Practical Beacon Placement for Link Monitoring Using Network Tomography

Ritesh Kumar and Jasleen Kaur

Abstract— **Recent interest in using tomography for network monitoring has motivated the issue of whether it is possible to use only a small number of probing nodes (beacons) for monitoring all edges of a network in the presence of dynamic routing. Past work has shown that minimizing the number of beacons is NP-hard, and has provided approximate solutions that may be fairly suboptimal. In this paper, we use a two-pronged approach to compute an efficient beacon set: (i) we formulate the need for, and design algorithms for, computing the set of edges that can be monitored by a beacon under all possible routing states; and (ii) we minimize the number of beacons used to monitor all network edges. We show that the latter problem is NP-complete and use various approximate placement algorithm that yields beacon sets of sizes within $1 + ln(|E|)$ of the optimal solution, where E is the set of edges to be monitored. Beacon set computations for several Rocketfuel ISP topologies indicate that our algorithms may reduce the number of beacons yielded by past solutions by more than 50% and are, in most cases, close to optimal.**

I. Introduction

THE last two decades have witnessed an exponential growth of the Internet in terms of its infrastructure, its traffic load and composition, as well as its commercial usage. In order to provide good connectivity, reliability, and quality of service to Internet users, it is important to have the ability to monitor the health of the networks that comprise the Internet. Consequently, there is significant interest in developing network monitoring infrastructures that allow ISPs as well as end-users to monitor network links and nodes.

An important consideration in the design of monitoring infrastructures is that of developing low-cost solutions. In particular, the idea of placing and operating sophisticated monitors at all nodes in a network is neither cost-efficient nor practical (especially when monitoring is performed by end-users). Instead, there has been significant recent interest in relying on *tomographic* techniques that use only a few probing end-nodes (beacons) for monitoring the health of all network links. They do so by sending specially-designed probes along the IP routes to the two end-points of a given link, only one of which traverses the link—by observing the difference in the results of the two probes, properties of the link are estimated [1]–[9]. Though there are several properties of a link which can be measured topographically, we consider only link failure and link delay monitoring in this work.

A central issue in the design of a monitoring infrastructure is that of *beacon placement*—given a set of links to be monitored, which network nodes should be used to construct a beacon set? Two requirements guide the design of a good beacon placement strategy:

- *Minimizing the number of beacons.*
  One of the prime motivations for using tomography for network monitoring is to reduce the cost of the monitoring infrastructure. However, even a tomographic infrastructure involves the development, installation, debugging, operation, and maintenance of specialized software/hardware on each beacon. In order to minimize the cost of doing so, it is important that the number of beacons used to monitor all links of a given network are minimized.
- *Robustness to routing dynamics and uncertainty.*
  A monitoring infrastructure should not assume a specific routing configuration in selecting a beacon set. This is because of two reasons. First, routing state in many networks responds to changes in traffic patterns and link loads, as well as to link failures. Since Internet traffic conditions are highly dynamic, the default IP routes in a given network may change at time-scales much smaller than the time-scales at which beacons are deployed. Second, the routing state within individual ASes may be considered proprietary information and may not even be available—this is an important consideration for monitoring infrastructures that cover multiple ASes. Consequently, a beacon placement strategy should find a beacon set that is able to monitor all relevant network links, independent of the current route configuration.

A few recent efforts have focused on the problem of finding beacon sets for a network [4], [7]. These, however, do not adequately meet the above challenge—the beacon set of [4] is not robust to changes in IP routes, and the beacon set proposed in [7] can be quite large for real ISP topologies (details in Sections II and VI). In this paper, we present and evaluate beacon placement strategies that meet both aspects of the above challenge.

Contributions

We formally model the problem of beacon placement using a generic framework that allows us to evaluate several beacon placement strategies—proposed here as well as in related work—that incorporate different beacon types as well as policy constraints. We approach the beacon placement problem both theoretically and experimentally. Our analytical framework relies on a two-pronged methodology:

- First, we consider the issue of diversity in routing configurations and in the probing-flexibility of beacons, that govern *how monitoring is done by beacons*. We present

This research was supported in part by NSF CAREER grant CNS-0347814, a UNC Junior Faculty Development Award, and NSF RI grant EIA-0303590.

R. Kumar and J. Kaur are with the University of North Carolina at Chapel Hill. Email: {ritesh, jasleen}@cs.unc.edu

a generic framework that allows the incorporation of this diversity. Our framework relies on defining the concept of a *Deterministically Monitorable Edge Set (DMES)* of a beacon as the set of links that can be monitored by the beacon under all possible route configurations. We present efficient graph-theoretic algorithms for computing the DMES of two types of beacons discussed in the literature.

- Second, we consider the optimization problem of finding the minimum number of beacons which can collectively monitor all the links of a network (the union of their DMES covers all network edges). We show that this problem is intractable. We derive approximation algorithms that yield beacon sets of sizes within $1 + ln(E)$ of the optimal solution in general, and within a constant of 2 for one of the beacon types considered.
  We also establish additional properties of beacon sets that help in improving the computation efficiency of our approximation algorithms.

We simulate our beacon placement algorithms on several real ISP topologies obtained from the Rocketfuel project [10]. Our experimental results illustrate that: (i) our beacon placement strategies yield beacon sets that are $50 - 80\%$ smaller than those yielded by [7]; (ii) in practice, the best-performing heuristics for our approximation algorithms yield solutions fairly close to optimal; and (iii) the routing-dependent beacon placement strategy of [4] yields smaller beacon sets (by 10-75%) for statically-routed networks, but the beacon sets are much larger in the long-run if traffic-dependent dynamic routing is employed.

Finally, we extend our analytical framework to incorporate realistic network scenarios that include half-duplex links as well as non-transit networks.

The rest of this paper is organized as follows. We formulate the problem of Beacon Placement and discuss past work in Section II. We define and compute DMESs in Section III and address the Beacon Minimization Problem in Sections IV and V. We present our evaluations on Rocketfuel topologies in Section VI. We derive additional insights for specialized beacon types and network scenarios in Sections VII-VIII. We summarize our conclusions in Section IX.

Notations and Assumptions

We model a network as an undirected graph $G(V, E)$, where V is the set of network nodes and E is the set of links (or edges)—in section VIII-A, we extend our analysis to directed graphs as well. We use the terms networks and graphs—and also links and edges—interchangeably. We assume that G is connected (there exists a path from any node to any other node). We also assume that all routes are *acyclic (simple)*. We say that two physical paths between a pair of nodes are *distinct*, if they differ in even one of the edges traversed.

Finally, we make an important assumption that if two node are physically connected, there *exists a network route* between them. In fact, we assume that no physically available loop-free (simple) path in the network is prohibited as a network route; in section VIII-B we will lift this assumption to incorporate real-world scenarios of non-transit networks.

