  $v$ . Define  $X_u$  similarly. Then it suffices to show that  $X_u > X_v$ . When we (re-)compute the expected amount of data loss after adding in the new node, the only nodes that change the value of the computation are the nodes along the path from  $v$  to the root if node  $i$  is attached to node  $v$  (there are  $r$  links on this path), or along the path from  $u$  to the root if node  $i$  is attached to node  $u$  (this path consists of  $s$  links). In the (re-)computation we

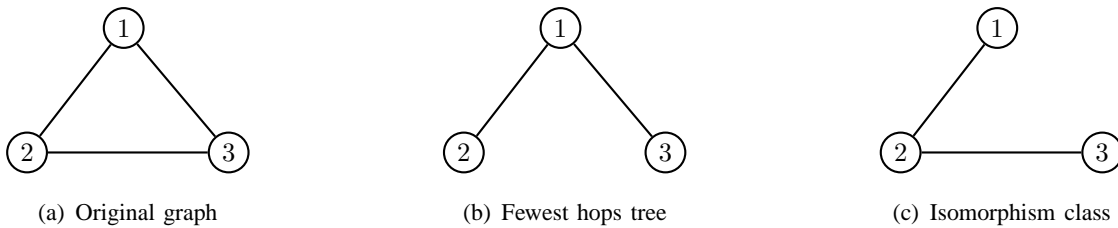


Fig. 4. Base Case for Theorem 1

add one for each node along the path from  $i$  to the root.

$$X_u = X + s < X + r = X_v \quad (14)$$

$$X_u < X_v \quad (15)$$

The inequalities are true since  $s < r$  and so the theorem is true for any size graph. Thus choosing the smallest number of hops  $s$  away from the root will minimize the expected amount of data loss. ■

A corollary to Theorem 1 arises when there are multiple parent nodes for connecting a new node, and all of the possible parents are equidistant from the root. That is, in the arbitrary underlying graph, the new node can communicate with more than one node that is exactly  $s$  hops away from the root and  $s$  is minimal. As far as minimizing the expected value of data loss in this case, it makes no difference to which of these equidistant nodes we attach the new node. Of course this extension is only valid when all nodes are equally likely to fail. In the case of different probabilities for node failures we would need to consider the node degrees.

### B. Power Consumption

Nodes consume power when they transmit and receive data. More power is required to transmit and receive over longer distances and through obstructions. The amount of data forwarded up the tree also plays a role in power consumption since more data means more transmissions. Hence parents of large subtrees are subject to fast depletion of their power reserves. A partial offset to this situation occurs when data can be aggregated and thus fewer forwarded messages are required. However, even with data aggregation, parent nodes must still receive the data to be aggregated and also expend processor power performing the aggregation itself. Furthermore, data aggregation does not reduce the number of sampled data points and so the loss of a large subtree

will still result in a much smaller sample size, regardless of the extent of data aggregation. We consider three metrics for power consumption: 1) the total amount of power consumed by the network, 2) the maximum amount of power consumed by a single node, and 3) the number of messages associated with the node that consumes the maximum amount of power. The second and third perspectives are important since the particular node in question will be the first to deplete its power supply.

Ignoring data aggregation for the moment, the amount of network power consumed when all nodes send one message to the root node is the sum over all nodes of the product of the number of messages sent and the power required to transmit a single message. Again we let  $m_i$  be the number of nodes in the subtree rooted at node  $i$ . and let  $z_{i,j}$  is the weight on the link from node  $i$  to its parent node  $j$ . Then the total network power  $P$  required to collect a single data observation is

$$P = \sum_{i \in V(T)} m_i \times z_{i,j}. \quad (16)$$

Using the spanning trees of Figure 3 to illustrate the calculation, the total network power for the SP tree of Figure 3(a) is (going breadth-first through the tree)

$$P = (6 \times 1) + (1 \times 1) + (2 \times 2) + (1 \times 1) + (1 \times 1) + (1 \times 1) = 14. \quad (17)$$

Similarly,  $P$  for the FH tree of Figure 3(b) is 17, and  $P$  for the robust tree of Figure 3(c) is 16. It can be shown that SP trees will admit the minimal values for  $P$ . (The proof is very similar to the proof in Theorem 1 and is therefore omitted.) This is natural since the weights on the paths to the root node are smallest. We may now start to see the trade-off between expected data loss and power consumption. At one end of the spectrum, SP trees use low power, but expose the network to greater possibilities of data loss when nodes fail. On the other hand, FH trees minimize expected data loss, but consume more power on the whole. The trees generated by our methodology offer a compromise that the results of the next section make apparent.

