QoS Aware and Survivable Network Design for Planned Wireless Sensor Networks

Abhijit Bhattacharya and Anurag Kumar
Dept. of Electrical Communication Engineering
Indian Institute of Science, Bangalore, 560012, India
email: abhijit@ece.iisc.ernet.in, anurag@ece.iisc.ernet.in

Abstract—We study the problem of wireless sensor network design by deploying a minimum number of additional relay nodes (to minimize network cost) at a subset of given potential relay locations, in order to convey the data from already existing sensor nodes (hereafter called source nodes) to a Base Station, while meeting a certain specified hop count bound (the hop count bound is chosen to ensure a pre-determined probability of the data being delivered to the BS within a given maximum delay). We study two variations of the problem.

First we study the problem of guaranteed QoS connected network design, where the objective is to have at least one path from each source to the BS with the specified hop count bound. We show that the problem is NP-Hard. For a problem in which the number of existing sensor nodes and potential relay locations is \( n \), we propose an \( O(n) \) approximation algorithm of polynomial time complexity. Results show that the algorithm performs efficiently (in over 90% of the tested scenarios, it gave solutions that were either optimal or were worse than optimal by just one relay) in various randomly generated network scenarios.

Next, we study the problem of survivable network design with guaranteed QoS, i.e., the requirement is to have at least \( k > 1 \) node disjoint hop constrained paths from each source to the BS. We show that the problem is NP-Hard. We also show that the problem of finding a feasible solution to this optimization problem is NP-Complete. We propose two polynomial time heuristics for this problem, and compare their performance on various randomly generated network scenarios.

I. INTRODUCTION

Large industrial establishments such as refineries, power plants, and electric power distribution stations, typically have a large number of sensors distributed over distances of 100s of meters from the control center. Individual wires carry the sensor readings to the control center. Recently there has been increasing interest in replacing these wireline networks with wireless packet networks \( (1, 2) \). A similar problem arises in an intrusion detection application using a fence of passive infrared (PIR) sensors \( (3) \), where the event sensed by several sensors has to be conveyed to a Base Station (BS) quickly and reliably.

The communication range of the sensing nodes is typically a few tens of meters (depending on the RF propagation characteristics of the deployment region). Therefore, usually multi-hop communication is needed to transmit the sensed data to the BS. The problem then is to design a multi-hop wireless mesh network with minimum deployment cost, i.e., minimum number of additional relays, so as to communicate from each sensing (source) node to a central node, which we will call the BS (we shall use the terms BS and sink interchangeably), while meeting certain performance objectives such as a delay bound, and packet delivery probability.

The relay placement problem can be broadly classified into two classes, namely, the \textit{unconstrained} relay placement problem, where the relay locations can be anywhere in the 2-dim plane, and the \textit{constrained} relay placement problem, where the relays can be placed only at certain pre-specified potential locations. In most practical applications, due to the presence of obstacles to radio propagation (e.g., a firewall, a large machine, or a building), or due to taboo regions (e.g., a pond or a ditch), we can not place relay nodes anywhere in the plane, but only at certain designated locations. This leads to the problem of \textit{constrained relay placement} in which the relays are constrained to be placed at certain potential relay locations, and the placement has to be done so that the resulting multi-hop network satisfies a hop count bound (which is chosen to ensure an end-to-end worst case delay bound, and a given packet delivery probability).

Figure \( [\text{I}] \) depicts the constrained relay placement problem.

- The source locations and the potential relay locations are specified.
- Only certain links are permitted. This is because some links could be too long, leading to high bit error rate and hence large packet delay. Other potential links may not exist due to an obstacle, e.g., a firewall, or a building.
- The problem is to obtain a subnetwork that connects the source nodes to the base station with the...
requirement that
1) A minimum number of relay nodes is used.
2) There are at least $k$ node disjoint paths from each source node to the BS.
3) The maximum delay on any path is bounded by a given value $d_{\text{max}}$.
4) The packet delivery probability (the probability of delivering a packet within the delay bound) on any path is $\geq p_{\text{del}}$.

In this paper, we address this problem for the case in which (a) the nodes use the CSMA/CA Medium Access Control (as standardized in IEEE 802.15.4), and (b) the traffic from the source nodes is such that at any point of time only one measurement packet flows from a source in the network to the base station. We call this the "lone packet traffic model", which is realistic for many applications where the time between successive measurements being taken is sufficiently long so that the measurements can be staggered so as not to occupy the medium at the same time. For example, see Figure 2 which depicts the CDF of end-to-end delay along the longest path (5 hops) in a typical example design obtained using our algorithm (presented in Section IV) on a practical scenario. The CDF can be obtained as a convolution of per hop delay distributions which are obtained using the backoff parameters given in the standard [4].

From Figure 2, we see that the end-to-end delay is $\leq 69$ msec (without considering the node processing delay) with probability 0.99. The per hop processing delay was measured to be 15.48 msec [5, p. 31, Section 2.2.5]. Thus, the total end-to-end delay over the longest (5 hops) path in the example design turns out to be $\leq 146.4$ msec with probability 0.99. Thus, the "lone packet" based designs allow us to monitor measurements that arrive slower than about 150 msec times the number of sources. Such a situation can be expected to arise in certain condition monitoring applications, where the sensors only generate low duty-cycle measurements not directly connected with mission critical process monitor-

Note that even if the traffic is so infrequent, the end user may still like to constrain the delay between when a measurement packet is generated and when the packet is received. In applications, the measurements are currently conveyed to the BS via a wireline network. While replacing the wireline network (which is expensive to install and maintain) with a wireless mesh network, we would like to constrain the end-to-end performance achieved by the wireless network.

We consider the practical situation in which packet losses can be caused by random channel errors, and, therefore, a random number of retransmissions are required until each packet is delivered across each link, or is dropped due to excessive retransmissions. We also permit slow fading so that the packet error probabilities on the links vary slowly over time. Statistical models for link errors and fading can be obtained from field measurement. Using such a model, we show that, to achieve the packet delivery objectives (delivery probability within a delay bound), it is sufficient to impose a certain hop count bound. This analysis is developed in Section IV.
II. DESIGN CONSTRAINTS TO ENSURE END-TO-END PERFORMANCE OBJECTIVES

A. Assumptions

In several industrial telemetry applications, the rate at which measurements are obtained from the sensors is low, for example, as little as one reading per hour from each sensor. We also assume that the alarm traffic is so infrequent that it does not interfere with any regular data transmission. Then, if the data transmission from the sensors is staggered over the hour, it can be assumed that each measurement packet flows over the network with no interference from any other packet flow. Our work in this paper is concerned with this “lone packet’’ traffic model. We also assume that IEEE 802.15.4 standard [4] is used for PHY and MAC layers.

We can obtain the bit error rate, \( \epsilon \), on a link as a function (which depends on the modulation scheme) of received Signal to Noise Ratio (SNR), by using a formula given in the standard. Then, for a Physical layer (PHY) packet data unit length of \( L \) bytes, the packet error rate (PER) on a link can be obtained as \( 1 - (1 - \epsilon)^b \).

We also permit slow fading of links; so the link PER can vary slowly over time, thus leading to the concept of “link outage”. We say, a link is in outage if the PER of the link exceeds a target maximum link PER, designated by \( q_{\text{max}} \) (obtained as a function of the target SNR, \( \gamma_{\text{min}} \)). Let us denote by \( p_{\text{out}} \), the maximum probability of a link being in outage.

Before proceeding further, we summarize for our convenience, the notations used in the development of the model.

**User requirements:**
- \( L \): The longest distance from a source to the base-station (in meters)
- \( k \): The required number of node disjoint paths between each source and the base-station
- \( d_{\text{max}} \): The maximum acceptable end-to-end delay of a packet sent by a source (packet length is assumed to be fixed and given)
- \( P_{\text{del}} \): Packet delivery probability: the probability that a packet is not dropped *and* meets the delay bound (assuming that at least one path is available from each source to the base station).

**Parameters obtained from the standard:**
- \( D_q(\cdot) \): The c.d.f. of packet delay on a link with PER \( q \), given that the packet is not dropped; \( D_q^{(h)}(\cdot) \) denotes the \( h \)-fold convolution of \( D_q(\cdot) \)
- \( b(\cdot) \): The mapping from SNR to link BER for the modulation scheme
- \( \delta(\cdot) \): The mapping from PER to packet drop probability over a link

**Design parameters:**
- \( P_{\text{xmt}} \): The transmit power over a link (assumed here to be the same for all nodes)
- \( \gamma_{\text{min}} \): The target SNR on a link
- \( q_{\text{max}} \): The target maximum PER on a link (derived from the BER function \( b(\gamma_{\text{min}}) \) and the customer specified packet length)

**Parameters obtained by making field measurements:**
- \( r_{\text{max}} \): The maximum allowed length of a link on the field to meet the target SNR, and outage probability requirements
- \( p_{\text{out}} \): The maximum probability of a link SNR falling below \( \gamma_{\text{min}} \) due to temporal variations. A link is “bad” if its outage probability is worse than \( p_{\text{out}} \), and “good”, otherwise

**To be derived:**
- \( h_{\text{max}} \): The hop count bound on each path, required to meet the packet delivery objectives

**Remark:** In practice, the value \( k \) can be chosen so that a network monitoring and repair process ensures that a path is available from each source to the BS at all times. The choice of \( k \) is not in the scope of our formulation, and would depend on how quickly the network monitoring process can detect node failures, and how rapidly the network can be repaired. We, thus,
assume that, whenever a packet needs to be delivered from a source to the BS, there is a path available, and, by appropriate choice of the path parameters (the length of each link, and the number of hops), we ensure the delivery probability, $p_{\text{del}}$.

### B. Design Constraints from Packet Delivery Objectives

Consider, in the final design, a path between a source $i$ and the base-station, which is $L$ meters away. Suppose that this path has $h_i$ hops, and the length of the $j$th hop on this path is $r_{i,j}$, $1 \leq j \leq h_i$. Then we can write

$$L_i \leq \sum_{j=1}^{h_i} r_{i,j} \leq h_i r_{\text{max}} \quad (1)$$

where the first inequality derives from the triangle inequality, and the second inequality is obvious. Since $L$ is the farthest that any source is from the base station, we can conclude that the number of hops on any path from a source to a sink is bounded below by $\frac{L}{r_{\text{max}}}$.

Following a conservative approach, we take the PER on every link to be $q_{\text{max}}$ (we are taking the worst case PER on each link, and are not accounting for a lower PER on a shorter link).