II. Problem Formulation

Beacon Placement

In a tomographic network monitoring infrastructure, each network link is monitored by a special probing node, referred to as a *beacon*. The basic idea behind most tomographic setups is fairly simple: the beacon sends a pair of nearly-simultaneous probes to the two end-nodes of the link, only one of which traverses the link. Each end-point sends back a response to the beacon—this may be implemented, for instance, using ICMP echo messages. The results of the probes can then be used to infer properties of the link. For instance, if the objective is to measure link delays, then the difference in round-trip times of the two probes can be used as an estimate. If the objective is to simply detect link transmission failures, the success and failure of the two probes may be used as reasonable estimators. In this paper, we consider the problem of monitoring only *single* link failures.

Note that, in general, a beacon is capable of monitoring several network links.[1] A set of beacons that can be collectively used to monitor *all* the links of a network is referred to as a *beacon set*. A central issue in the design of a monitoring infrastructure is that of *beacon placement*—which network nodes should be used to construct a beacon set? Specifically, and as motivated in Section I, our objective is to: *find the smallest number of beacons required to deterministically monitor all the links of a given network, even in the presence of dynamism and uncertainty in IP routes.*

MES and Past Work

Our methodology for finding the smallest beacon set for a network first enlists the edges that can be monitored by each candidate beacon—this is referred to as the *monitorable edge set* (MES) of the beacon. Note that the union of MES of all beacons in a beacon set is equal to the set of all network edges. In general, the larger is the average MES size in a beacon set, the smaller is the beacon set.[2]

We briefly discuss below two beacon placement schemes proposed recently, which differ in their assumptions about which links comprise the MES of a beacon.

- *Simple Beacons*:
  In [4], the authors assume that the MES of a beacon consists of all links that can be reached by the beacon—which are links that lie on its IP routing tree.[3] In order

[1]We assume that each beacon node is capable of monitoring all of its directly-connected links using a link-layer technology. Additionally, each beacon can monitor some remote links as described above. We also assume throughout this paper that *all* links of a network need to be monitored—it is straightforward to tune our analysis to the case when only a subset of all network links needs to be monitored.

[2]Perhaps the largest MES (and smallest beacon set) that can be envisioned is when a *single* beacon monitors *all* the links of a network—this is feasible, for instance, in a network which supports source-routing [11]. In such a network, a beacon can precisely specify the path traversed by its probes, and hence can probe the end-points of any network link. However, this strategy relies on the availability of source-routing support at *all* network nodes, which is the not the case with a majority of current networks [4].

[3]The IP routing tree of a node refers to the tree, rooted at the node, which is formed by the links that lie on the default IP routes from that node to each of the other nodes in the network.

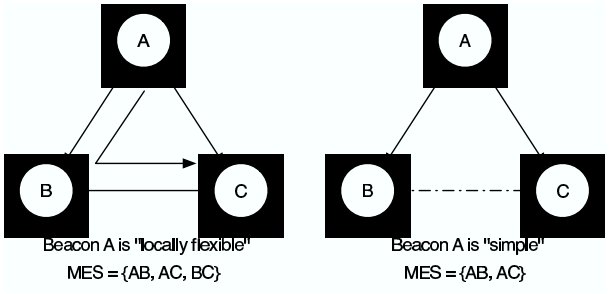


Fig. 1. Simple vs. Locally-flexible Beacons

to monitor a link in its MES, such a beacon—henceforth referred to as a "simple" beacon—sends probes to the end-points of the link, along the default IP paths to those end-points. The authors demonstrate that the problem of minimizing the size of the beacon set with such beacons is NP-hard, and provide a placement strategy that produces a beacon set no larger than $1 + ln|E|$ times the optimal beacon set. Unfortunately, since the authors assume that all links within the routing tree of a beacon belong to its MES, their strategy is not robust to changes in routing trees and works only for networks with static routes.

- *Locally-flexible Beacons*:
  In [7], the authors consider beacons that have a greater flexibility in selecting the paths taken by the probes. Specifically, the beacons—henceforth referred to as "locally-flexible" beacons—are capable of selecting the *first* link (outgoing link from beacon) on which a probe to any destination is transmitted. A probe can, therefore, be sent to a destination either along the current IP route to the destination, or along one of the current IP route from any immediate neighbor to the same destination (see Figure 1).[4] Furthermore, the authors do not assume static routing state and define the MES of a beacon to consist of links that, irrespective of what current routes are, can always be monitored. The authors do not provide a mechanism to compute such an MES for a beacon, but show that even if these sets are known, the beacon set minimization problem is NP-hard. The authors instead suggest an alternative beacon-placement strategy which, unfortunately, can result in fairly large beacon sets for current network topologies (see Section VI).

To summarize, existing beacon placement strategies are either not robust to routing dynamics or are inefficient in minimizing the number of beacons.[5]

[4]The authors in [7] implicitly assume that the default IP route from any neighbor to a given destination will not go through the beacon node. This assumption may get violated when a path through the beacon has a smaller cost that any other physical path between a neighbor and the destination.

[5]An orthogonal problem of beacon placement for detecting *multiple link failures* that occur simultaneously has been considered in [12]. In general, it is not possible to detect all cases of simultaneous link failures in a given network. In [12], the authors restrict their attention to those simultaneous link failures that can be detected in the absence of any limitations on the number of beacons and probes. They then provide efficient algorithms for minimizing the number of beacons and probes needed for detecting these failures. Like [4], this work assumes "simple" beacons and uses the IP routing tree in the beacon set computation and, hence, is applicable only to networks with non-dynamic routes. We believe that it is possible to use our formulations from this paper to extend the work in [12] to locally-flexible beacons, as well as to networks with dynamic routing.

Our Approach

In this paper, we build on past work to address these limitations by using a two-pronged approach:

1) **Deterministic MES Computation:** In order to achieve robustness to routing dynamics, for each candidate beacon, we determine the set of edges—referred to as its Deterministic MES (DMES)—that can be monitored by it under *all* possible routing configurations.
2) **Beacon Set Minimization:** In order to minimize infrastructure cost, we address the problem of finding the *smallest* beacon set.

In the following two sections we present our abstractions and methodologies for implementing the above for both simple and locally-flexible beacons.

III. Deterministically Monitorable Edge Sets

The first key problem we need to solve is to find the set of edges that can be monitored by a beacon, independent of the routing configuration. This is formally captured in the following definition.

Definition 1: An edge is said to be *deterministically monitorable* by a beacon if the beacon can monitor it under all possible route configurations. The *Deterministically Monitorable Edge Set* (DMES) of a beacon is the set of all deterministically monitorable edges associated with that beacon.

In what follows, we consider both simple and locally-flexible beacons and present algorithms for computing their DMES. For clarity, we first establish an equivalence between "deterministically monitoring an edge" and the topological structure of the network. Lemma 1 does that:

Lemma 1: If all possible (simple) physical paths from a beacon u to a node y end in the same edge e, then u can deterministically monitor edge e.

Proof: Since all possible simple physical paths from u to y end in e, then the current network route from u to y also ends in e.[6] This implies that whenever a probe is sent from u to y and it reaches y, the probe always passes through the edge e. If the other end-point of e is x, any monitorable property of e may be estimated regardless of the current routing state of the network, by sending probes from u to each of x and y. ■

The advantage of the above formulation is that it allows us to ignore the network routing state, which is *dynamic* and derived from an exponentially large set of possible paths, and use only the *static* topology of the network for computing the DMES. This can be done by relying on graph-theoretic analysis (such as depth-first search) for efficiently finding for each potential beacon u, the set of edges e such that all physical paths to one of the end-points of e end in that edge. Lemma 1 ensures that u can monitor such an edge e under all possible route configurations.