### *C. Simulation and Results for Randomly Generated Networks*

Using the metrics that we defined for expected data loss and power consumption, we now evaluate the performance and robustness of different spanning trees via simulation on three categories of randomly generated networks. The three categories are

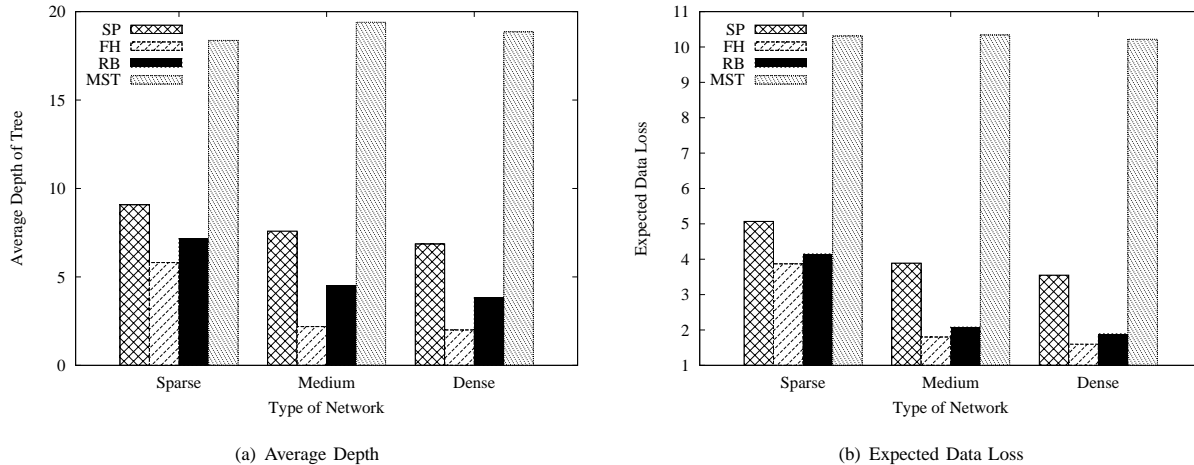


Fig. 5. Average tree depth and expected data loss results on randomly generate networks.

Sparse: Each node has between 1 and 10 neighbors.

Medium: Each node has between 20 and 30 neighbors.

Dense: Each node has between 40 and 50 neighbors.

The number of neighbors is uniformly distributed in the respective ranges. For each category, we generated 100 random graphs using the method and software described in [19]. Our procedure for generating a random graph was to produce the uniformly distributed degree sequence, ensure that a graph with that degree sequence indeed exists<sup>2</sup>, and then use the software of Viger and Latapy to generate the simple<sup>3</sup>, connected graph. The edge weights for all three categories were uniformly distributed between 0.1 and 10. For each of the three categories, the data loss and power consumption metrics presented in sections IV-A and IV-B were computed for each of the 100 randomly generated graphs. The results presented in this section and in section V are the averages over the 100 random graphs in the respective categories.

Average tree depths are shown in Figure 5(a). Naturally, the FH trees exhibit the smallest depths. The robust trees are deeper than FH, but not as deep as SP. MST does not consider the location of the root node and so it generates the deepest trees. For this reason, MST is not a good candidate for sensor networks. We show the results of MST for comparison purposes only. In accordance with Theorem 1 and as shown in Figure 5(b), FH trees admit the smallest values

<sup>2</sup>This is accomplished by employing the Theorem of Havel-Hakimi [12].

<sup>3</sup>Simple means no loops and no multiple edges.

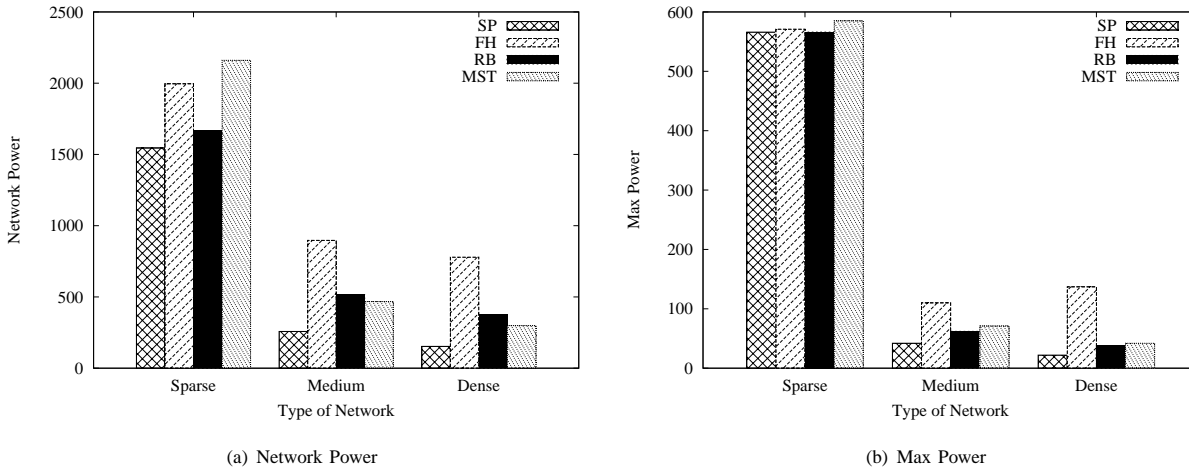


Fig. 6. Power consumption results on randomly generated networks.