Suppose that we have obtained a network in which there are $k$ node independent paths from each source to the base-station, and all the links on these paths are good (“good” in the sense explained earlier in the definition of $p_{\text{out}}$). Consider a packet arriving at Source $i$, for which, by design, there are $k$ paths, with hop counts $h_{\ell}, 1 \leq \ell \leq k$, and suppose that at least one of these paths is available (i.e., all the nodes along that path are functioning). The availability of such a path will be determined by a separate route management algorithm, which is out of the scope of this paper. We select one of these good paths to route the packet. The path selection algorithm would incorporate a load and energy balancing strategy. If the chosen path has $h$ hops in it, then the probability that none of the edges along the chosen path is in outage is given by

$$(1 - p_{\text{out}})^{h_{\ell}}$$

Increasing $h_{\ell}$ makes this probability smaller. With this in mind, let us seek an $h_{\text{max}}$, by the following conservative approach. First, we lower bound the probability of the chosen path not being in outage by

$$(1 - p_{\text{out}})^{h_{\text{max}}}$$

Now we can ensure that the packet delivery constraint is met by requiring

$$(1 - p_{\text{out}})^{h_{\text{max}}} (1 - \delta(q_{\text{max}}))^{h_{\text{max}}} D^{(h_{\text{max}})}(d_{\text{max}}) \geq p_{\text{del}} \quad (2)$$

where the additional terms lower bound the probability that the packet is not dropped along the chosen path $((1 - \delta(q_{\text{max}}))^{h_{\text{max}}})$ and that the end-to-end delay is less than or equal to $d_{\text{max}} (D^{(h_{\text{max}})}(d_{\text{max}}))$. Recall that we take the PER on each “good” link to be $q_{\text{max}}$. The left hand expression in (2) is decreasing as $h_{\text{max}}$ increases; let $h_{\text{max}}$ be the largest value so that the inequality is met. Thus, we can meet the end-to-end performance objectives by imposing a hop count constraint $h_{\text{max}}$ from each source to the BS.

Also, combining (1) and the $h_{\text{max}}$ just obtained, we get, for every source $i$

$$r_{\text{max}} \geq \frac{L}{h_{\text{max}}} \quad (3)$$

Hence, under a given physical setting, we can convert the problem of network design with end-to-end delay bound, and guaranteed packet delivery probability to a problem of network design with end-to-end hop constraint on each path.

Henceforth in this paper, we shall concentrate on this problem of hop constrained $k$ connected cost optimal network design. We will consider the case of $k = 1$ first, and then the case, $k > 1$.

### III. THE NETWORK DESIGN PROBLEMS

#### A. Problem Formulation

With the link length constraint $r_{\text{max}}$, and the hop constraint $h_{\text{max}}$ defined in Section II, we can proceed to formulate our relay placement problems as follows:

1) **One Connected Network Design Problem:** Given the set of source nodes or required vertices $Q$ (including the BS) and the set of potential relay locations $R$ (also called Steiner vertices), consider the graph $G = (V, E)$ on $V = Q \cup R$ with $E$ consisting of edges of length $\leq r_{\text{max}}$. Then the problem is to extract from this graph, a spanning tree on $Q$, rooted at the BS, using minimum number of relays such that the hop count from each source to the BS is $\leq h_{\text{max}}$. Let us call this problem the Rooted Steiner Tree-Minimum Relays-Hop Constraint (RST-MR-HC) problem.

2) **$k$-Connected Network Design Problem:** The requirement is to have at least $k$ node disjoint hop constrained paths from each source to the sink. Then, we can formulate our relay placement problem as follows:

Given the set of source nodes or required vertices $Q$ (including the BS) and the set of potential relay locations $R$, also called Steiner vertices, consider the graph $G = (V, E)$ on $V = Q \cup R$ with $E$ consisting of edges of length $\leq r_{\text{max}}$. Then the problem is to extract from this graph, a subgraph spanning $Q$, rooted at the BS, using minimum number of relays such that each source has at least $k$ node disjoint paths to the sink, and the hop count from each source to the BS on each path is $\leq h_{\text{max}}$. Let us call this problem the Rooted Steiner Network-$k$-
Connectivity-Minimum Relays-Hop Constraint (RSNk-MR-HC) problem.

B. Complexity of the Problems

Proposition 1. The RST-MR-HC problem is NP-Hard.

Proof: The subset of RST-MR-HC problems where the hop count bound is trivially satisfied is precisely the class of RST-MR problems (consider, for example, all RST-MR-HC problems where \(|Q| + |R| = n\), \(n\) being some positive integer, and the hop count bound is \(h_{\text{max}} = n - 1\). Clearly, the hop count bound is trivially satisfied in these problems). Thus, the RST-MR problem is a subclass of the RST-MR-HC problem. This leads to the situation shown in Figure 3. But, the RST-MR problem is NP-Hard (see [6]). Hence, the RST-MR-HC problem, being a superclass of the RST-MR problem, is also NP-Hard [7] p. 63, Section 3.2.1.

Proposition 2. The RSNk-MR-HC problem is NP-Hard.

Proof: We have just proved that the problem is NP-Hard even for \(k = 1\), since that is just the RST-MR-HC problem. Therefore the general problem is also NP-Hard, using the “restriction” argument [7] p. 63, Section 3.2.1.

C. Related Literature

We see that the problem we have chosen to address belongs, broadly, to the class of Steiner Tree Problems (STP) on graphs ([8], [9], [10]).

The classical STP is stated as: given an undirected graph \(G = (V, E)\), with a non-negative weight associated with each edge, and a set of required vertices \(Q \subseteq V\), find a minimum total edge cost subgraph of \(G\) that spans \(Q\), and may include vertices from the set \(S := V - Q\), called the Steiner vertices.

The classical STP dates back to Gauss and it has been proven to be NP-Hard. Lin and Xue [11] proposed the Steiner Tree Problem with Minimum Number of Steiner Points and Bounded Edge Length (STP-MSPBEL). The STP-MSPBEL was stated as: given a set of \(n\) terminal points \(Q\) in 2-dimensional Euclidean plane, find a tree spanning \(Q\), and some additional Steiner points such that each edge has length no more than \(R\), and the number of Steiner points is minimized. The problem was shown to be NP-complete and a polynomial time 5-approximation algorithm was presented. This problem was the first well-studied problem on optimal relay placement (relay locations unconstrained).

Cheng et al. [12] studied the same problem as Lin and Xue, and proposed a 3-approximation algorithm and a 2.5-approximation algorithm.

Lloyd and Xue [13] studied a generalization of STP-MSPBEL problem where each sensor node has range \(r\) and each relay node has range \(R \geq r\). They provided a 7-approximation polynomial time algorithm. They also studied the problem of minimum number of relay placement such that there exists a path consisting solely of relay nodes between each pair of sensors. For this problem, they provided a \((5 + \epsilon)\)-approximation algorithm. The problems studied by Lloyd and Xue, as well as Cheng et al. fall in the category of unconstrained relay placement problem.

Voss [14] studied the Steiner Tree Problem with Hop Constraints (STPH). This problem is stated as: given a directed connected graph \(G = (V, E)\), with non-negative weight associated with each edge, consider a subset of \(V\), namely, \(Q = \{0, 1, 2, \ldots, n\}\) with 0 being the root vertex, and a positive integer \(H\). The problem is to find a minimum total edge cost subgraph \(T\) of \(G\) such that there exists a path in \(T\) from 0 to each vertex in \(Q\), \(Q\setminus\{0\}\) not exceeding \(H\) arcs (possibly including vertices from \(S := V - Q\)). We can call this problem the Rooted Steiner Tree-Minimum Weight-Hop Constraint problem (RST-MW-HC). This problem was shown to be NP-Hard, and a Minimal Spanning Tree based heuristic algorithm was proposed to obtain a good quality feasible solution, followed by an improvement procedure using a variation of Local Search method called the Tabu search heuristic. No performance guarantee or complexity analysis of the heuristic was provided. Also, the tabu search heuristic may not be polynomial time.

Note that an instance of the RST-MR-HC problem can be converted to an instance of the RST-MW-HC problem in polynomial time as follows: replace each relay with a directed edge of weight 1, and replace each edge associated with the relay with two directed edges (each of weight 0), one incident into the tail of the edge substituting the relay, and one going out of the tip of the edge substituting the relay. Then, minimizing the number of relays in the original problem is equivalent to minimizing the total weight in the converted problem.
Then, one could use Voss’s algorithm on this instance of RST-MW-HC problem to solve the original problem. But, as we mentioned earlier, Voss’s algorithm does not provide any performance guarantee, and because of the tabu search heuristic (which may not be polynomial time), it may take long to converge to a solution.

Costa et al. [15] studied the Steiner Tree Problem with revenue, budget, and hop constraints. Given a graph \( G = (V, E) \), with a cost associated with each edge, and a non-negative revenue associated with each vertex, the problem is to determine a revenue maximizing tree subject to a total edge cost constraint, and a hop constraint between the root vertex and every other vertex in the tree. They propose a greedy algorithm for initial solution followed by destroy-and-repair or tabu search to improve the initial solution.

Kim et al. [16] studied the Delay and Delay Variation Constrained multicasting Steiner Tree Problem. The problem is similar to the one studied by Voss, with a delay constraint instead of the hop constraint, and a constraint on delay variation between two sources. With the delay variation constraint relaxed, Kim’s problem becomes the Rooted Steiner Tree-Minimum Weight-Delay Constraint problem. They proposed a polynomial time heuristic algorithm to obtain feasible solutions, but they also did not provide any performance guarantee for their algorithm.

Bredin et al. [17] studied the problem of optimal relay placement (unconstrained) for \( k \)-connectivity. They proposed an \( O(1) \) approximation algorithm for the problem with any fixed \( k \geq 1 \).

Misra et al. [6] studied the constrained relay placement problem for connectivity and survivability. They provided \( O(1) \) approximation algorithms for both the problems. We can call their first problem the Rooted Steiner Tree-Minimum Relays problem, and their second problem, the Rooted Steiner Tree-Minimum Relays-Survivability problem. Although their formulation takes into account an edge length bound, namely edge length \( \leq r_c \), which can model the link quality, the formulation does not involve a path constraint such as the hop count along the path; hence, there is no constraint on the end-to-end delay.

In Table I we present a brief comparison of the problem under study with some of the closely related problems studied in the literature.

### IV. SPTIRP: A Heuristic for RST-MR-HC Problem and Its Analysis

#### A. Shortest Path Tree (SPT) based Iterative Relay Pruning Algorithm (SPTIRP)

1) **The Zero Relay Case**: Find the SPT on \( Q \) alone, rooted at the sink. If the hop count \( \leq h_{\text{max}} \) for each path, we are done; no relays are required in an optimal solution. Else, go to the next step.