It is important to emphasize that there may be edges in a graph which may never qualify to become a DME using the formulation above. For example, in figure 2, the DMEs for *all* the nodes in a graph are the edges 1-2 and 5-6. To work

[6]This is because of our assumption that network routes are simple; therefore, the current route is also a member of the set of all simple physical paths from u to y.

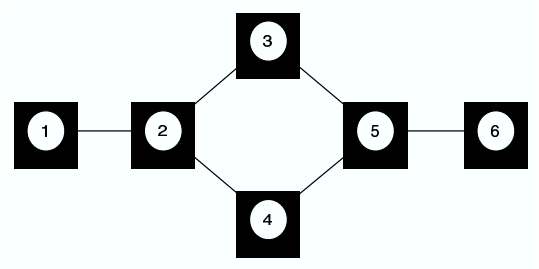


Fig. 2. The DMES may not be a connected graph.

around this, we assume that all edges directly connected to a beacon can also be monitored by the beacon. Such monitoring is done using either physical layer or device driver queries on the directly connected link. This gives us the hope of using extra beacons to monitor edges which don't fall under the above formulation. We shall refer to edges monitored in this way also as DMEs in the interest of reducing terminology, though we will differentiate between these kinds of DMEs in Section VII.

We present graph-theoretic algorithms for computing the DMEs for simple and locally-flexible beacons in Sections III-B and III-C, respectively. Below, we first illustrate how a beacon can monitor a DME for link failures and latencies.

A. Monitoring DMEs

We illustrate the use of the DMES formulation by considering two kinds of probes which can be used to monitor link failures and latencies, respectively, for beacon-link pairs that satisfy the pre-condition in Lemma 1. Lemma 2 first establishes a crucial property relevant for link failure monitoring. It is important to observe that link failure is perhaps the most fundamental property of a link that can be monitored; if a link is down, it is unlikely that additional properties of the link—such as latency and bandwidth—can be monitored.

Lemma 2: Let the two end-nodes on the link e be x and y such that a probe packet from the beacon u to y traverses in the direction $x \rightarrow y$. If a probe to x is successfully responded to and if e and y are up, then a probe to y will also be successfully responded to.

Proof: Let p be the current network route of a probe from u to x; we make the following observations;

1) p cannot traverse y. This is because if it does then p contains a cycle that includes x and y, which contradicts our assumption that network routes are simple.
2) If p and e are up, then there exists at least one simple *physical* path between u and y that is also up. This is because p can always be extended by e to yield such a path, p'. Note from the first observation above that p' does not contain cycles, and hence is a *simple* path.
3) If p and e are up, then there exists a network route between u and y. This is because p' is up and can be used in the event that all other candidate routes from u to y are broken.[7] Note that this relies on our assumption that no physically connected path in the network is prohibited as a network path.

The above observations indicate that if p and e are up, there is a valid network route from u to y, and hence the probe to y should be successful. ■

Monitoring link failure: Link failures can be monitored by exchanging *ping*-type request-response messages between the beacon u and each of the two end-points, x and y, of the link e. The probe results can then be used by u to infer whether e is up or down as follows:

- If the probe to y is successful (y's response reaches u), then e is up. This is because if e were down, then no physical path would exist from u to y (pre-condition in Lemma 1), and the probe would not have been successfully responded to.
- The probe to y fails but the probe to x is successfully responded to. From Lemma 2, this implies that e is down.
- The probes to both x and y fail. This leads to uncertainty in concluding whether e is up or down, as there could be failures in the paths to each of x and y.
- It is not possible for the probe to y to be successfully responded to without the probe to x being successfully responded to as all probes that reach y also reach x.

Monitoring link latency: Link latencies can be monitored for networks that rely on a *monotonic* routing policy. Using the construction in Lemma 2, this means that when routes have stabilized after any failures, the path p is embedded in the path p'. Beacons can then exchange *ping*-type request-response messages with each of the two end-points, x and y, of the link e. The probe results can then be used by u to infer Probe results can then be used by u to infer the latency of e as follows:

- We require probes from u to each of x and y to be successfully responded to.
- Assuming monotonic routing, the route of the probe from u to y is exactly the same route as that of the probe from u to x except the last edge e.
- The difference in the round trip times for both these probes gives us the round trip delay of the link e.

B. DMES for Simple Beacons

Theorem 1: Let u be a simple beacon and let $S(v)$ be the set of all distinct physical paths from u to another node v. The link $l(v)$ is deterministically monitorable by u if for *all* paths p in $S(v)$, $l(v)$ is the last edge on p. The DMES of u is the set of all such edges $l(v)$ for all nodes $v \in V$.

Proof: Since all paths from the simple beacon u to v have $l(v)$ as the last edge, the current IP route from u to v (which takes one of these paths) also ends in the edge $l(v)$. From Definition 1 and Lemma 1, therefore, simple beacon u is able to monitor the link $l(v)$. ■

Note that a DMES yielded by Theorem 1 need not form a connected sub-graph; Figure 2 illustrates that the DMES of node 1 includes only the edges 1-2 and 5-6. We now present an efficient algorithm for computing the DMES for simple beacons.

[7]Note that in case a link failure occurs in the network, it may take some time for the network routing state to converge to the above-mentioned path p'.

Algorithm 1 Computing DMES of a simple beacon u.

Initialize S to be an empty set
for all edges l neighboring u **do**
  Include l in S
end for
for all nodes v in V **do**
  Do a depth first search from v (we get subgraphs each connected to v by one or more edges)
  if u lies in a subgraph connected to v by only a single edge e **then**
    Include the edge e in S
  end if
end for
S is the DMES for u

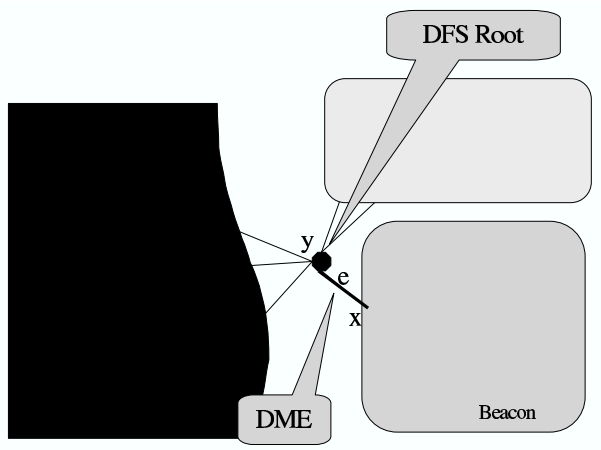


Fig. 3. The subgraphs of the DFS tree. Running a DFS allows us to see if an edge can be deterministically monitored by a beacon.