for expected data loss. Note that the expected data loss for RB trees are only slightly greater: 2.08 vs. 1.80 for medium density networks and 1.89 vs. 1.60 for dense networks. Thus with respect to data loss the RB trees perform quite well. However, the real benefit of the RB method comes from the combination of low data loss and relatively low power consumption. This can be seen in the results for network power consumed and the maximum power used by any one node, as shown in Figures 6(a) and 6(b), respectively. Ignoring MST as a candidate, the power consumption metrics for RB fall in between SP (the lowest) and FH (the highest). Especially for maximum power used, the RB values are closer to SP than to FH. This is evidence of the best of both worlds: data loss similar to FH trees and power consumption closer to SP trees.

A metric that combines the notions of data loss and power consumption can be viewed in the following way. The node that consumes the most power, i.e. the node that accounts for the maximum power metric, will be the first node to deplete its power supply. We consider the size of the subtree rooted at this node (i.e.  $m_i$ ), and hence the number of data points lost when this node ceases operation. In other words, we measure the number of messages that are forwarded by the node that first depletes its power supply. This metric is shown in Figure 7. In this respect, for medium and dense networks, the RB trees exhibit the smallest amount of data loss.

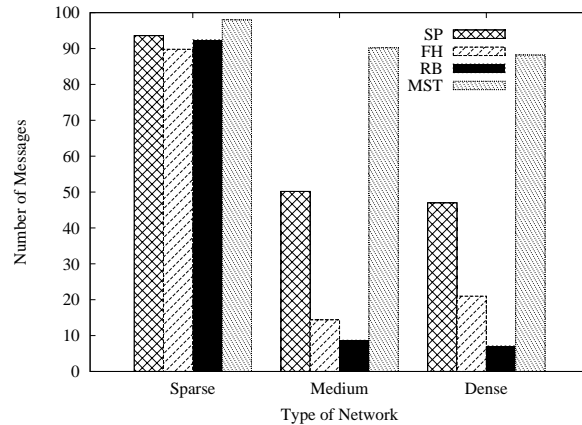


Fig. 7. Results for the number of messages associated with the node that consumes the maximum amount of power and hence is the first to deplete its power supply.

## V. APPLICATION TO DIVISIBLE LOAD SCHEDULING

Divisible Load Scheduling (DLS) is the process of simultaneously scheduling the data and computations of a data-parallel application onto multiple processors. In the basic version of the problem the data originates at a single processor (the root node) and the objective is to assign each processor an amount of data such that the total time to transmit and process all of the data (the makespan) is minimized. Thus the flow of data is reversed when compared to the data collection operation of sensor networks. A key aspect of the problem is that the data may be divided into chunks of arbitrary size. Some applications that are amenable to this computing paradigm include image processing, data mining, and matrix-vector multiplication. The DLS problem has received a significant amount of attention in the computer science literature [20]–[25]. Most of the existing work has been oriented toward solutions for structured, well-defined, multi-processor networks: linear-chain, bus, tree, hypercube, and 3-D mesh networks. Moreover, the vast majority of existing work has also assumed that all computational resources, i.e. transmission links and processors, are dedicated to the application at run-time. While that work forms the foundation of the theory surrounding DLS, there is a need to examine the performance of these types of applications in modern distributed computing systems. A parallel application running in such an environment may not have full, dedicated access to a resource during its execution. In this situation the topology of the overlay network on which the DLS problem is solved plays an important role in the quality of the overall solution.

Contrary to the work in structured networks, we are concerned with solving the DLS problem on arbitrary graphs, such as those effected by wide area networks and wireless networks. To solve the DLS problem, we adopt the technique of Yao and Veeravalli [26], which distributes the data onto a spanning tree of the underlying network. The structure of the spanning tree affects both the speed of processing and the robustness to network disturbances. In this regard, we show that the solutions obtained on the RB spanning trees generated by Algorithm 1 (the centralized algorithm) achieve both speed and robustness. With respect to the robustness of a solution, we consider the possibility that any particular transmission link or processor will be congested and will therefore perform at a reduced speed. We examine the impact on the total processing time of a DLS application due to delays in the transmission of data. In the next section we show that it is appropriate to only consider tree-based solutions to the DLS problem on arbitrary networks. We then show how to measure transmission delay and processing delay for the DLS problem when the network and computational resources are subject to congestion from resource sharing. In doing so we present results and compare the performance and robustness of the different spanning tree construction methods on randomly generated networks.