2) Find the Shortest Path Tree \( T \) on \( G \), rooted at the sink.

3) **Checking Feasibility**: If for any path in the SPT, the path weight exceeds \( h_{\text{max}} \), declare the problem infeasible. (Clearly, if the shortest path from a node to the sink does not meet the hop count bound, no other path from the node to the sink will meet the hop count bound). Else go to the next step.

**Pruning the SPT**:

4) Discard all nodes in \( R \setminus T \). Note that this step may lead to suboptimality as some nodes in \( R \setminus T \) could be part of an optimal solution.

5) Now, for the remaining relay nodes in \( R \), define the weight of a relay node as the number of paths in the SPT that use the node.

6) Arrange the paths in SPT in increasing order of hop count.

7) Choose the least cost path that contains relay nodes. Arrange the relay nodes on this path in increasing order of their weights as defined in (5).

8) Remove the least weight relay node and consider the restriction of \( G \) to the remaining nodes in \( T \). Find an SPT on this graph. If in this SPT, path cost exceeds \( h_{\text{max}} \) for any path, then discard this SPT, replace the removed relay node, and repeat this step with the next least weight relay node. If all the relays in the least cost path have been tried without success, move on to the next least cost path, and repeat steps 7 and 8 for the relays in this path that have not yet been tried.

9) If in the above step, the SPT obtained satisfies the delay constraint for all the paths, then delete the removed relay node permanently from \( R \) and repeat steps 4 through 9.

10) Stop when no more relay pruning is possible without violating the hop constraint on one or more of the paths.

### Discussion:

---

**Table I**

| Problem          | End-to-End Performance Objective | Complexity       | Approximation Guarantee of Proposed Algorithm |
|------------------|---------------------------------|------------------|---------------------------------------------|
| RST-MR [6]       | ×                               | NP-Hard          | ×                                           |
| RST-MW-HC [14]   | ✓                               | NP-Hard          | ×                                           |
| RST-MW-MR [16]   | ✓                               | NP-Hard          | polynomial factor                           |
| RSN-R-MR-HC <    | ✓                               | NP-Hard          | polynomial factor for a subclass of problems |
Step 1 of the above algorithm ensures that if the optimal design does not use any relay node, then the same goes true for our algorithm. That way we can make sure that the algorithm does not do infinitely worse in the sense that $\frac{\text{RelayGain}}{\text{RelayGain}_{\text{opt}}}$ is finite.

The idea behind Steps 7, 8 and 9 is that choosing to remove a relay from the path with most slack in cost (i.e., delay or hop constraint), we stand a better chance of still meeting the delay requirement with the remaining relays. Also, removing a relay of less weight would mean affecting the cost of a small number of paths. So by pruning relays in the manner as described in Steps 7, 8 and 9, we aim for a better exploration of the search space.

B. Analysis of the Algorithm

1) Complexity: The complexity of determining the shortest path tree on $N$ nodes is $N \log N$ [18]. Let us denote this function by $g_{\text{SPT}}()$. In iteration 1 of the algorithm, the complexity is $g_{\text{SPT}}(|Q|)$ and in iteration 2, it is $g_{\text{SPT}}(|Q| + |R|)$. In subsequent iterations, we remove 1 relay node at a time and find the SPT on the resultant complete graph; if no improvement is found, we replace that node and continue. Thus, for the $k^{th}$ iteration, the worst case complexity will be $(|R| - k + 3)g_{\text{SPT}}(|Q| + |R| - k + 2)$, where in the worst case, $k = 3, 4, \ldots, |R| + 1$. Let $g(.)$ denote the overall complexity. Thus, the overall complexity will be

$$g_{\text{spt}}(|Q| + |R|) = g_{\text{SPT}}(|Q| + |R|) + \sum_{j=1}^{|R|}(g_{\text{SPT}}(|Q| + |R| - j))(|R| - j + 1) \leq (1 + |R|^2)(g_{\text{SPT}}(|Q| + |R|))$$

which is polynomial time.

2) Worst Case Approximation Factor: The worst case occurs when the SPT obtained before we enter Step (4) does not contain any relay node(s) that correspond to some optimal design. If no relays are used in any optimal design, then the algorithm will yield an optimal design (Step 1). Hence, the worst possibility is that the optimal design uses just 1 relay node, whereas the SPT obtained in Step (2) consists of all the remaining $(|R| - 1)$ relays, and moreover, pruning any of these $(|R| - 1)$ relays will cause one or more paths in the resulting SPT to violate the hop constraint. Thus, in the worst case, the algorithm leads to a design with $(|R| - 1)$ relays instead of the optimal design with one relay. Hence, we have a polynomial factor worst case approximation guarantee of $(|R| - 1)$.

3) Sharp Examples: Let us now present a sequence of problems of increasing complexity for which the approximation guarantee is strict, i.e., for these problems, the algorithm ends up using $(|R| - 1)$ relays, while the optimum design uses one relay. Such examples are worthwhile to explore as they help to show that the approximation factor obtained above cannot be improved. Consider the situation shown in Figure 4. The green hexagons denote the relay node locations and the black circles represent the source node locations. Only the edges shown (coloured or black) are permitted. Consider the RST-MR-HC problem on this graph with $h_{\text{max}} = 3$. Clearly, the optimal solution will use only one relay, $R_1$, to reach from each source to the BS within the specified hop count bound. The red links correspond to the optimum solution. The red link will belong to both the optimum solution and the outcome of our algorithm as it is a direct link between source $S_1$ and the BS. Our SPT based algorithm will calculate the shortest paths and thus end up using relays $R_2, R_3, \ldots, R_n$, leaving out $R_1$. The black solid links correspond to the solution given by our algorithm. Clearly, in such problems, we end up using $(|R| - 1)$ relays instead of just one.

Another sequence of problems of increasing complexity for which the algorithm gives the optimal design can be constructed as shown in Figure 5. Such examples help to show that a proposed algorithm does provide an optimal solution in some scenarios.
As before, the green hexagons represent relay locations and the black dots represent source nodes. Suppose \( h_{\text{max}} = 2 \). Then clearly, the optimal solution is as shown in the figure. The algorithm, after calculating the SPT, will end up with the same solution.

V. SPTiRP: NUMERICAL RESULTS

To test the algorithm, we generated 1000 random networks as follows: A 150m \( \times \) 150m area is partitioned into square cells of side 10m. Consider the lattice created by the corner points of the cells. 10 source nodes are placed at random over these lattice points. Then the potential relay locations are obtained by selecting \( n \) points uniformly randomly over the 150m \( \times \) 150m region; \( n \) was varied from 100 to 140 in steps of 10, and for each value of \( n \), we generated 200 random network scenarios (thus yielding 1000 test cases). We chose \( r_{\text{max}} = 60 \) meters, and \( h_{\text{max}} = 6 \) for the experiments.

Given the outcome of the SPT based algorithm, an optimal solution can be obtained as follows:

Suppose the SPT based algorithm uses \( n \) relays. Then perform an exhaustive search over all possible combinations of \((n-1)\) and fewer relays to check if the performance constraints can still be met.

In none of the 1000 scenarios tested, the hop constraint turned out to be infeasible. The results are summarized in Table II.

| Potential Relay count | Scenarios | Optimal Design | Off by one | Max off from optimal |
|-----------------------|-----------|----------------|------------|----------------------|
| 100                   | 200       | 154            | 42         | 3                    |
| 110                   | 200       | 154            | 40         | 2                    |
| 120                   | 200       | 158            | 39         | 2                    |
| 130                   | 200       | 155            | 36         | 2                    |
| 140                   | 200       | 161            | 38         | 2                    |
| Total                 | 1000      | 782            | 195        | 3                    |

The efficiency of the algorithm can be easily visualized from the pie chart in Figure 6.

Observations

1) In over 97% of the tested scenarios, the algorithm ends up giving optimal or near-optimal (exceeding optimum just by one relay) solutions.

2) In the remaining cases, where it is off by more than one relay, the maximum difference was found to be 3 relays.

In Table III, we have compared the execution time of the SPTiRP algorithm against the time required to compute an optimal solution, given the outcome of the SPTiRP algorithm. Both the SPTiRP algorithm, and the postprocessing on its outcome were run in MATLAB 7.0.1 on a Windows Vista (basic) based PC (Dell Inspiron 1525) having Intel Core 2 Duo T5800 CPU with processor speed of 2 GHz, and 3 GB RAM. As can be seen from the table, while the SPTiRP algorithm computes a very good (often optimal) solution in at most a second or two (averaging less than a second), computing the optimal solution even after being provided with a very good upper bound on the required number of relays by SPTiRP, can be actually quite time consuming, running into minutes.

Also, we note from Table III that, as the node density increases, the computation time of the SPTiRP algorithm also increases.

VI. COMPLEXITY OF OBTAINING FEASIBLE SOLUTION FOR RSNk-MR-HC

Lemma 1. Given a graph \( G = (V, E) \), specified vertices \( s \) and \( t \), positive integers \( k \geq 1 \), and \( H \subseteq |V| \). The problem to determine if \( G \) contains \( k \) or more mutually vertex disjoint paths from \( s \) to \( t \), none involving more than \( H \) edges is NP-Complete for all fixed \( H \geq 5 \).

Proof: See [Itai, Perl, and Shiloach, 1977]. The proof there is via transformation from 3 Satisfiability.

Corollary 1. Given an edge weighted graph \( G = (V, E) \), with \( V = Q \cup R \), the problem of finding a subgraph with \( k \) vertex disjoint paths from each source node to sink such that each path has hop count \( \leq h_{\text{max}} \) (RSNk-HC) is NP-Complete.

Proof: From Lemma 1 it follows that this problem is NP-Complete for any \( h_{\text{max}} \geq 5 \). Hence, the general problem is NP-Complete.

Corollary 2. Unless \( P = NP \), there does not exist any polynomial time complexity algorithm for providing a feasible solution to the RSNk-MR-HC problem.
Corollary 3. Unless $P = NP$, no polynomial time complexity algorithm can provide finite approximation guarantee for the general RSN-$k$-MR-HC problem.

Proof: Suppose, in a certain instance of the RSN-$k$-MR-HC problem, the source nodes alone are sufficient to meet the design requirements, i.e., there exist node disjoint hop constrained paths from each source to the sink, involving only other source nodes, and no additional relay nodes. But, from Corollary 1 it follows that no polynomial time algorithm is guaranteed to predict the existence of such a solution even when there exists one.