Proof: (Proof of correctness) Consider a depth first tree (along with its back edges) constructed from the node v. If we consider all subgraphs sprouting from the neighboring edges of v, then these might connect to v via one or more edges. These subgraphs are connected to each other only through v. Separating subgraphs this way helps us to isolate all possible paths from the beacon u to the node v. Any probe packet from u to v is entirely confined to paths in the subgraph containing u. Now, if the beacon u lies in a subgraph which connects to v via only one edge, all paths from u to v have to cross this edge at the end of the path. However, if u lies in a subgraph which is connected to v via two or more edges, then there exist at least two distinct paths from the simple beacon u to the node v which end in different edges to the node v. This means that the edges are not deterministically monitorable from u (Theorem 1). ■

Time Complexity: The cost of computing the DMES of a simple beacon is essentially that of running a depth first search (DFS) algorithm at every node in the network. Since the time complexity of running a depth first search on $G(V, E)$ is $\Theta(|E| + |V|)$, the time complexity of Algorithm 1 is $\Theta(|V|(|E| + |V|))$.

Note that the DMES for multiple simple beacons can be computed in parallel. After running DFS on a node v, we can add an incident edges of v to the DMES of all nodes that belong to the subgraph rooted at the edge, if there are no more edges connecting that subgraph to v. Since the number of potential nodes is bounded by $|V|$, and the time complexity of depth first search is $\Theta(|E|+|V|)$, the time complexity for the parallel DMES computation algorithm is the same as above. Hence, we can calculate the DMES of *all* nodes in $G(V, E)$ in $\Theta(|V|(|E| + |V|))$ time.

C. DMES for Locally-flexible Beacons

Theorem 2: Let u be a locally-flexible beacon and E_u be the set of edges directly connected to u. For each edge $i \in E_u$, let $S_i(v)$ be the set of all paths from u to any other node v, that start with the edge i. A link $l_i(v)$ is deterministically monitorable from u if for all paths in $S_i(v)$, $l_i(v)$ is the last edge. The DMES of u is the set of all deterministically-monitorable edges $l_i(v)$, for all $v \in V$ and all $i \in E_u$.

Proof: Since locally-flexible beacons can select the outgoing link on which to transmit a probe, we need to consider only those paths to v which start from a specific edge in E_u, to see if there is a common ending edge. Thus, even if u has paths to v which end with different edges, if all paths to v that start from u with edge i end with a common edge $l_i(v)$, u has the control over the ability to reach v through $l_i(v)$. From Definition 1 and Lemma 1, therefore, the common edge is deterministically monitorable. ■

Below, we present an algorithm for computing the DMES for locally-flexible beacons.

Algorithm 2 Computing DMES of a locally-flexible beacon u.

Initialize S to be an empty set
for all edges i neighboring u **do**
  Include i in S
  Remove i from E
end for
for all nodes v in V **do**
  Do a depth first search from v (we get subgraphs each connected to v by one or more edges)
  if one of u's neighbors lies in the subgraph connected to v by a single edge e **then**
    Include the edge e in S
  end if
end for
S is the DMES for u

Proof: (Proof of Correctness) The proof is similar to that for Algorithm 1. Let u_i be the neighbor connected to u through i. The subgraph containing u_i also contains all paths from u to v that start in i. This is because, if there was another path from u to v through i, v and u_i would have been connected which would be captured in the depth first search. Conversely, consider any simple path from u_i to v. Since Algorithm 2 removes i from E before running the DFS, a possible path in the subgraph containing u_i doesn't have i in it. Thus, adding i at the start of the path still retains the loop-free (simple) property of the path. Such a path is a valid path from u to v starting with edge i. Since we removed u's neighboring edges

from E, one could argue that some paths might be missing. However, this cannot be true because no simple paths from v to u would transit u in the middle of the path. Hence the subgraphs obtained by removing the edges neighboring u are representative of all paths from u's neighbors to v. ■

Time Complexity: The cost of this algorithm is that of running a depth first search on each node and for each depth first search run checking if any of the neighbors of u are in a singly connected subgraph. Thus, if the degree of u is k, the time complexity of the algorithm is $\Theta(|V|(|E|+|V|+k))$. Since k is bounded by $|V|$, the time complexity is $\Theta(|V|(|E|+|V|))$. Note that, unlike simple beacons, DMES for locally-flexible beacons can not be computed in parallel for multiple nodes because for each locally-flexible beacon we customize the graph $G(V,E)$ (removal of neighboring edges) specific to the beacon before doing all the depth first searches. The complexity of computing DMES for *all* nodes in $G(V,E)$ is, therefore, $\Theta(|V|^2(|E|+|V|))$.

IV. Beacon Set Minimization

The second key problem—of minimizing the beacon set for a network—is formally stated below:

Beacon Minimization Problem (BMP): Let D_u be the DMES associated with a node $u \in V$. Then the *beacon-minimization problem* is to find the smallest set of beacons, $B \subseteq V$, such that $\bigcup_{b\in B} D_b = E$.

The Beacon Placement Problem is intractable for the beacon types considered in this paper. We formally prove in section VII-A that BMP is NP-Complete for the case of simple beacons. BMP has also been shown to be NP-complete for locally-flexible beacons in [7]. Below we develop a correspondence between the general BMP (independent of beacon type) and Minimum Set Cover problem (MSCP)—this will let us apply well-known MSCP heuristics for addressing BMP.

Theorem 3: BMP is a special case of the Minimum Set Cover problem.

Proof: MSCP [13] can be stated as follows. Consider a set S with elements $e_1, e_2, \ldots$. Now consider a group of arbitrary subsets of S; $X_1, X_2, X_3 \ldots$ such that $\bigcup_i X_i = S$. The Min Set Cover problem is to find a collection of X_i's (say set Q), such that $\bigcup_{X_i\in Q} X_i = S$ and $|Q|$ is the minimum.

To show that BMP is a special case of MSCP, consider a graph $G(V,E)$. Since every node can deterministically monitor at least its neighboring edges, $\bigcup_{v\in V} D_v = E$. Also, $\forall v \in V : D_v \subseteq E$. To solve BMP, we need to find the *smallest* subset, $B \subseteq V$, such that $\bigcup_{v\in B} D_v = E$; then B is the required beacon set. Now consider a set $B' = \{D_v : v \in B\}$. Note that $|B'| = |B|$. Also note the correspondence between MSCP and BMP given the associations $S \rightarrow E, X_i \rightarrow D_v$ and $Q \rightarrow B'$. ■

MSCP is known to be NP-Complete [13], [14]. However, MSCP has a pruning-based approximate solution—below, we adapt the pruning algorithm and use heuristics from the literature [15] to establish approximate solutions with bounded optimality for BMP.

It is straightforward to see that the algorithm returns a valid beacon set. This is because every edge that was eliminated from E' could be deterministically monitored by some node that was already included in the beacon set.

Algorithm 3 Find the beacon set for completely monitoring a graph $G(V,E)$.

Initialize B to be an empty set
Initialize $E' = E$
while E' is not empty **do**
  Select* node u from V not in B
  $E' = E' -$ the DMES of u
  Include u in B
end while
B is the required beacon set.