#### A. *Tree-based Solutions*

In [26] Yao and Veeravalli present a method for allocating divisible loads to processors in arbitrary networks. Their technique, RAOLD-OS (Resource Aware Optimal Load Distribution with Optimal Sequencing), is to generate a minimum-weight spanning tree (MST) on the arbitrary network and then solve the DLS problem on that tree. The allocation of load on a tree network is accomplished by a two-step process:

- 1) Child nodes are reduced with the parent node into a representation of an equivalent processor. This recursive process stops when the entire tree has been reduced to a single equivalent processor.
- 2) While expanding back into the tree network, the workload is allocated onto single-depth subtrees according to time-balance equations.

The principle of optimality in the DLS literature states that in the optimal allocation of load, all processors must stop executing at the same time instant [27]. Otherwise, some processor will be idle and could have accepted more load, thus reducing load on other processors and shortening the makespan. Constructing a spanning tree overlay network on an arbitrarily complex architecture

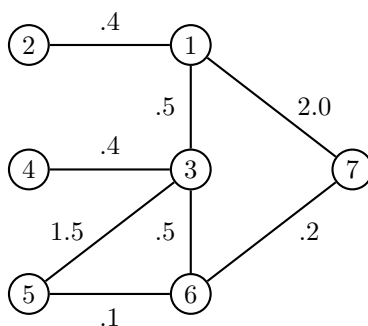


Fig. 8. An Example Network for the DLS Problem

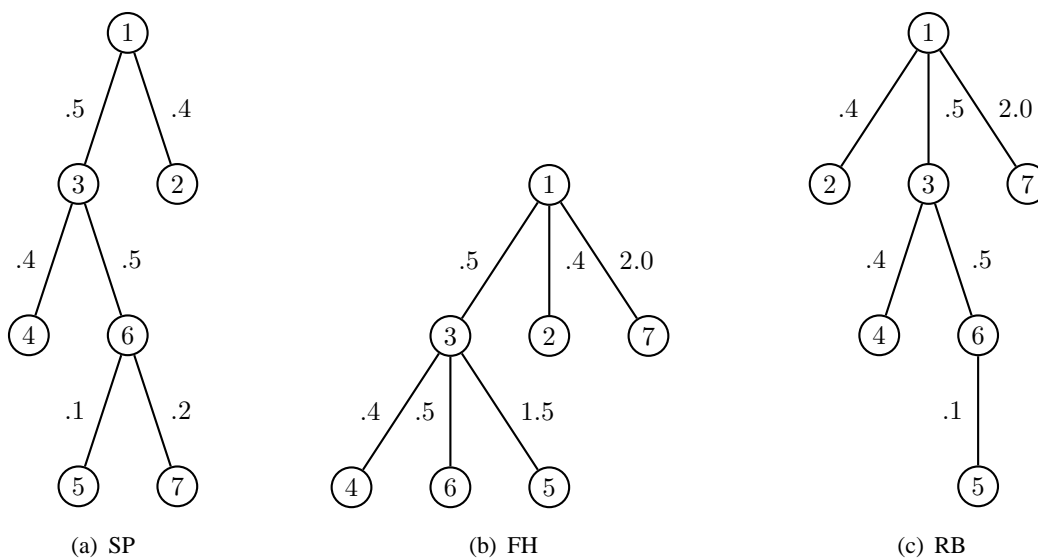


Fig. 9. Spanning Trees of the Network in Figure 8

is a natural approach to simplify the problem. In fact it is not difficult to show [28] that the optimal solution to the DLS problem indeed occurs on *some* spanning tree of the original graph.

So given an arbitrary network, the optimal solution to the DLS problem occurs on a spanning tree. Furthermore, the RAOLD-OS procedure finds the optimal solution for the particular spanning tree on which it is executed [26]. The question then arises: How do we find the spanning tree that admits the global optimal solution? In this work we do not address the optimality criterion directly, but rather, our goal is to identify spanning trees that are easy to construct and that exhibit qualities of fast processing time and robustness to network disruptions. The spanning trees generated by Algorithm 1 satisfy this goal.