Therefore (unlike the SPTiRP algorithm which uses zero relays whenever the optimal solution uses zero relays), in this case, one might end up using a non zero number of relay nodes despite the fact that the optimal solution uses zero relays. Hence, for the general RSN-$k$-MR-HC problem, a polynomial time algorithm cannot provide finite approximation guarantee.

VII. POLYNOMIAL TIME HEURISTICS FOR RSN-$k$-MR-HC

We propose two different algorithms for the RSN-$k$-MR-HC problem, both of which build on the SPTiRP algorithm for one connectivity, described in Section IV. Before we discuss the algorithms, we describe below a few limitations that are common to both of the proposed algorithms as well as any other polynomial time algorithm for RSN-$k$-MR-HC problem.

Recall from the proof of Corollary 3 that no polynomial time algorithm for the RSN-$k$-MR-HC problem is guaranteed to predict the existence of a solution involving only the source nodes whenever such a solution exists.

Also, from Corollary 2 it follows that unless $P = NP$, no polynomial time algorithm for the RSN-$k$-MR-HC problem is guaranteed to find a feasible solution whenever there exists one. Therefore, if a polynomial time heuristic for the RSN-$k$-MR-HC problem fails to find a feasible solution, we shall say that the problem is possibly infeasible.

### TABLE III

| Potential Relay Count | Mean execution time of SPTiRP in sec | Max execution time of SPTiRP in sec | Max execution time to obtain an Optimal Solution in sec |
|-----------------------|--------------------------------------|-----------------------------------|--------------------------------------------------------|
| 100                   | 0.38812                              | 1.038                            | 1828.7                                                 |
| 110                   | 0.70544                              | 2.081                            | 722.29                                                 |
| 120                   | 0.81154                              | 1.591                            | 944.74                                                 |
| 130                   | 0.99343                              | 2.606                            | 2674.9                                                 |
| 140                   | 1.1438                               | 2.808                            | 355.46                                                 |
| Overall 1000          | 0.84847                              | 2.808                            | 2674.9                                                 |

### A. The Main Idea/Key Steps in the Algorithms

- Given $G = (V, E)$, where $V = Q \cup R$, connectivity requirement $k$, and hop constraint $h_{max}$
- Check for $k$ connectivity with hop constraint on $Q$ alone
  - If the answer to this step is positive, done
  - Else go to the next step
- Run the SPTiRP algorithm on the entire graph of sources and potential relay locations, to obtain a one connected hop constrained network with a small number of relays. If this step completes successfully, we get a hop constrained path from each source to the sink.
- If the SPTiRP algorithm returns failure, we can declare the problem to be infeasible, and stop, as we could not even obtain a one connected network satisfying the hop constraint.
- Next, for each source, we try to obtain an alternate node disjoint hop count feasible path to the sink.
- If we fail to find an alternate hop count feasible node disjoint route for some of the sources, we declare the problem to be possibly infeasible, and stop.
- If we can find an alternate hop count feasible node disjoint route from the source to the sink, we start with that feasible solution, and try to prune relays, while retaining hop count feasibility, in order to obtain a better solution in terms of relay count.
  - The two algorithms presented here differ in their approach to this relay pruning procedure
- This procedure is repeated until all the sources have $k$ node disjoint hop constrained paths to the sink, or the problem has been declared (possibly) infeasible.

### B. Algorithm 1

We present the pseudo code of the algorithms (in the boxes), interspersed with Remarks that help to explain the immediately preceding steps.

1) **Phase 1: Checking for $k$-connectivity on $Q$ alone:**

In this phase, we shall check if the design objectives ($k$ connectivity with hop constraint) can be met using
only the source nodes, and no additional relays. In other words, we aim at finding \( k \) node disjoint hop constrained paths from each source to the sink, using only other source nodes.

| **Input:** \( Q \) = \((Q, E_Q), h_{\text{max}}, k \) |
| **comment:** \( E_Q \) is the set of all edges of length \( \leq r_{\text{max}} \) on \( Q \) |

| **Output:** \( T \) (the desired network) |

| **flags:** boolean \( F_{\text{inf}} \) (if \( F_{\text{inf}} = 1 \), no feasible solution found) |

| **Initialize:** \( T = \emptyset \), \( F_{\text{inf}} = 0 \) |

**Outer loop:** for each source \( S_i \), \( 1 \leq i \leq |Q| - 1 \)

| **Step 1:** \( Q = \{S_i, 0\}; R = Q \setminus Q \) |

**Remark:** For each source, we treat all the other sources as relays (the set \( R \)), and try to obtain \( k \) node disjoint hop constrained paths from the source to the sink, using the nodes in \( R \).

**Step 2:** \( l = 1 \) (\( l \) is the loop variable for the inner loop, described next)

**Inner loop:** while \( l \leq k \) (**comment:** the following steps will be repeated until we have \( k \) node disjoint paths from source \( i \) to sink)

| **Step 3:** \( \text{path}_l(S_i, 0) \leftarrow \text{SPTiRP}(Q \cup R, E_Q) \) |

**Remark:** Treating the remaining sources as relays, we run the SPTiRP algorithm to obtain the \( l^{th} \) node disjoint path (\( \text{path}_l(S_i, 0) \)) from source \( i \) to sink; the reason for using the SPTiRP algorithm is to use as few nodes as possible from the set \( R \), so that there are enough nodes left to construct the \( (l + 1)^{th} \) node disjoint path in the next iteration of the inner loop.

| **Step 4:** if hopcount(\( \text{path}_l(S_i, 0) \)) > \( h_{\text{max}} \)  
\( F_{\text{inf}} \leftarrow 1 \);  
exit Phase 1  
else go to next Step |

**Remark:** Note that if the hop constraint cannot be met in the first iteration (of the inner loop) itself (i.e., for \( l = 1 \)), it implies that the shortest path from source \( i \) to sink using only the other source nodes does not satisfy the hop constraint. Then, we can conclude for sure that the design objectives cannot be met using only the source nodes, and we can proceed to Phase 2 of the algorithm.

However, if the hop constraint is met in the first iteration, and cannot be satisfied in some subsequent iteration (i.e., for some \( l > 1 \)), we cannot conclude for sure that \( Q \) alone was not sufficient to meet the design requirements (recall Corollary [1] and our discussion at the beginning of Section VII [1]). All we can say at this point is that Phase 1 of our algorithm failed to find a feasible solution on \( Q \) alone, and therefore, we shall proceed to the next phase of the algorithm, assuming that the problem on \( Q \) alone is possibly infeasible.

| **Step 5:** \( T \leftarrow T \cup \text{path}_l(S_i, 0) \) |

**comment:** We augment the network with the current feasible path.

| **Step 6:** \( R_{\text{used}} \leftarrow R \cap \text{path}_l(S_i, 0) \)  
\( R \leftarrow R \setminus R_{\text{used}} \)  
l \leftarrow l + 1 |

**Remark:** We identify the nodes in \( R \) used in the current path from source \( i \) to sink, and remove them from \( R \) before proceeding to the next iteration of the inner loop.

At the end of Phase 1, we either have a network \( T \), consisting only of the source nodes, and meeting the design requirements (in which case, we are done), or we find that the problem on \( Q \) alone is possibly infeasible (\( F_{\text{inf}} = 1 \)), in which case, we proceed to Phase 2 of the algorithm.

2) **Phase 2: Obtaining node-disjoint paths from each source to the sink with a small relay count:** We come to this phase if Phase 1 fails to find a network on \( Q \) alone satisfying the design objectives. Our objective in this phase is to obtain \( k \) node disjoint hop constrained paths from each source to the sink, using as few additional relays as possible. To that end, we proceed as explained in Section VII-A

| **Input:** \( G = (V, E), h_{\text{max}}, k \) |
| **comment:** \( V = Q \cup R \), and \( E \) is the set of all edges of length \( \leq r_{\text{max}} \) on \( Q \cup R \). |

| **Output:** \( T \) (the desired network) |

| **flag:** boolean \( F_{\text{inf}} \) (if \( F_{\text{inf}} = 1 \), problem is (possibly) infeasible) |

| **Initialize:** \( T = \emptyset \), \( F_{\text{inf}} = 0 \) |

| **Step 1:** \( (T, F_{\text{inf}}) \leftarrow \text{SPTiRP}(G) \) |

**Remark:** We run the SPTiRP algorithm on \( G \) to obtain a one connected hop constrained network with as few relays as possible.

| **Step 2:** if \( F_{\text{inf}} = 1 \)  
exit Phase 2  
else go to next step |

**Remark:** If the SPTiRP algorithm fails to meet the hop constraint for some of the sources, we declare the problem to be infeasible, and stop.
Step 3: \( n = 2 \) (comment: \( n \) is the loop variable for the outer loop described next)

**Outer loop:** while \( n \leq k \) (comment: the following steps (Steps 4 to 12) will be repeated until all the sources have \( k \) node disjoint hop constrained paths to the sink)

**Inner loop:** for each source \( S_i \in Q, 1 \leq i \leq \left| Q \right| - 1 \) (comment: Steps 4 to 11 will be repeated for each source)

Step 4: \( V^{\text{used}} \leftarrow V \cap \{ \cup_{i=1}^{n-1} \text{path}_i(S_i, 0) \} \)

**Remark:** We designate by \( \text{path}_i(S_i, 0) \), the \( i^{\text{th}} \) node disjoint hop constrained path from source \( S_i \) to sink (\( 1 \leq i \leq k \)). In Step 4, we identify the set of nodes (designated by \( V^{\text{used}} \)) used by the first \( n - 1 \) node disjoint paths from source \( S_i \) to sink. Since we want to find another node disjoint path from source \( S_i \) to sink in the current \((n^{\text{th}})\) iteration (of the outer loop), we need to remove the set of vertices, \( V^{\text{used}} \), except \( S_i \) and 0, from consideration for the \( n^{\text{th}} \) path before proceeding further in the current iteration. We do that in the next step.

Step 5: \( V^{(n)}_i \leftarrow \{ V \setminus V^{\text{used}} \} \cup \{ S_i, 0 \} \)

**Remark:** \( V^{(n)}_i \) is the set of vertices not used in the first \( n - 1 \) paths from source \( S_i \) to sink, and therefore, eligible to be part of the \( n^{\text{th}} \) node disjoint path from source \( S_i \) to sink.

Step 6: \( G^{(n)}_i \leftarrow \text{restriction of } G \text{ to } V^{(n)}_i \)

**Remark:** In order to obtain the \( n^{\text{th}} \) node disjoint path from source \( S_i \) to sink, we restrict the graph \( G \) to the eligible node set \( V^{(n)}_i \).