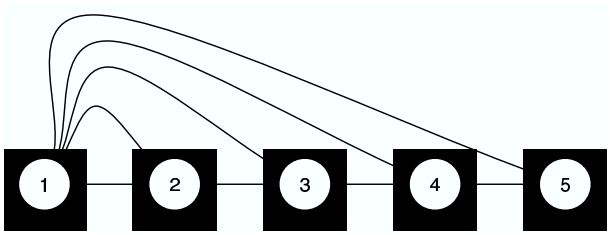


Fig. 4. Optimality of the "pruning" algorithm depends on the order node-selection.

The efficacy of the "pruning" algorithm in minimizing the size of the beacon set depends on the order of selection (the *ed operation in Algorithm 3) of nodes. For instance, consider the topology in Figure 4. The optimal beacon set (with locally-flexible beacons) for this topology contains just node 1. However, the "pruning" algorithm will lead to a non-optimal beacon set if it selects any node other than node 1 as its first beacon. In fact, selecting the nodes in the order 5, 4, 3, 2 and 1 causes the "pruning" algorithm to select all the nodes in the graph for the beacon set.

Fortunately, there exists a known heuristic for the MSCP pruning-based solution that ensures that the size of the solution is within a bound of the optimal [15]. The heuristic dictates the following node-selection rule (* in above algorithm) for BMP: select the node with a DMES that has the maximum overlap with the current pruned graph. Specifically, if E' is the current set of edges of the pruned graph then we choose the node v such that $|D_v \cap E'|$ is maximum. This heuristic results in provable [15] bounds on optimality of the beacon set: $\frac{|B(heuristic)|}{|B(optimal)|} = 1 + ln|E|$.

A. Monitoring a Subset of the Links

Some network operators may be interested in investing in a monitoring infrastructure that monitors only a critical subset of network edges. For such cases, our pruning algorithm (Algorithm 3) can be extended to find a potentially smaller beacon set as follows.

Given a graph $G(V,E)$, such that for every node $i \in V$, its DMES is D_i, let L, $L \subseteq E$, be the subset of links that need to be monitored. Consider the following algorithm:

Algorithm 4 differs from Algorithm 3 in two prime ways:

- E' is initialized to L rather than E.
- The node selection heuristic selects a node whose DMES has the maximum overlap with the pruned set of edges,

Algorithm 4 Find a beacon set to monitor the set of edges L in the graph $G(V, E)$.

Initialize B to be an empty set
Initialize $E' = L$
while E' is not empty **do**
  Select* node u from V not in B
  $E' = E'-$ the DMES of u
  Include u in B
end while
B is the required beacon set.

which is a subset of L (still equivalent to "currently pruned graph").

Theorem 4: Algorithm 4 finds a beacon set for monitoring edges in L using no more than $\Theta(1 + ln|L|)$ beacons.

Proof: Using the same terminology of beacons, DMES and links, consider a Min Set Cover problem as follows. Let the set of edges $E = L$, the set of DMESs under consideration $= \{D_i : D_i \cap L \neq \emptyset\}$. By the node selection heuristic used in algorithm 4, the algorithm reduces to the standard Min Set Cover heuristic for the smaller instance of Min Set Cover problem outlined above. Hence, the optimality of the solution is bounded by the optimality bound for the smaller instance of Min Set Cover problem which is $\Theta(1 + ln|L|)$. ■

V. High Arity Nodes and Beacon Sets

In [7], the authors introduced the concept of high arity nodes which was used in constructing a beacon set. In this section, we show how the concept of node arity can be useful in speeding-up the computation of a small beacon set, as formulated in Section IV. Below, we restate the definition of node arity from [7] in a slightly different manner.

Definition 2: *(Node Arity)* The arity of a node, u, *with respect to another node*, v, is defined as the number of distinct paths that exist between the two nodes such that each of these paths starts from a unique outgoing edge from u. The arity of a node u is defined as the maximum of arities of u with respect all nodes of the graph.

Using the terminology of [7], we refer to a node as "high arity" if the node's arity is more than one. Note that since $G(V, E)$ is assumed to be connected, there is at least one path from every node to every other node (assuming $|V| > 1$). Hence, the arity of a node is always greater than or equal to one. Also, since the maximum number of distinct paths (with a unique outgoing edge) from u to v can not be more than the degree of u, the arity of a node is bounded by its degree.

Algorithm 7 in the Appendix gives an efficient way for finding whether a node in $G(V, E)$ is high arity or not. Our algorithm is based on the insight that if a node v has arity one, all subgraphs generated in the depth-first search from v will be connected to v by a single edge.

A. Single Arity Networks

It is interesting to study the Beacon Placement Problem for a graph $G(V, E)$ that contains only single arity nodes. We show the optimal beacon set for such a graph is a singleton set, and can contain any one node of the graph.

Lemma 3: A graph $G(V, E)$ with no high arity nodes is a tree.

Proof: Since the graph is connected, the nodes present in the graph have an arity of at least one. Since there are no high arity nodes in graph, the arity of all nodes is one. Suppose there is a cycle in G. Then, any two distinct nodes in the cycle have separate paths to each other from their different outgoing edges. Thus, the nodes in the cycle are high arity, which is a contradiction. Hence, the graph cannot contain cycles. This implies that $G(V, E)$ is a tree. ■

Theorem 5: A network with no high arity nodes can be monitored by a single beacon on any node in the network.

Proof: From Lemma 3, we know that a network with no high arity nodes is a tree. Consider an arbitrary node u in the tree as a beacon; then the network graph can be considered as a tree rooted at u. Since the graph is a tree, there is only one simple physical path available from u to any other node. In particular, this implies that given *any* network edge e, all physical paths (in this graph, there is only one such path) from u to one of the end-points of e, end in the edge e. Thus, e is a DME of u. Thus, all edges of the graph belong to the DMES of u, and can be monitored by it. ■

B. High Arity Networks

We next show that using node arity analysis, the beacon search space in the pruning algorithms of Section IV can be substantially reduced. Specifically, in Theorem 6 we show that only high arity nodes need to be considered as potential beacons. Lemma 4 is used to prove the theorem.

Lemma 4: A graph which has at least one high arity node cannot be completely monitored by a single simple or locally-flexible beacon placed on a single arity node.

Proof: Consider a graph $G(V, E)$, which has a high arity node x. Also consider a single beacon on a single arity node b. Since x is a high arity node, there exists a node y, to which x has multiple paths with different outgoing edges from x and multiple incoming edges to y. Thus, x and y are part of a cycle. b cannot be on this cycle as otherwise it would be high arity. Note that b cannot deterministically monitor any edges in this cycle. This is because for any edge in the cycle, b has multiple paths to one of its end-points, which end in a different edge. Hence, b cannot completely monitor the entire graph. ■

Theorem 6: For a network with at least one high-arity node, an optimal beacon set is a subset of the set of high arity nodes.