TABLE I  
OPTIMAL LOAD ALLOCATION PERCENTAGES FOR THE NETWORK OF FIGURE 8

Spanning Tree	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$	$\alpha_6$	$\alpha_7$	Makespan
SP, MST	.398	.284	.126	.090	.035	.038	.029	398.1
FH	.385	.275	.125	.090	.024	.060	.042	384.7
RB	.384	.274	.124	.089	.041	.046	.042	384.1

Let us examine the optimal solutions for the example network in Figure 8 and its associated spanning trees in Figure 9 when there is no disruption to the network. That is, all transmission links and processors operate at their prescribed speeds. For this example all processors operate at the same speed of  $\omega = 1$ . Node 1 is the load origination point and the total amount of workload to be transmitted and processed is  $L = 1000$ . The percentage of the total load assigned to processor  $i$  is  $\alpha_i$ . Naturally, all of the load allocation percentages must sum to one. For most DLS problems on arbitrary networks, SP admits the smallest makespan. However, the network of Figure 8 serves as an example that this is not always the case. The optimal load allocations and makespans listed in Table I show that FH and RB give smaller makespan values. Figure 10 shows more comprehensive results for solving the DLS problem on two sets of randomly generated networks: medium-density and dense as described in section IV-C. The results for sparse networks are not shown because the the performance of all spanning trees were approximately the same. As indicated earlier, SP trees show the best performance in terms of makespan. However, the RB spanning trees exhibit the second-best makespan values and we will see in the next section that this performance combined with their robustness to network disruptions makes them very attractive candidates on which to solve the DLS problem.

### B. Transmission Delay

Similar to the way we measured data loss for sensor networks, we take a probabilistic approach to measuring transmission delay for the DLS problem. This is because even though we assume that disruption to the network will occur, we do not know in advance exactly which link or processor will be congested. If we had such knowledge then we would factor it into the problem's solution. Therefore we compute the expected value of the transmission delay given the number of links that will experience congestion and the magnitude of that congestion. Similar to the

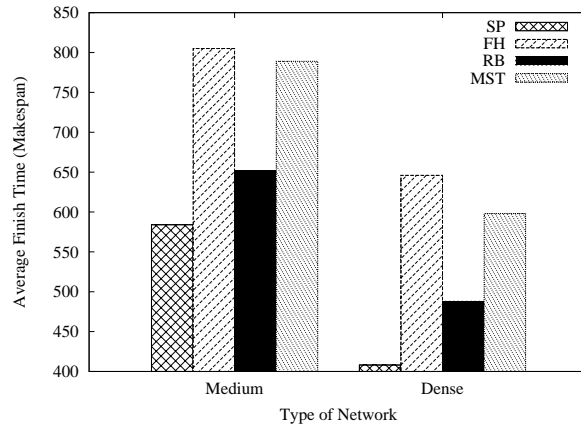


Fig. 10. Average finish time (or makespan) for the DLS problem on randomly generated networks.

notation found in the DLS literature, we let  $z_{i,j}$  be the inverse speed of the link from node  $i$  to node  $j$ . The units of  $z_{i,j}$  are seconds per unit of work and hence smaller values indicate faster transmission speeds.  $T_{\text{cm}}$  is the time to transmit a standard unit of workload.

Now, let us compute the expected transmission delay that would occur if a link  $(i, j)$  exhibits congestion and its bandwidth is reduced to a certain percentage of its full capacity, specified by  $\kappa$ , where  $0 < \kappa \leq 1$ . The effective transmission speed of link  $(i, j)$  is reduced to  $z_{i,j}/\kappa$  and the delay  $D$  incurred for transmitting  $\alpha_j \times L$  units of workload is<sup>4</sup>

$$D = \alpha_j L \frac{z_{i,j}}{\kappa} T_{\text{cm}} - \alpha_j L z_{i,j} T_{\text{cm}} = \left(\frac{1}{\kappa} - 1\right) (\alpha_j L z_{i,j} T_{\text{cm}}), \quad (18)$$

where  $0 < \kappa \leq 1$ . Let  $x_{i,j}$  be the total amount of workload transmitted on link  $(i, j)$ . This amount of workload includes the portion processed by processor  $j$  and all of the workload that  $j$  passes on to other processors, i.e.  $j$ 's children. Let  $q_{i,j}$  be the probability that link  $(i, j)$  exhibits the performance degradation specified by  $\kappa$ . Then the expected value of the total amount of transmission delay  $D$  is

$$\mathbf{E}\{D\} = \sum_{(i,j) \in E(T)} q_{i,j} \left(\frac{1}{\kappa} - 1\right) x_{i,j} z_{i,j} T_{\text{cm}}, \quad (19)$$

where the summation is over all links in the tree  $T$ . Let us use equation 19 to compare the robustness of the spanning trees in Figure 9 to a single link congestion in the amount  $\kappa = .50$ .

<sup>4</sup>We still think of  $j$  as the parent of  $i$ ; hence the subscript  $j$ .