Step 7: \( \text{path}^{(n)}_i(S_i, 0) \leftarrow \text{SPT}(G^{(n)}_i) \)

Step 8: If hopcount(\( \text{path}^{(n)}_i(S_i, 0) \)) > \( h_{\text{max}} \)$\{\( F_{\text{inf}} \leftarrow 1; \) \( \) exit Phase 2 \} else go to next Step

**Remark:**
1) We obtain the shortest path (designated by \( \text{path}^{(n)}_i(S_i, 0) \)) from source \( S_i \) to sink in \( G^{(n)}_i \); if this path meets the hop constraint, we have a feasible solution for the \( n^{\text{th}} \) node disjoint path from source \( S_i \) to sink, and we can proceed to the next step to prune relays from the feasible solution in order to achieve a better solution in terms of relay count. If, however, the path obtained in Step 7 does not meet the hop constraint, we declare the problem to be possibly infeasible, and stop, as we have failed to obtain a feasible solution for the \( n^{\text{th}} \) node disjoint path (\( 1 < n \leq k \)) from source \( S_i \) to sink.

2) Successful completion of Steps 7 and 8 thus guarantee the existence of a hop count feasible, node disjoint path from source \( S_i \) to sink. Now, since our objective is to meet the design requirements using as few relays as possible, in the next step, we shall try to obtain a better path (in terms of relay count) from source \( S_i \) to sink, by pruning relays from the feasible path obtained earlier.

3) Note that we can simply run the SPTiRP algorithm on the graph \( G^{(n)}_i \) to obtain the \( n^{\text{th}} \) node disjoint hop constrained path from source \( S_i \) to sink, update the network \( T \) with that path, and move on to the next source (next iteration of the inner loop). But since the SPTiRP algorithm is designed to ‘optimize’ the total number of relays used by all the source nodes in \( G^{(n)}_i \), the path so obtained from source \( S_i \) to sink may not be the best in terms of relay count for source \( S_i \). Hence, we shall explore other methods of reducing relay count in the path from source \( S_i \) to sink, as indicated in Step 9.

Step 9: \( \text{path}^{(1)}_i(S_i, 0) \leftarrow \text{SPTiRP}(G^{(n)}_i) \) (comment: Candidate path 1)

\( \text{path}^{(2)}_i(S_i, 0) \leftarrow \text{RoutineA}(G^{(n)}_i, \text{path}^{(n)}_i(S_i, 0), R) \) (comment: Candidate path 2)

**Remark:** We obtain two candidate routes for the \( n^{\text{th}} \) node disjoint hop constrained path from source \( S_i \) to sink in \( G^{(n)}_i \). The first candidate path is obtained simply by running the SPTiRP algorithm on the graph \( G^{(n)}_i \). The second candidate route is obtained by pruning relays from the feasible path, \( \text{path}^{(n)}_i(S_i, 0) \), obtained in Steps 7 and 8 earlier. The routine for this relay pruning procedure, namely RoutineA, is described next. Once we have the candidate routes, we shall choose the best among them in terms of relay count.

**Pseudo code for RoutineA**

Input: \( G^{(n)}_i, \text{path}^{(n)}_i(S_i, 0), R \)

Output: \( \text{path}^{(2)}_i(S_i, 0) \) (comment: the second candidate path)

Initialize: \( \text{path}^{(2)}_i(S_i, 0) \leftarrow \text{path}^{(n)}_i(S_i, 0) \)

Step a: \( Q_i \leftarrow V^{(n)}_i \cap Q \)

\( R_i \leftarrow R \cap \text{path}^{(n)}_i(S_i, 0) \)

\( V^{(n)}_i \leftarrow Q_i \cup R_i \)

\( G^A_i \leftarrow \text{restriction of } G^{(n)}_i \text{ to } V^{(n)}_i \) (comment: This is the graph over which we shall search for a hop constrained path after pruning a relay)

**Remark:** \( R^A_i \) is the set of relays used in the feasible path, \( \text{path}^{(n)}_i(S_i, 0) \), from source \( S_i \) to sink. We shall try to prune the relays in the set \( R^A_i \) one by one to achieve a better path in terms of relay count.
$Q_i^A$ is the set of sources (may or may not be used in the initial feasible path, $path_n^b(S_i,0)$) belonging to the eligible vertex set, $V_i^n$, defined earlier in Step 5 of Algorithm 1, phase 2. We designate by $V_i^A$, the vertex set consisting of the sources in $Q_i^A$, and the relays in $R_i^A$.

We shall allow our search space to be the vertex set $V_i^A$ (i.e., the graph $G_i^A$), i.e., we shall allow all the eligible source nodes in $V_i^n$, irrespective of whether they were used or not in the initial feasible path, and allow only the relays used in the initial feasible path, $path_n^b(S_i,0)$.

- Since we are allowing only the relays used in the initial feasible path, this method will still ensure a reduction in relay count, if a hop constrained path is found after pruning a relay.
- In the worst case, the routine may end up with $path_n^b(S_i,0)$ as outcome.

**Step b: Loop:** for each node $j \in R_i^A$ (comment: the following steps will be repeated until no more relay pruning is possible without violating the hop constraint)

**Step c:** TempPath($S_i,0$) ← $SPT(G_i^A\setminus j)$

Remark: With slight abuse of notation, we designate by $G_i^A\setminus j$, the restriction of the graph $G_i^A$ to the node set $V_i^A\setminus j$. After pruning a relay $j$, we obtain the shortest path (TempPath($S_i,0$)) from source $S_i$ to sink, using the remaining vertices in $V_i^A$.

**Step d:** if hopcount(TempPath($S_i,0$)) > $h_{\text{max}}$ continue; (comment: Go back to step 2 and try pruning the next relay in $R_i^A$)

else go to next Step

Remark: If the shortest path from $S_i$ to sink in $G_i^n$ after pruning relay $j$ does not satisfy the hop constraint, we replace back the relay $j$, and try pruning the next relay in $R_i^A$.

**Step e:** $path_n^b(2)(S_i,0) \leftarrow$ TempPath($S_i,0$)

$R_i^A \leftarrow R_i \cap path_n^b(2)(S_i,0)$

$V_i^A \leftarrow Q_i^A \cap R_i^A$

$G_i^A \leftarrow$ restriction of $G_i^n$ to $V_i^A$

Remark: If relay $j$ can be pruned successfully without violating the hop constraint, we update the candidate path, $path_n^b(2)(S_i,0)$, and the sets $V_i^A$ and $R_i^A$ (the set of relays used in the candidate path) as above before proceeding to the next iteration of the loop (i.e., before pruning the next relay).

Note that since relay $j$ has been pruned, the updated relay set $R_i^A$ (and hence the updated candidate path) will have at least one relay less than the relay set used by the initial feasible path, $path_n^b(1)(S_i,0)$.

**end of Pseudo code for RoutineA**

**Step 10:** (of Algorithm 1, phase 2):

$loop_{path_n}(S_i,0) \leftarrow \arg \min \{\text{relaycount}(path_n^b(1)(S_i,0)),\$

$\text{relaycount}(path_n^b(2)(S_i,0))\}$

Remark: Once we have the candidate routes for the $n^{th}$ node disjoint hop constrained path from source $S_i$ to sink, we choose the best among them in terms of relay count.

**Step 11:** $T \leftarrow T \cup path_n(S_i,0)$

Inner loop end

**end of Pseudo code for Algorithm 1**

While Algorithm 1 performs quite well in terms of relay count as compared to the worst case theoretical performance bound (as we shall see in Section VIII), it has certain limitations, as mentioned below.

**Limitations of the Algorithm 1**

While finding alternate node disjoint route for a source, we did not put any emphasis on the reuse of

- relays that are already part of the one connected network that is obtained in Step 1 of Phase 2
- relays that have been used to construct alternate routes for the previous sources

Such reuse of relays could reduce the overall relay count. With these in mind, we proceed to design Algorithm 2, which reuses some of the key ideas of Algorithm 1 (and hence borrows some of the initial steps of Algorithm 1), while trying to overcome the limitations of Algorithm 1.

**C. Algorithm 2:**

1) **Phase 1: Checking for $k$-connectivity on $Q$ alone:**

- We aim at finding $k$ node disjoint hop constrained paths from each source to the sink, using only other source nodes.
- The procedure, and the conclusions are the same as in Algorithm 1.
- If hop constraint is met for all sources, we are done. No relays required for $k$-connectivity. Else, proceed to the next phase.

2) **Phase 2: Obtaining node-disjoint paths from each source to the sink with a small relay count:** We come to this phase if phase 1 fails to find a network on $Q$ alone.
satisfying the design objectives. Our objective in this phase is to obtain $k$ node disjoint hop constrained paths from each source to the sink, using as few additional relays as possible. To that end, we proceed as explained in Section VII-A.

We present below, the detailed pseudo code for this phase, along with necessary remarks, and explanations.

**Input:** $G = (V, E)$, $h_{\text{max}}$, $k$

**Output:** $T$ (the desired network)

**flag:** boolean $F_{\text{inf}}$ (if $F_{\text{inf}} = 1$, problem is (possibly) infeasible)

**Initialize:** $T = \emptyset$, $F_{\text{inf}} = 0$

Step 1: $(T, F_{\text{inf}}) \leftarrow \text{SPTiRP}(G)$

**Remark:** We run the SPTiRP algorithm on $G$ to obtain a one connected hop constrained network with as few relays as possible.

Step 2: if $F_{\text{inf}} = 1$

- exit Phase 2

else go to the next step

**Remark:** If the SPTiRP algorithm fails to meet the hop constraint for some of the sources, we declare the problem to be infeasible, and stop. Otherwise, we proceed to find alternate node disjoint hop constrained paths from each of the sources to the sink, as below.

Step 3: $Q \leftarrow \text{sort}(Q)$ in decreasing order of Euclidean distance from the sink

**Remark:** We arrange the sources in decreasing order of their Euclidean distances from the sink; we shall start the alternate route determination procedure with the farthest source and proceed in that order. The logic behind this approach is as follows:

- Farther sources are likely to consume more relays.
- So meet their need first.
- As we move closer to the sink, coax other sources to share the already used relays.

Step 4: $n = 2$ (comment: $n$ is the loop variable for the outer loop to be defined next; $n$ keeps track of the number of node disjoint paths discovered, including current iteration)

**Outer loop:** while $n \leq k$ (comment: the following steps will be repeated until all the sources have $k$ node disjoint hop constrained paths to the sink)

Step 5: $L^{(n)} \leftarrow R \cap T$

Remark: In the $n^{th}$ iteration of the outer loop, we shall try to obtain the $n^{th}$ node disjoint, hop constrained path from each source to the sink, using as few relays as possible.