Proof: Let B be an optimal beacon set of graph $G(V, E)$. Suppose $b \in B$ is a single arity node. Since B is optimal, removing b from B causes at least one edge $e \in E$ to be not deterministically monitorable by B. Let x and y be the two nodes on either side of e and, without loss of generality, assume that e is traversed when a probe packet is sent from b to y (and hence not traversed when sent to x). Now consider a depth first search from y. Consider the subgraphs sprouting from the edges adjacent to y. Since e is deterministically

monitorable from b, b lies in the subgraph, F_e, sprouting from the edge e. Also, F_e is connected to y by only e and no other edge (otherwise, e wouldn't have been deterministically monitorable by b). Since, e was exclusively monitorable by only b, there are no other beacons in F_e.

Now consider the following two cases:

- *F_e has at least one high arity node.*
  Let h be a high arity node in F_e. Since e is the only edge connecting the subgraph to the rest of the graph, h must be high arity with respect to a node in the subgraph itself. Hence from Lemma 4, b cannot monitor the entire subgraph. Also, no other beacon outside F_e can completely monitor F_e, as all paths from such beacons must traverse e; just like b, such beacons can not deterministically guarantee the last edge in their paths to nodes in the cycle formed by h (Lemma 4). Since b should be able to monitor the entire subgraph as it is the sole beacon in it, we have a contradiction. It follows that all nodes in the subgraph must be single arity.
- *F_e has only single arity nodes.*
  If F_e contains only single arity nodes then all edges in the subgraph can be monitored by a single beacon outside the subgraph. This is because F_e is a tree, and is reachable only through e from a beacon outside F_e. Any path from an outside beacon to any node in F_e has a unique subpath after crossing e. Hence, all edges within F_e can be monitored by it. This implies that b is a redundant beacon, and removing b from B doesn't effect the monitorable edge coverage of the beacon set B. This contradicts the definition of B as an optimal beacon set.

Thus, there can not exist a single arity node in B. Hence, B is a subset of the set of high arity nodes. ■

Theorem 6 lets us reduce the set of potential beacons used in Algorithm 3 to the set of high arity nodes. This can lead to substantial computational savings. For instance, we show in Section VI (and Figure 6), that the fraction of single arity nodes in current ISP topologies can be quite high.

In [7] the authors have shown that the set of high arity nodes in a graph is a beacon set—though potentially a non-optimal set—when beacons are locally-flexible. We have strengthened this result by showing that the optimal beacon set is *always* a subset of the set of high-arity nodes (even with simple beacons). Hence, our pruning algorithm is potentially capable of finding smaller beacon sets for all topologies. In order to empirically evaluate the efficacy of our methodology, we next compute the beacon sets for a few real-world ISP topologies.

VI. Experimental Results

In this section, we empirically evaluate the performance of our beacon placement strategies. Specifically, we compare the sizes of beacons sets for several current ISP topologies, for which beacon sets are computed: (i) by the beacon placement solution with locally-flexible beacons suggested in [7]; (ii) by our beacon placement algorithms for simple beacons; and (iii) by our algorithms for locally-flexible beacons. We also study the overhead of incorporating dynamism in routing, by comparing our solution to that obtained in [4], which applies only to statically-routed networks.

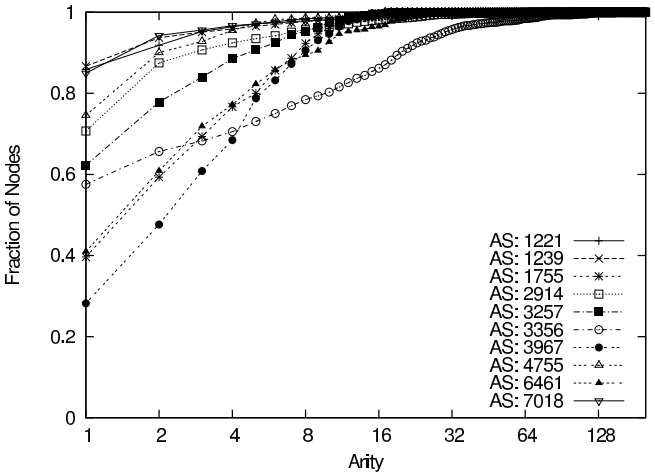


Fig. 5. Distribution of node arities for ten Rocketfuel topologies.

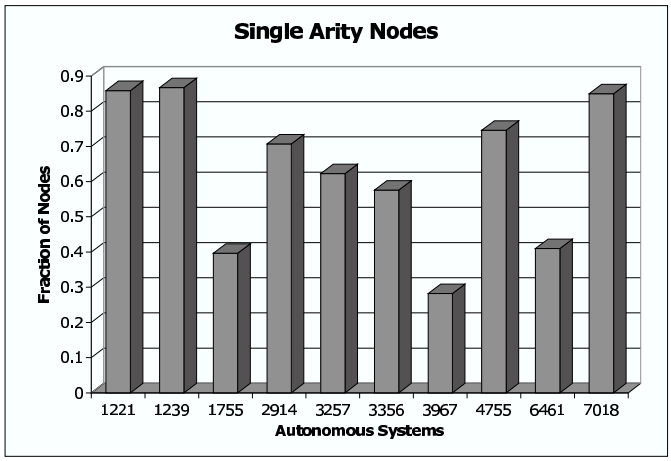


Fig. 6. Fraction of nodes which are single arity.

We implement and run all of these solutions on ten major ISP topologies obtained from the Rocketfuel project at the University of Washington [10] and present the results below.

A. Node Arities

We first study the arities of nodes in the ten topologies—Fig 5 plots the distribution of node arity in each. Fig 6 plots the fraction of nodes that are single-arity in each topology. We observe that:

- The distribution of node arities are quite different for different ISPs, indicating that ISP topologies can be quite diverse in their topological structure. In particular, some ISP topologies have a long-tailed arity distribution, indicating that only a handful of nodes have significant redundancy in the manner in which they connect to the rest of the network. For most topologies, a majority of nodes have arities within 20, although some nodes can have arities higher than 150.
- The fraction of single arity nodes in the ISP topologies varies from less than 30% to more than 85%. It is important to note that as the last edge on the path from every other node in the network, a single arity node has only one local edge that can be used to reach it. Single

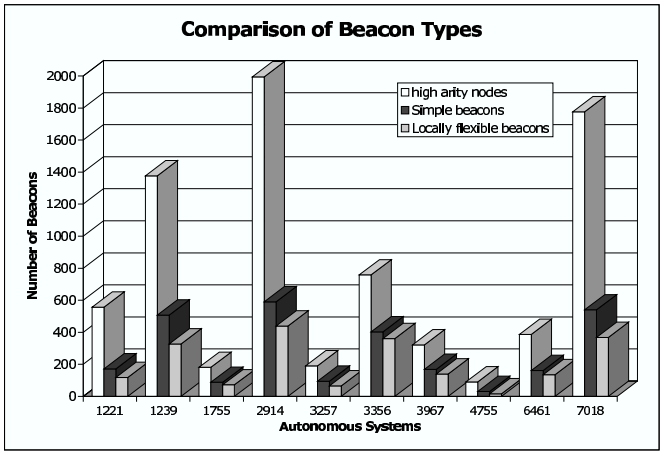


Fig. 7. Beacon set sizes (as absolute numbers) yielded by different strategies for the Rocketfuel topologies.