Recall that we do not know in advance exactly which link will be congested, so for this example we assume that all links have an equal probability of being congested, i.e.  $q_{i,j} = 1/6$  for all links  $(i, j)$  in the spanning tree. For the spanning tree of Figure 9(a), setting all processor speeds  $\omega_i = 1$ ,  $L = 1000$ ,  $T_{cm} = 1$ , and factoring the constant terms gives

$$\begin{aligned} \mathbf{E}\{D \mid \text{single link congestion, } \kappa = .50\} = \\ \frac{1}{6} \left( \frac{1}{\kappa} - 1 \right) L \left[ (\alpha_2)z_{1,2} + (\alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 + \alpha_7)z_{1,3} \right. \\ \left. + (\alpha_4)z_{3,4} + (\alpha_5 + \alpha_6 + \alpha_7)z_{3,6} + (\alpha_5)z_{6,5} + (\alpha_7)z_{6,7} \right] = 61.5 \quad (20) \end{aligned}$$

The same parameters for the FH spanning tree of Figure 9(b) give  $\mathbf{E}\{D\} = 74.1$  and for the RB spanning tree of Figure 9(c) give  $\mathbf{E}\{D\} = 71.0$ . SP (which happens to be the same as MST in this small example) shows the greatest robustness to link congestion. Figure 11(a) shows the results of expected transmission delay for the two sets of randomly generated networks. RB and MST are the most robust, with MST being slightly better for medium-density networks and RB being slightly better for dense networks. In dense networks there is more freedom for Algorithm 1 to add nodes into the spanning tree and this results in a tree structure that is better suited to our purposes. For both categories of network density, RB shows more robustness than SP to link delay. Recall that SP admits the best performance in the form of small values for makespan. Thus we see that the RB spanning trees of Algorithm 1 again achieve a desirable trade-off: acceptable performance and very good robustness to link congestion.

One may take a more conservative (or pessimistic) approach to measuring transmission delay and consider what would happen in the worst case scenario of link congestion. In other words, we can measure the delay that would occur if the link that causes the maximum amount of delay is in fact the one to experience congestion. (This is the bottleneck link.) We simply measure the delay (potentially) caused by all  $n - 1$  links and take the maximum,  $D_{\max}$ .

$$D_{\max} = \left( \frac{1}{\kappa} - 1 \right) \max_{(i,j) \in E(T)} \{x_{i,j} z_{i,j} T_{cm}\}. \quad (21)$$

The maximum delay results for the two sets of randomly generated networks are presented in Figure 11(b) (for  $\kappa = .50$ ). Under this conservative metric, RB trees are the most robust to network disruptions. It is interesting that SP trees are the second-most robust, which is unlike the results for expected link delay in Figure 11(a). This is further evidence that the RB trees

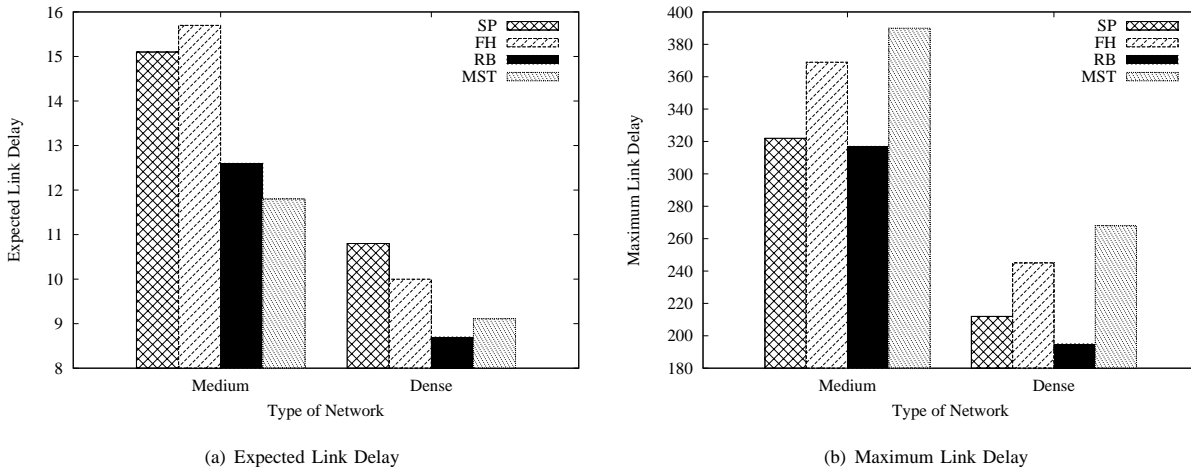


Fig. 11. Transmission delay results for the DLS problem on randomly generated networks.

are robust in general. Indeed good performance under a wide variety of metrics is a hallmark of robustness.