We define a Locked set $L^{(n)}$ as the set of relays used so far in the course of the network design algorithm (and hence, are part of the final desired network). For example, at the start of the $n^{th}$ iteration ($n = 2, \ldots, k$), $L^{(n)}$ consists of the relays used in the first $(n - 1)$ node disjoint paths from each of the sources to the sink.

We also define a free relay set, $R$, as the set of relays not used so far in the network design.

Therefore, in our attempt to minimize the number of additional relays used, we shall try to reuse relays from the Locked set $L^{(n)}$ whenever possible, and try to minimize the use of relays from the free relay set. With this in mind, we proceed to obtain the alternate node disjoint paths from each source to the sink as below.

Step 6:

**Inner loop:** for each source $S_i \in Q$, $1 \leq i \leq |Q| - 1$ (comment: The following steps will be repeated for each source, starting with the source farthest from the sink)

Step 7: $V^{\text{used}} \leftarrow V \cap \{\bigcup_{i=1}^{n-1} \text{path}_i(S_i, 0)\}$

**Remark:** Recall from Algorithm 1 that we designate by $\text{path}_i(S_i, 0)$, the $l^{th}$ node disjoint hop constrained path from source $S_i$ to sink $(1 \leq l \leq k)$. In Step 7, we identify the set of nodes (designated by $V^{\text{used}}$) used by the first $n - 1$ node disjoint paths from source $S_i$ to sink. Since we want to find another node disjoint path from source $S_i$ to sink in the current ($n^{th}$) iteration (of the outer loop), we need to remove the set of vertices, $V^{\text{used}}$, except $S_i$ and 0, from consideration for the $n^{th}$ path before proceeding further in the current iteration. We do that in the next step.

Note that Steps 7 to 11 are exactly same as the Steps 4 to 8 in Phase 2 of Algorithm 1. Hence, to avoid repetition, we shall not go into detailed explanation of these steps. At the end of these steps, we shall either have a feasible solution for the $n^{th}$ node disjoint path from source $S_i$ to sink, or we shall end up with possible infeasibility.

Step 8: $V_i^{(n)} \leftarrow \{V \setminus V^{\text{used}}\} \cup \{S_i, 0\}$

**Remark:** $V_i^{(n)}$ is the set of vertices not used in the first $n - 1$ paths from source $S_i$ to the sink, and therefore, eligible to be part of the $n^{th}$ node disjoint path from source $S_i$ to sink.

Step 9: $G_i^m \leftarrow \text{restriction of } G \text{ to } V_i^{(n)}$

**Remark:** In order to obtain the $n^{th}$ node disjoint path from source $S_i$ to sink, we restrict the graph $G$ to the eligible node set $V_i^{(n)}$. 
Step 10: \( \text{path}^{\text{sh}}_n(S_i, 0) \leftarrow \text{SPT}(G_i^n) \)

Step 11: if hopcount(\( \text{path}^{\text{sh}}_n(S_i, 0) \)) > \( h_{\text{max}} \)
   \{ \text{Finf} \leftarrow 1; \text{exit Phase 2} \}
   \text{else go to the next Step} \\

Remark:

1) If \( \text{path}^{\text{sh}}_n(S_i, 0) \), the shortest path from source \( S_i \) to sink in \( G_i^n \), does not meet the hop constraint, we declare the problem to be possibly infeasible, and stop, as we have failed to obtain a feasible solution for the \( n^{th} \) node disjoint path (\( 1 < n \leq k \)) from source \( S_i \) to sink. If hop constraint is satisfied by \( \text{path}^{\text{sh}}_n(S_i, 0) \), we have a feasible solution for the \( n^{th} \) node disjoint path from source \( S_i \) to sink, and we proceed to the next step to prune relays from this feasible solution in order to achieve a better solution in terms of relay count.

Algorithm 2 will differ from Algorithm 1 in this relay pruning procedure.

Step 12: \( L_i^{(n)} \leftarrow L_i^{(n)} \cap V_i^{(n)} \)
\( Q_i^{(n)} \leftarrow Q \cap V_i^{(n)} \)

Remark: We designate by \( L_i^{(n)} \), the members of the locked relay set that are eligible to be part of the \( n^{th} \) node disjoint path from source \( S_i \) to sink. Similarly, \( Q_i^{(n)} \) denotes the set of source nodes that are part of the eligible node set \( V_i^{(n)} \). Note that, apart from the sets \( L_i^{(n)} \) and \( Q_i^{(n)} \), the only other component of the eligible node set \( V_i^{(n)} \) is the free relay set, \( R \), i.e., \( V_i^{(n)} = L_i^{(n)} \cup Q_i^{(n)} \cup R \).

Now, in our attempt to minimize the number of relays used, we shall try to prune one at a time, the free relays (i.e., relays in \( R \)) that are part of the initial feasible path, \( \text{path}^{\text{sh}}_n(S_i, 0) \), and try to obtain an alternate hop constrained path, reusing the relays in the eligible locked set \( L_i^{(n)} \) as much as possible. We explain this procedure in the next steps.

Step 13: \( \overline{R}_i \leftarrow R \cap \text{path}^{\text{sh}}_n(S_i, 0) \)

Remark: We identify as \( \overline{R}_i \), the set of free relays that are part of the initial feasible solution, \( \text{path}^{\text{sh}}_n(S_i, 0) \), for the \( n^{th} \) node disjoint path from source \( S_i \) to sink. To reduce the relay count, we shall try to prune the relays in \( \overline{R}_i \) one at a time, while maintaining the hop constraint.

Step 14: \( V_i^{(n)} \leftarrow Q_i^{(n)} \cup L_i^{(n)} \cup \overline{R}_i \)
\( G_i^{(n)} \leftarrow \text{restriction of } G_i^{(n)} \text{ to } V_i^{(n)} \) [comment: After pruning a relay from the initial feasible path, we shall search for a hop constrained path over this graph, and not over \( G_i^{(n)} \).]

Remark: After pruning a “free” relay from the feasible path, \( \text{path}^{\text{sh}}_n(S_i, 0) \), we shall search for a better path (in terms of relay count) using only the remaining free relays in \( \text{path}^{\text{sh}}_n(S_i, 0) \), and the locked relays and sources in \( V_i^{(n)} \), irrespective of whether they are part of \( \text{path}^{\text{sh}}_n(S_i, 0) \). In other words, we shall restrict our search space for the hop constrained path to the graph \( G_i^{(n)} \), thereby disallowing the use of free relays that are not part of \( \text{path}^{\text{sh}}_n(S_i, 0) \), while still allowing the use of (eligible) locked relays and sources even if they are not part of the initial feasible path. The idea behind this selection of search space is as follows:

- Restricting the search space to only the free relays in \( \text{path}^{\text{sh}}_n(S_i, 0) \) ensures reduction in relay count, if we can find a hop constrained path after relay pruning.
- Since the relays in \( L_i^{(n)} \), and the sources \( Q_i^{(n)} \) are already part of the final desired network, their inclusion in the current search space does not contradict our objective of reducing relay count, but helps by improving the chance of finding a hop constrained path after pruning a free relay.
- This selection of search space, thus, enforces the reuse of relays in Locked set, while trying to avoid the use of free relays, thereby improving the relay count.

Step 15: for each node \( j \in \overline{R}_i \) (comment: the following sub steps will be repeated until no more relay pruning from the set \( \overline{R}_i \) is possible without violating the hop constraint)

Step 15a: \( \text{TempPath}(S_i, 0) \leftarrow \text{SPT}(G_i^{(n)}\backslash j) \)

Remark: With slight abuse of notation, we designate by \( G_i^{(n)}\backslash j \), the restriction of the graph \( G_i^{(n)} \) to the node set \( V_i^{(n)}\backslash j \). After pruning a relay \( j \), we obtain the shortest path (\( \text{TempPath}(S_i, 0) \)) from source \( S_i \) to sink, using the remaining vertices in \( V_i^{(n)} \).

Step 15b: if hopcount(\( \text{TempPath}(S_i, 0) \)) > \( h_{\text{max}} \)
   continue; (comment: Go back to Step 15, and try pruning the next relay in \( \overline{R}_i \))
   else go to next Step

Remark: If the shortest path from \( S_i \) to sink in \( G_i^{(n)} \) after pruning relay \( j \) does not satisfy the hop constraint, we replace back the relay \( j \), and try pruning the next relay in \( \overline{R}_i \).

Step 15c: \( \text{path}(S_i, 0) \leftarrow \text{TempPath}(S_i, 0) \)
\( V_i^{(n)} \leftarrow V_i^{(n)}\backslash j \)
\( \overline{R}_i \leftarrow \overline{R}_i\backslash j \)
\( G_i^{(n)} \leftarrow \text{restriction of } G_i^{(n)} \text{ to } V_i^{(n)} \)

Remark: If relay \( j \) can be pruned successfully without
violating the hop constraint, we update the $n^{th}$ node disjoint path, $\text{path}_n(S_i, 0)$, and the sets $V_i^{(n)}$ (the set of vertices to be part of the search space for a feasible path) and $\mathcal{R}_i$ (the set of free relays to be pruned) as above before proceeding to the next iteration of the loop (i.e., before pruning the next relay in $\mathcal{R}_i$).

Note that since relay $j$ has been pruned, the updated candidate path will have at least one relay less than the relay set used by the initial feasible path, $\text{path}_{n^*}^{th}(S_i, 0)$.

\begin{align*}
\text{Step 16: } T &\leftarrow T \cup \text{path}_n(S_i, 0) \\
L^{(n)} &\leftarrow L^{(n)} \cup \{R \cap \text{path}_n(S_i, 0)\} \\
\mathcal{R}_i &\leftarrow \mathcal{R}_i \setminus L^{(n)}
\end{align*}

\text{End of Inner loop}

Remark: After obtaining the $n^{th}$ node disjoint, hop constrained path from a source $S_i$ to the sink using as few relays as possible, we augment the network $T$ with the path from source $S_i$ to sink, namely, $\text{path}_n(S_i, 0)$. Also, before proceeding to the next source (i.e., the next iteration of the inner loop), we update the Locked relay set with the new relays used in $\text{path}_n(S_i, 0)$; the free relay set $\mathcal{R}_i$ is also updated accordingly.

\begin{align*}
\text{Step 17: } n &\leftarrow n + 1 \\
\text{End of Outer loop}
\end{align*}

Remark: When we have obtained a node disjoint, hop constrained path from all the sources to the sink in the current iteration (of the outer loop), we proceed to the next iteration of the outer loop.

\text{End of pseudo code for Algorithm 2}

D. Analysis of the Algorithms

1) Time complexity: We show below that the time complexities of the algorithms are upper bounded by polynomials in $|Q|$, $|R|$, and $k$. Hence the algorithms are polynomial time.