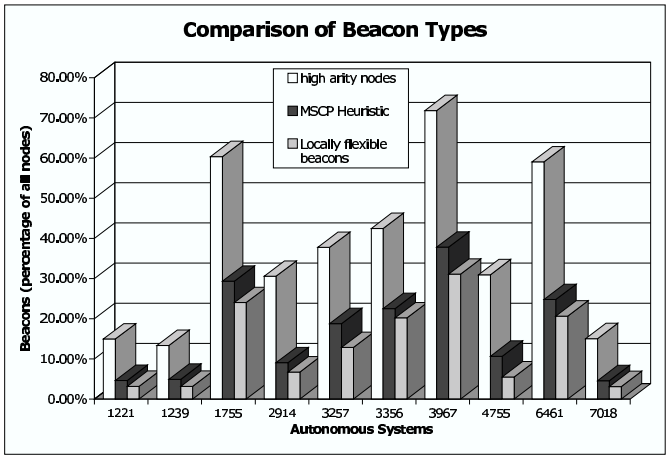


Fig. 8. Beacon set sizes (as fraction of total number of edges) yielded by different strategies for the Rocketfuel topologies.

arity nodes, therefore, are not robust to failures of local links. We find that for most topologies, more than half of the nodes have a single arity.

A large fraction of single-arity nodes also implies that the optimizations proposed in Section V to enable fast computation of beacon sets, can result in substantial savings.

It is important to note that Rocketfuel ISP topologies are subject to inference errors. In particular, [16] demonstrates that the inclusion of links that do not exist and the omission of links that are actually present can inflate path diversity in these inferred topologies. This also limits the accuracy of node arities computed above.

B. Beacon Set Sizes

It is important to mention that the Rocketfuel topologies for an ISP may not be single connected graph (possibly due to lack of data about some links). Thus, some of the topologies we analyze have multiple (independent) connected components. More importantly, some of the components consist of only single-arity nodes (such components have a tree structure). For a fair comparison with the previous work in [7], which does not apply to single-arity networks, we ignore such components when computing beacon sets. This eliminates only a small fraction of nodes from each of the ISP topologies.

For any ISP topology, we add up the sizes of beacon sets computed for each of the remaining components to get the total beacon set sizes—$|B_{HA}|$, $|B_S|$, and $|B_{LF}|$—for the three solutions being compared. Figs 7-8 plot the histograms of these beacon set sizes for the ten topologies. We observe that:

- *Beacon sets with locally-flexible beacons.*
  Our beacon placement solution for locally-flexible beacons reduces the beacon set sizes yielded by [7] by $50 - 80\%$. More importantly, we find that some major ISP topologies can be completely monitored, independent of routing state, using less than a hundred locally-flexible beacons. This is an encouraging observation as it suggests that a tomography-based monitoring infrastructure may be feasible even for major ISP topologies.
- *Beacon sets with simple beacons.*
  Even with simple beacons, our beacon placement solution reduces the beacon set sizes of [7] by $40 - 70\%$. This suggests that it may be feasible to design a simpler monitoring infrastructure that does not require that network nodes use different transmission rules for probe packets (as is required with locally-flexible beacons).
  This conclusion is further supported by the comparison of beacon set sizes yielded by our solution for simple vs. locally-flexible beacons, which indicates that locally-flexible beacons may not yield significant gains for many major ISP topologies.

C. Overhead of Allowing Dynamic Routing

The beacon-placement framework and strategy developed in [4] applies only to networks that employ *static* routes—the authors assume that the IP route taken by packets between any two nodes is fixed and known. This assumption ensures that the DMES of a node consists of *all* links that comprise the IP routing tree sourced at the node. Such routing-based DMESs are likely to be much larger than the topology-based DMESs formulated in our framework. Consequently, the beacon set sizes yielded by the strategy of [4]—henceforth, referred to as SB—are likely to be much smaller.

In order to quantify this, we evaluate the performance of SB on the Rocketfuel topologies. Unfortunately, the routing policy of these networks is not known publicly—consequently, it is not possible for us to obtain the actual network routes (which are needed by SB). Instead, we compute one possible set of routes by using the shortest path routing algorithm after assigning uniform weights to all edges. Fig 9 plots the resultant beacon set sizes obtained using SB (Shortest path tree) and compares it to our placement strategy with simple beacons (MSCP heuristic). We find that the beacon sets obtained using SB for statically-routed networks are much smaller (by 10-75%) than those obtained using our strategy for dynamically-routed networks.

It is important to note, however, that strategies like SB can not be applied to networks that allow dynamism in IP routing. For instance, several ISPs are known to employ load-balancing mechanisms that lead to routing dynamism even in

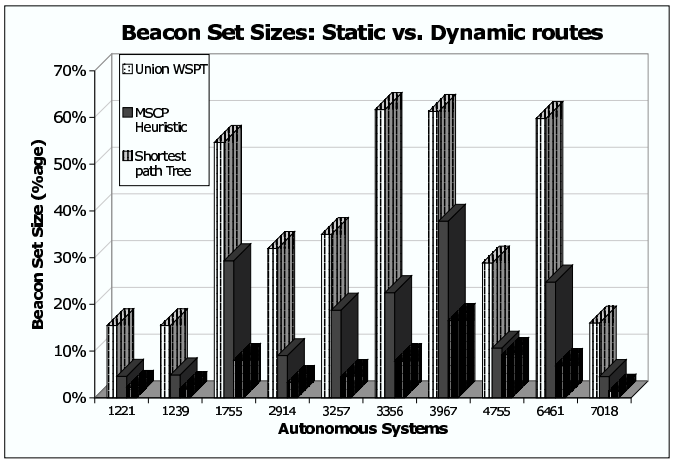


Fig. 9. Beacon set sizes (as fraction of total number of edges) with routing-dependent vs. topology-dependent placement.

the absence of link failures. The beacon set computed for a network using SB is likely to change if routes change, and new beacons would then have to be deployed. To illustrate this, we re-run SB for 10 different randomly-computed instances of network routing state in the Rocketfuel topologies. Each instance is obtained: (i) by assigning weights randomly (from 1 to 10) to all edges, and (ii) by re-running the shortest-path routing algorithm. We compute the beacon set for each of the routing states and plot the size of the union of the 10 beacon sets in Fig 9 (Union WSPT). We find that the union is much larger than the beacon set obtained using our strategy. The implies that for networks where traffic loads vary randomly (and which change link weights to reflect traffic loads), a routing-dependent strategy like SB is likely to require beacon sets much larger in the long-run than those yielded by topology-dependent strategies like ours or that of [7].

VII. Improving Optimality of Simple Beacon Sets

Our results in Section VI indicate that, in practice, using locally-flexible beacons instead of simple beacons offers only a slight improvement in the size of the computed beacon sets. In this section, we conduct additional analysis for *simple* beacons, that lets us derive approximate solutions to the corresponding BMP, that have better optimality bounds than those presented in Section IV. We conduct both theoretical and experimental analysis to evaluate the new beacon sets.