## VI. CONCLUSION

Robustness is an important property for distributed computing systems. These systems are subject to resource contention and hence node failures and transmission delays are common enough to warrant their consideration in system design. This is especially true when the application designer has some control over the manner in which data is routed and computations are performed, such as the choice of topology for an overlay network. In this work we presented a methodology for constructing a spanning tree overlay network that exhibits robustness to network disturbances. The construction technique employs a weighted formula for hop count and path weight that changes the relative importance as the distance from the root node changes. This results in trees that perform well for a wide variety of metrics. When compared to the most common forms of spanning trees, our robust trees are closest in appearance to fewest-hops spanning trees. However, the node degree distribution is not as highly skewed, which results in less probability for massive data loss when highly connected nodes fail.

To construct such a topology, we presented both centralized and fully distributed versions of the algorithm. We applied the distributed version to the problem of data loss and power consumption in wireless sensor networks, and the centralized version to the problem of network congestion

in divisible load scheduling. The approach we used to measure data loss was statistical since we assumed no à priori knowledge concerning which nodes would fail. Experiments on three sets of randomly generated graphs show that the robust spanning trees exhibit values for expected data loss that are comparable to the best possible values of fewest-hops spanning trees, while displaying power consumption values that are closer to those of shortest-paths spanning trees. Hence we were able to effect a trade-off that achieves the best characteristics of two opposing metrics. When used as the overlay network for the divisible load scheduling problem on arbitrary graphs, our spanning trees admit good performance for makespan values and at the same time are extremely robust to link congestion. Again the results show that these spanning trees achieve good performance for two opposing metrics where traditional forms of spanning trees do not.

Our approach toward robustness is proactive rather than reactive. It is natural to ask when a node realizes that its parent has failed, why not simply choose another parent (assuming the node has multiple neighbors)? This may or may not be desirable. If there are many nodes that choose a new parent, then the properties of the tree will be unknown. For example, several nodes could (unknowingly) choose the same parent and cause its power supply to be quickly depleted. If the goal of the system is to collect as much data as possible in a short amount of time, then this could be a good strategy. However, if the goal is to collect a reasonable amount of data over a long period of time, then it would be better to use a (partially failed) topology about which we have some statistics. It seems that the pertinent question is: At what point is it worth rerunning the spanning tree construction algorithm to construct a new tree? This is one subject of our future work.

#### ACKNOWLEDGMENT

The authors would like to acknowledge the support of the National Science Foundation under grants CNS-0305641 and ITR-0325949, the Department of Energy's Office of Science under grant DE-FG02-03ER25554, and the Minnesota Supercomputing Institute for Digital Simulation and Advanced Computation and the Digital Technology Center at the University of Minnesota. Bharadwaj Veeravalli would like to acknowledge the support under grant (0520150024/R-263-000-350-592) by A\*STAR SERC through National Grid Office, Singapore.

## REFERENCES

- [1] S. D. Gribble, "Robustness in complex systems," in *Proceedings IEEE Eighth Workshop on Hot Topics in Operating Systems*, May 2001, pp. 21–26.
- [2] D. England, J. Weissman, and J. Sadagopan, "A new metric for robustness with application to job scheduling," in *IEEE International Symposium on High Performance Distributed Computing 2005 (HPDC-14)*, July 2005, research Triangle Park, NC.
- [3] D. Oppenheimer, V. Vahdat, and D. A. Patterson, "Towards a framework for automated robustness evaluation of distributed services," in *FuDiCo II: S.O.S. Survivability: Obstacles and Solutions, 2nd Bertinoro Workshop on Future Directions in Distributed Computing*, June 2004, university of Bologna Residential Center, Bertinoro, Italy.
- [4] M. Aldana and P. Cluzel, "A natural class of robust networks," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 100, July 2003, pp. 8710–8714.
- [5] R. Albert, H. Jeong, and A. L. Barabási, "Error and attack tolerance of complex networks," *Nature*, vol. 406, pp. 378–382, July 2000.
- [6] J. M. Carlson and J. Doyle, "Highly optimized tolerance: Robustness and design in complex systems," *Physical Review Letters*, vol. 84, pp. 2529–2532, 2000.
- [7] S. Ali *et al.*, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 630–641, July 2004.
- [8] J. M. Carlson and J. Doyle, "Highly optimized tolerance: A mechanism for power laws in designed systems," *Physical Review E*, vol. 60, 1999.
- [9] G. Barlas and B. Veeravalli, "Optimized distributed delivery of continuous-media documents over unreliable communication links," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 10, Oct. 2005.
- [10] D. Ghose, H. J. Kim, and T. H. Kim, "Adaptive divisible load scheduling strategies for workstation clusters with unknown network resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 10, Oct. 2005.
- [11] L. Bölöni and D. C. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, vol. 5, no. 5, pp. 395–412, 2002.
- [12] D. B. West, *Introduction to Graph Theory*, 2nd ed. Prentice Hall, 2001.
- [13] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. North-Holland, 1976.
- [14] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.
- [15] D. Ganesan *et al.*, "Large-scale network discovery: Design tradeoffs in wireless sensor systems," in *Proceedings of the Symposium on Operating Systems Principles (SOSP 2001)*, Oct. 2001, lake Louise, Banff, Canada.
- [16] I. F. Akyildiz *et al.*, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–116, Aug. 2002.
- [17] K. Sohrabi *et al.*, "Protocols for self-organization of a wireless sensor network," *IEEE Personal Communications*, pp. 16–27, Oct. 2000.
- [18] P. Santi, "Topology control in wireless ad hoc and sensor networks," *ACM Computing Surveys*, vol. 37, no. 2, pp. 164–194, June 2005.
- [19] F. Viger and M. Latapy, "Efficient and simple generation of random simple connected graphs with prescribed degree sequence," in *The Eleventh International Computing and Combinatorics Conference*, Aug. 2005, kumming, China.
- [20] B. Veeravalli and N. Viswanadham, "Suboptimal solutions using integer approximation techniques for scheduling divisible