**Lemma 2.** 1) The time complexity of Algorithm 1 is upper bounded by $k(|Q| - 1)g_{\text{sptirp}}(|Q|) + g_{\text{sptirp}}(|Q + |R|) + (|Q| - 1)(k - 1)g_{\text{sptirp}}(|Q| + |R|) + |R|g_{\text{sptirp}}(|Q + |R|) + g_{\text{sptirp}}(|Q + |R|))$.

2) The time complexity of Algorithm 2 is upper bounded by $k(|Q| - 1)g_{\text{sptirp}}(|Q|) + g_{\text{sptirp}}(|Q + |R|) + (k - 1)(|Q| - 1)(|R| + 1)g_{\text{sptirp}}(|Q + |R|)$ where $g_{\text{sptirp}}(\cdot)$ is the time complexity of the SPTiRP algorithm, and $g_{\text{sptirp}}(\cdot)$ is the time complexity of finding the shortest path tree.

**Proof:** Time complexity of Algorithm 1 Phase 1 of Algorithm 1 involves repeating the SPTiRP algorithm on the set $Q$ alone at most $k$ times for each of the sources. Hence, the time complexity of this phase is upper bounded by $k(|Q| - 1)g_{\text{sptirp}}(|Q|)$.

Phase 2 starts by running the SPTiRP algorithm on the entire graph. The time complexity involved therein is $g_{\text{sptirp}}(|Q| + |R|)$. Now the alternate route finding procedure involves finding SPT on $G_i$ (see the algorithm for definition of $G_i$), followed by two different methods of finding candidate routes.

Now, the time complexity of finding SPT on $G_i$ is upper bounded by $g_{\text{sptirp}}(|Q| + |R|) + \sum_{j=1}^{k} g_{\text{sptirp}}(|Q| + |R|)$, and this step is repeated at most $k - 1$ times for each source.

RoutineA (for computing the second candidate path) involves pruning relays from a path, one at a time, and running SPT on the remaining subgraph (i.e., the relay on the path, and all the eligible source nodes. See Algorithm 1 for detailed explanation) to check hop constraint feasibility. The time complexity of RoutineA is, therefore, upper bounded by $|R|g_{\text{sptirp}}(|Q| + |R|) + (j-1)(k-1)g_{\text{sptirp}}(|Q| + |R|) + |R|g_{\text{sptirp}}(|Q| + |R|) + g_{\text{sptirp}}(|Q| + |R|))$.

**Time complexity of Algorithm 2** The first and second terms in the above expression can be derived by similar arguments as given for Algorithm 1 above.

Now, the alternate route finding procedure starts by finding an SPT on a graph $G_i$, a restriction of the graph $G$. The complexity of this is upper bounded by $g_{\text{sptirp}}(|Q| + |R|)$. This is repeated for each source at most $k - 1$ times.

The next step in alternate route finding consists of pruning from a path (path$_2(S_i, 0)$), relays chosen from a certain selected set $(R_i)$, one at a time, and finding the SPT on the resulting restricted graph (see Algorithm 2 for detailed explanation) to check if hop constraint is satisfied by the resulting path. The worst case complexity of this step is upper bounded by $|R|g_{\text{sptirp}}(|Q| + |R|)$. This step is also repeated at most $k - 1$ times for each source.

Hence, the time complexity of Algorithm 1 is upper bounded by $k(|Q| - 1)g_{\text{sptirp}}(|Q|) + g_{\text{sptirp}}(|Q + |R|) + (|Q| - 1)(k - 1)g_{\text{sptirp}}(|Q| + |R|) + |R|g_{\text{sptirp}}(|Q + |R|) + g_{\text{sptirp}}(|Q + |R|))$.

Hence, the time complexity of Algorithm 2 is upper bounded by $k(|Q| - 1)g_{\text{sptirp}}(|Q|) + g_{\text{sptirp}}(|Q + |R|) + (k - 1)(|Q| - 1)(|R| + 1)g_{\text{sptirp}}(|Q + |R|)$.

From Lemma 2 it follows that Algorithm 1 and Algorithm 2 are polynomial time complexity algorithms.

2) Approximation guarantee: As already stated in Corollary 3, no polynomial time algorithm can provide finite approximation guarantee for the general class of RSNk-MR-HC problems. However, for a subclass of RSNk-MR-HC problems where the optimal solution for one connectivity with hop constraint (RST-MR-HC) is
non zero, we can derive a bounded worst case approximation guarantee for polynomial time complexity algorithms, as discussed below.

**Lemma 3.** If the optimal solution for the RST-MR-HC problem on an edge-weighted graph \( G = (V, E) \), with edge weights as defined earlier, uses \( n > 0 \) relays, then the optimal solution for the RSNk-MR-HC problem on that graph with the same hop constraint and \( r_{\text{max}} \) as the RST-MR-HC problem, uses at least \( n + k - 1 \) relays, where \( k > 1 \) is the number of node disjoint paths required from each source to the sink.

**Proof:** Consider a problem instance where the optimal solution for the RST-MR-HC problem uses \( n > 0 \) relays.

Suppose we claim that the optimal solution for the RSNk-MR-HC problem with \( k = 2 \) on that problem instance also uses \( n \) relays.

Therefore, for that problem instance, there exists a relay set \( R_{\text{opt}} = \{R_1, R_2, \ldots, R_n\} \) such that each source has a pair of node disjoint paths to the sink involving some of those relays.

Note that \( R_{\text{opt}} \) is an optimal solution for the RST-MR-HC problem on this problem instance. Consider the relay \( R_i \in R_{\text{opt}} \). Let \( S_i \) be the set of sources using \( R_i \) in the optimal solution for the RST-MR-HC problem.

Now, by our claim, \( R_{\text{opt}} \) is also the optimal solution for the RSNk-MR-HC problem with \( k = 2 \). Therefore, each of the sources in \( S_i \) has a hop constrained path to the sink using only the relays in \( R_{\text{opt}} \setminus R_i \). This, in turn, implies that the set \( R_{\text{opt}} \setminus R_i \) is sufficient to obtain one hop constrained path from each of the sources to the sink, i.e., \( R_{\text{opt}} \setminus R_i \) is, in fact, an optimal solution for the RST-MR-HC problem. This contradicts our earlier assumption that the optimal solution for RST-MR-HC problem uses \( n \) relays.

Therefore, our claim that the optimal solution for the RSNk-MR-HC problem with \( k = 2 \) uses \( n \) relays is wrong. Hence, the optimal solution for RSNk-MR-HC problem with \( k = 2 \) must use at least \( n + 1 \) relays.

Now, if the optimal solution for RSNk-MR-HC with \( k = 2 \) uses \( n + 1 \) relays, we can show using similar arguments as above that the optimal solution for RSNk-MR-HC problem with \( k = 3 \) would use at least \( n + 2 \) relays.

Proceeding thus, the optimal solution for the RSNk-MR-HC problem with \( k \) paths from each source to the sink would require at least \( n + k - 1 \) relays.

**Theorem 1.** For the set of problem instances where the optimal solution for RST-MR-HC problem uses non zero number of relays, the worst case approximation guarantee given by polynomial time complexity algorithms for the RSNk-MR-HC problem is \(|R|/k\).

**Proof:** Since the optimal solution for the RST-MR-HC problem uses non zero relays, the worst case scenario is that the optimal relay count for RST-MR-HC problem is just 1, and hence, from Lemma 3, the optimal solution for RSNk-MR-HC problem uses at least \( k \) relays, whereas a polynomial time algorithm for the same problem may end up using all the \(|R|\) relays. Hence, the worst case approximation guarantee is \(|R|/k\).

### VIII. Numerical Results for the \( k \)-Connectivity Algorithms

To compare the performance of the \( k \) connectivity algorithms, we ran the algorithms to solve the RSNk-MR-HC problem with \( k = 2 \) on the same random network scenarios that were generated to test the SPTIRP algorithm (see Section V). In none of the 1000 scenarios tested, the hop constraint turned out to be infeasible. The results are summarized in Table XIV.

From Table XIV we can make the following observations:

1) In all 5 sets of experiments (with different node densities), the average relay count required by Algorithm 2 to achieve 2 connectivity is less than that required by Algorithm 1.

2) In over 65% of the tested scenarios, Algorithm 2 performed better than Algorithm 1 in terms of relay count. In another 28.1% of cases, they performed equally well.

3) In all 5 sets of experiments, the maximum relay-count required by Algorithm 1 is more than that required by Algorithm 2 (although the maximums for the two algorithms may have been on different random scenarios).

4) In terms of mean execution time, Algorithm 2 performed much better than Algorithm 1 in all 5 sets of experiments.

5) For both the algorithms, the average relay count did not vary much with the node density.

6) For both the algorithms, the average execution time increases with increasing node density.

For each of the five sets of experiments, we also noted the minimum (maximum) relay count required by either algorithm over scenarios where the other algorithm uses a maximum (minimum) number of relays. The comparative study is summarized in Table XIV.

From Table XIV we observe that

1) For all 5 sets of experiments, in scenarios where Algorithm 1 performs at its worst in terms of relay count, the minimum relay count of Algorithm 2 is always much better than the relaycount of Algorithm 1. Also observe that the maximum relay count used by Algorithm 2 in all sets of experiments is better than that of Algorithm 1.
TABLE IV

| Potential relay count | Scenarios | Algo 1 Relay Count | Algo 2 Relay Count | Algo 2 better than Algo 1 | Algo 2 same as Algo 1 | Algo 2 worse than Algo 1 | Mean execution time in sec |
|-----------------------|-----------|--------------------|--------------------|--------------------------|---------------------|--------------------------|---------------------------|
|                       |           | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min |
| 100                   | 200       | 3.295   | 13  | 0   | 4.13    | 9   | 0   | 134     | 54  | 12  | 17.163  | 3.0429 |
| 110                   | 200       | 4.88    | 10  | 0   | 3.895   | 9   | 0   | 121     | 68  | 11  | 13.973  | 3.8558 |
| 120                   | 200       | 5.45    | 12  | 1   | 4.18    | 8   | 1   | 129     | 52  | 19  | 16.252  | 4.3314 |
| 130                   | 200       | 5.15    | 11  | 0   | 4       | 8   | 0   | 135     | 54  | 11  | 18.97   | 5.2316 |
| 140                   | 200       | 5.27    | 14  | 0   | 3.945   | 9   | 0   | 133     | 53  | 14  | 23.748  | 6.3596 |
| Total                 |           | 5.209   | 14  | 0   | 4.03    | 9   | 0   | 652     | 281 | 67  | 16.821  | 4.564  |

TABLE V

| Potential relay count | Scenarios | Max relay count of Algo 1 \( (m_1) \) | Min relay count of Algo 2 when Algo 1 uses \( m_1 \) | Max relay count of Algo 2 \( (m_2) \) | Min relay count of Algo 1 when Algo 2 uses \( m_2 \) | Min relay count of Algo 1 \( (m_1) \) | Max relay count of Algo 2 when Algo 1 uses \( m_1 \) | Min relay count of Algo 2 \( (m_2) \) | Max relay count of Algo 1 when Algo 2 uses \( m_2 \) |
|-----------------------|-----------|----------------------------------|-------------------------------------------------|----------------------------------|-------------------------------------------------|----------------------------------|------------------------------------------------|----------------------------------|------------------------------------------------|
|                       |           | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min |
| 100                   | 200       | 13      | 8   | 9   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   |
| 110                   | 200       | 13      | 8   | 9   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   |
| 120                   | 200       | 13      | 8   | 9   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   |
| 130                   | 200       | 13      | 8   | 9   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   |
| 140                   | 200       | 13      | 8   | 9   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   | 13      | 0   | 0   |

2) In scenarios where Algorithm 2 uses a maximum number of relays, the minimum relay count used by Algorithm 1 is still higher than the relay count of Algorithm 2.