- loads on distributed bus networks,” *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 30, no. 6, pp. 680–691, Nov. 2000.
- [21] M. Drozdowski, *Selected problems of scheduling tasks in multiprocessor computer systems*. Poznan University of Technology Press, 1997.
- [22] S. Bataineh, T.-Y. Hsiung, and T. G. Robertazzi, “Closed form solutions for bus and tree networks of processors load sharing a divisible job,” *IEEE Transactions on Computers*, vol. 43, no. 10, Oct. 1994.
- [23] V. Bharadwaj, D. Ghose, and V. Mani, “Optimal sequencing and arrangement in distributed single-level tree networks with communication delays,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 9, Sept. 1994.
- [24] J. Sohn and T. G. Robertazzi, “An optimal load sharing strategy for divisible jobs with time-varying processor speeds,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 3, pp. 907–923, July 1998.
- [25] G. D. Barlas, “Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 5, May 1998.
- [26] J. Yao and B. Veeravalli, “Design and performance analysis of divisible load scheduling strategies on arbitrary graphs,” *Cluster Computing*, vol. 7, no. 2, pp. 191–207, 2004.
- [27] V. Bharadwaj *et al.*, *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.
- [28] D. England, “Robust design for distributed computing systems,” Ph.D. dissertation, Department of Computer Science and Engineering, University of Minnesota, Twin Cities, June 2006.



**Darin England** Darin England received a PhD in Computer Science from the University of Minnesota in June 2006. His research interests are at the intersection of Computer Science and Operations Research and include stochastic modeling and performance analysis of distributed systems. He received a BS in Industrial Engineering from Purdue University and an MS in Operations Research from Georgia Institute of Technology. Dr. England also has eight years of professional experience as an Operations Research Analyst in the transportation industry, where he designed and implemented large-scale mathematical optimization models. He is currently a Senior Scientist at Ingenix, a division of United Health Group.



**Bharadwaj Veeravalli** Bharadwaj Veeravalli, Senior Member, IEEE and IEEE-CS, received his BSc in Physics, from MK University, India, 1987, ME(ECE) from IISc, Bangalore, India, 1991 and PhD(Aerospace Engg), IISc, Bangalore, India, 1994. He was a post-doctoral fellow in the Department of CS, Concordia University, Montreal, Canada, 1996. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) division, at The National University of Singapore, Singapore, as a tenured Associate Professor. His main stream research interests include, Cluster/Grid computing, Scheduling problems in PDS, Bioinformatics, and Multimedia computing. He has published over 85 papers in high-quality International Journals and Conferences and co-authored three research monographs in the areas of PDS(1996), Distributed Databases(2003), and Networked Multimedia Systems(2005). He is currently serving the Editorial Board of IEEE Transactions on Computers, IEEE Transactions on SMC-A, and International Journal of Computers and Applications, USA, as an Associate Editor. BV's most recent profile and other information can be found in <http://cnds.ece.edu.sg/elebv>



**Jon Weissman** Jon B. Weissman received the B.S. degree from Carnegie-Mellon University in 1984, and the M.S. and Ph.D. degrees from the University of Virginia in 1989 and 1995 respectively, all in Computer Science. He has been an Associate Professor of Computer Science at the University of Minnesota since 2003. His current research interests are in distributed systems, high performance computing, Internet systems, and Grid computing. He is a senior member of the IEEE.