3) In scenarios where Algorithm 1 uses zero relays, Algorithm 2 also uses zero relays (which is expected, since Phase 1 is same for both algorithms (see Algorithm 1 and Algorithm 2 in Section VII)).

4) In scenarios where Algorithm 1 uses the minimum non-zero number of relays (1 relay), the maximum relay count used by Algorithm 2 was just 1 more than the relay count used by Algorithm 1.

5) In scenarios where Algorithm 2 uses the minimum non-zero number of relays (1 relay), the maximum relay count used by Algorithm 1 was as high as 6.

Thus, from our observations in Table V we can conclude that the worst case performance of Algorithm 2 in terms of relay count is better than that of Algorithm 1.

To compare the performance of the two algorithms against the worst case performance bound given in Theorem 1 we did the following:

- For each of the five sets of experiments, we identified the scenarios where the optimal solution for the RST-MR-HC problem is non zero.
- For each of the scenarios thus identified, we can compute the lower bound on the optimal number of relays required for 2-connectivity, using Lemma 3 as follows. Thus, if the optimal solution for the RST-MR-HC problem uses \( n \) relays, the optimal number of relays required for 2-connectivity is lower bounded by \( n + 1 \).
- For each scenario, we obtained the approximation factor given by Algorithm 1 and Algorithm 2 w.r.t the lower bound computed above as approximation factor \( = \frac{Relay_{Algorithm}}{Relay_{lowerbound}} \).
- For each of the five sets of experiments, we obtained the worst and the best approximation factors (as computed above) achieved by the two algorithms, and also the worst case performance bound obtained from Theorem 1.

The results are summarized in Table VI. From Table VI we observe that:

1) For each of the five sets of experiments, the worst case approximation factor (as defined earlier, for scenarios where optimal solution of RST-MR-HC problem is non zero) achieved by both the algorithms is much better than the theoretical performance bound predicted in Theorem 1.

Also note that these approximation factors were computed based on a lower bound on the optimal solution for 2-connectivity; hence the actual performance of the algorithms is even better than this.

2) In all five sets of experiments, Algorithm 2 outperformed Algorithm 1 significantly in terms of the worst case approximation factor.

3) The best approximation factor achieved by both the algorithms was 1, i.e., the lower bound was actually achieved by the algorithms in some of the test cases.
In the relatively small number of test scenarios where Theorem 1 does not apply (i.e., optimal solution for RST-MR-HC problem is zero), and hence there is no bounded factor approximation guarantee for the algorithms, we obtained the maximum and minimum number of relays used by the two algorithms over those scenarios. The results are presented in Table VII.

From Table VII we see that even in scenarios where there is no bounded factor approximation guarantee for the algorithms, the performance of the algorithms is reasonably good, with the maximum relay count being 7 relays for Algorithm 1, and 5 relays for Algorithm 2. The minimum relay count for both the algorithms is zero (which is clearly optimal) in those scenarios.

IX. CONCLUSION

In this paper, we have studied the problem of determining an optimal relay node placement strategy such that certain performance objective(s) (in this case, hop constraint, which, in turn, ensures data delivery to the BS within a certain maximum delay) is (are) met. We studied two variations of the problem, namely, one connected hop constrained network design, and k-connected (survivable) hop constrained network design. We showed that the problems are NP-Hard, and proposed polynomial time approximation algorithms for the problems. The algorithm for one connected hop constrained network design problem, as can be concluded from numerical experiments presented in Section VII gives solutions of reasonably good quality, using extremely reasonable computation time. In the worst case, the algorithm can do as bad as using all but one potential relay locations instead of just one optimum relay. The algorithm can not do infinitely worse, in the sense that if the optimum design happens to use no relay node, the algorithm does not use any relay node either.

From the numerical results presented in Section VIII we can conclude that the algorithms proposed for the k-connected network design problem behave significantly better than the worst case performance bound predicted in Theorem 1 (in Section VII) for the subclass of problems to which the Theorem 1 applies. Even for problems where the algorithms do not have any bounded approximation guarantee, we found from our experiments that the algorithms behave reasonably well in terms of relay count. Also, from the results in Section VIII it can be concluded that Algorithm 2 performs better than Algorithm 1 in terms of relay count as well as execution time.

The algorithms proposed in this paper are basically local search algorithms where we start with a feasible solution based on an SPT on the graph, and then we search neighbourhoods of that solution until a local optimum is obtained. One might ask why these local search algorithms work so well in the tested random scenarios. The answer to this question is not immediately obvious, but, for the RST-MR-HC and RSNk−MR-HC problems, the graphs we ran our tests on were all geometric graphs; hence, a formal analysis of the properties of the underlying random geometric graph might provide some useful insights into the performance of these local search algorithms. We wish to address this issue in our future work.

Further, we are working on extending the design to traffic models more complex than the lone packet traffic model considered here. This requires the analysis of packet delays in a mesh network with more complex traffic flows and the nodes accessing the medium using CSMA/CA as defined in IEEE 802.15.4 [19], [20].

ACKNOWLEDGEMENT

This work was supported by the Automation Systems Technology (ASTeC) Center, a program of the Department of Information Technology at CDAC, Trivandrum, Kerala, India.

REFERENCES

[1] “www.honeywell.com/ps/wireless.”
[2] “www.isa.org/isa100.”
[3] R. A. Sajana, R. Subramanian, P. V. Kumar, S. Krishnan, B. Amrutur, J. Sebastian, M. Hegde, and S. Anand, “A low-complexity algorithm for intrusion detection using a PIR-based wireless sensor network,” in International Conference Series on Intelligent Sensors, Sensor Networks and Information Processing, (Sydney), 2009.
### TABLE VII
Performance of the $k$ Connectivity Algorithms for $k = 2$ in Scenarios where There is No Bounded Approximation Guarantee

| Potential relay count | Scenarios where RST-MR-HC has zero optimal solution | Theoretical performance bound | Max relaycount of Algo 1 | Min relaycount of Algo 1 | Max relaycount of Algo 2 | Min relaycount of Algo 2 |
|-----------------------|-----------------------------------------------------|-------------------------------|---------------------------|---------------------------|---------------------------|---------------------------|
| 100                   | 100                                                 | NA                           | 7                         | 0                         | 5                         | 0                         |
| 110                   | 110                                                 | NA                           | 7                         | 0                         | 4                         | 0                         |
| 120                   | 120                                                 | NA                           | 7                         | 0                         | 5                         | 0                         |
| 130                   | 130                                                 | NA                           | 7                         | 0                         | 5                         | 0                         |
| 140                   | 140                                                 | NA                           | 7                         | 0                         | 5                         | 0                         |
| Total                 | 280                                                 | NA                           | 7                         | 0                         | 5                         | 0                         |

[4] I. C. Society, *IEEE Standards Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs).* New York, October 2003.

[5] A. Bhattacharya, “Node Placement and Topology Design for Planned Wireless Sensor Networks,” Master’s thesis, Indian Institute of Science, 2010.

[6] S. Misra, S. D. Hong, G. Xue, and J. Tang, “Constrained Relay Node Placement in Wireless Sensor Networks to Meet Connectivity and Survivability Requirements,” in *IEEE INFOCOM,* 2008.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Bell Telephone Laboratories, Inc., 1979.

[8] “en.wikipedia.org/wiki/steiner tree problem.”

[9] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem,* vol. 53 of *Annals of Discrete Mathematics.* Elsevier Science Publishers B.V., 1992.

[10] V. V. Vazirani, *Approximation Algorithms.* Springer, 1st ed., 2001.

[11] G.-H. Lin and G. Xue, “Steiner tree problem with minimum number of Steiner points and bounded edge length,” *Information Processing Letters,* vol. 69, pp. 53–57, 1999.

[12] X. Cheng, D.-Z. Du, L. Wang, and B. Xu, “Relay sensor placement in wireless sensor networks,” *Wireless Netw,* vol. 14, pp. 347–355, 2008.

[13] E. L. Lloyd and G. Xue, “Relay node placement in wireless sensor networks,” *IEEE Transactions on Computers,* vol. 56, January 2007.

[14] S. Voss, “The Steiner tree problem with hop constraints,” *Annals of Operations Research,* vol. 86, pp. 321–345, 1999.

[15] A. M. Costa, J.-F. Cordeau, and G. Laporte, “Fast heuristics for the steiner tree problem with revenues, budget and hop constraints,” *European Journal of Operational Research,* vol. 190, pp. 68–78, 2008.

[16] M. Kim, Y.-C. Bang, and H. Choo, “On multicasting Steiner trees for delay and delay variation constraints,” 2006.

[17] J. L. Bredin, E. D. Demaine, M. T. Hajiaghayi, and D. Rus, “Deploying Sensor Networks with Guaranteed Capacity and Fault Tolerance,” in *MobiHoc'05,* ACM, 2005.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press and McGraw-Hill, 2 ed., 2001.

[19] C. K. Singh, A. Kumar, and P. M. Ameer, “Performance Evaluation of an IEEE 802.15.4 Sensor Network with a Star Topology,” *Wireless Network,* vol. 14, pp. 543–568, 2008.

[20] R. Srivastava, “Analysis and Design of Beacon-less IEEE 802.15.4 Multi-Hop Networks for Industrial Sensor Networking Applications,” Master’s thesis, Indian Institute of Science, 2011.