Our analysis in this section relies primarily on establishing the "cut" property of a DME of a simple beacon. Intuitively, this means that when such a DME is removed from a graph, the graph gets disconnected—note that this is true only for the DMEs which were found by the depth first search algorithm and not the ones that were included in the DMES simply because they were incident on a beacon. Since the "cut" property of a DME is a global property of the graph (and not dependent on the beacon under consideration), we also show that such DMEs are present in the DMES of all potential simple beacons in the network.

Lemma 5: A DME found using the depth first search part of Algorithm 1 is a cut edge. Also, such a DME is present in the DMES of all the nodes in the graph.

Proof: Using algorithm 1's formulation, edge e is the only edge which connects v to the subgraph containing u. Thus, if e is removed from the graph, the subgraph containing u will be disconnected from the rest of the graph. Thus e is a cut edge.

Let the end-nodes of e be v and w (the depth first search was run from v). Now consider the resultant subgraph layout if a depth first search is done from w. The edge e would be a single edge connecting an entire subgraph containing v. Now the DME e was monitorable by all the nodes in the subgraph containing u. By a depth first search on w, we see that e is also monitorable by the entire subgraph containing v. These two subgraphs collectively contain all nodes. Thus e is monitorable by all nodes in the graph. Consequently, e is present in the DMES of all nodes. ■

Lemma 5 motivates and allows us to define a set of all such DMEs. We then establish a structure on the DMES of all the nodes in the graph using the following definition.

Definition 3: The *cut edge set*[8] of a connected graph $G(V, E)$ is a set of edges Z such that if any edge $e \in Z$ is removed from G, then G is disconnected.

Theorem 7: The DMES of a simple beacon node u in a graph $G(V, E)$ is of the form $E_u \cup Z$, where E_u are the edges incident on node u and Z is the cut edge set of G.

Proof: Let us consider Algorithm 1 for finding the DMES of a simple beacon node u. S, which finally becomes the DMES of u, is initially an empty set. S is populated first with with the edges neighboring to u. At this stage $S = E_u$. Next, we run a depth first search on *all* the nodes and include cut edges in S. From lemma 5, the set of such edges is Z. Thus, the DMES of u is $E_u \cup Z$. ■

Note that Theorem 7 has an important implication for tree networks. All edges of a tree are cut edges. Thus, for a tree graph $G(V, E)$, $Z = E$. From Theorem 7, the DMES of *any* node in the graph is the entire set of edges E. This agrees well with our previous observation in Section V-A that single-arity networks can be monitored using a single beacon.

A. Revisiting Beacon Minimization

The additional structure on the DMES of simple beacons allows us to use an algorithm to minimize the beacon set, that results in a tighter bound on the optimality. In this section we will show how to use the Vertex Cover problem to do so.

Note that since the DMES of a node is now $E_u \cup Z$, the presence of a single simple beacon in the network ensures that all the edges in Z can be monitored. Thus, we can essentially remove the edges in Z from the graph and use the DMES of a node as E_u to come up with a beacon set. We will next show that finding an optimal number of beacons as described above is NP-Complete, by reducing the Vertex Cover Problem [13] to the Beacon Minimization Problem.

Theorem 8: The Beacon Minimization Problem (BMP) for simple beacons is NP-Complete.

Proof:

[8]In graph theory literature, the term *bridge* is also commonly used for a cut edge.

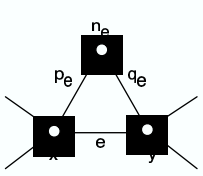
BMP is in NP: We have shown how to obtain the DMES for all the nodes in the beacon set in polynomial time. Hence taking a union of the DMESs and comparing it with the set of all edges allows us to verify in polynomial time if the solution is correct. This proves that the BMP is in NP.

Reduction from Vertex Cover: Consider an instance of a vertex cover problem with a graph $G'(V', E')$. In general, this graph may have one or more cut edges. We construct an instance of BMP from this setup as follows:

• For every cut edge e (with end nodes x and y) in E', add a node n_e and edges p_e and q_e so that e, p_e and q_e form a triangle with p_e and q_e incident on n_e. Now, the resultant graph (say $G''(V'', E'')$) doesn't have any cut edges, as removing any of the edges e still keeps G'' connected.

• Now, consider the Beacon Minimization Problem on graph G''. Since G'' has no cut edges, the DMES of a node u in G'' reduces to E''_u (the edges incident on u).

• Now, consider a solution to the BMP on G''. We have a set of nodes B such that all the edges are "covered" ($\bigcup_{u \in B} E''_u = E''$). Note that the sense of "covered" used here is exactly the same as the sense of "vertex covered" in the Vertex Cover problem.



To get a solution for the Vertex Cover problem from a solution to the BMP, consider any of the triangles (e, p_e, q_e) that were introduced above.

• To cover all the three edges, we need exactly two nodes (between x, y and n_e). Thus, two of these three nodes are present in B. If n_e is present in B then it covers both p_e and q_e. Hence, removing n_e, p_e and q_e from G'' (and n_e from B) ensures that the resultant graph can still be optimally "covered" by B. This is because n_e was not covering any other edge other than p_e and q_e.

• If n_e is absent from B then we first check if putting n_e in B and removing either x or y from B leaves any of the edges in the graph uncovered. If it doesn't then we have two optimal solutions to the beacon placement problem and we use the argument given above. However, if the transformation doesn't give us another optimal solution then it means that x and y are single handedly "covering" more edges then just p_e and q_e. Thus removing n_e, p_e and q_e from G'' still ensures that all the edges are covered optimally.

• We repeat this for all the n_e, p_e and q_e that we introduced to get a solution to the Vertex Cover problem on $G'(V', E')$.

Since Vertex Cover is NP-Complete [17] the above implies that BMP is also NP-Complete. ■

We use a known approximate solution to the Vertex Cover problem for deriving the following algorithm which yields an approximately-optimal beacon set.

Algorithm 5 has a known optimality bound of a constant 2 [17]). Note that this constant bound of 2 does not depend on the edge selection heuristic used (select*) in the above algorithm. Below, we enumerate three heuristics that we study next.

1) *Heuristic 1 (random):* We select the edges in the order they appear in the Rocketfuel data. The Rocketfuel data is arranged in an adjacency list format with node ids followed by the list of nodes adjacent to it. There is no specific ordering to these nodes besides that they are sorted based on their identifier value. In this heuristic, we select the edge formed by the first node listed in the pruned graph and the first node in the adjacency list of the node. This essentially means that the edge selection criteria is fairly random.
2) *Heuristic 2 (degree-order):* We sort the nodes of the graph according to their degrees; we also sort the nodes in the adjacency list (of a particular node) according to their degrees. We then use heuristic 1 on this preprocessed graph representation.
3) *Heuristic 3 (MSCP-adapted):* This heuristic maps directly to the Min Set Cover heuristic adopted in section IV. However, this is not an edge-selection heuristic, but a node-selection heuristic. We calculate the degree of all the nodes in the graph and select the one with the highest degree. We then include this node in our beacon set, remove this node (and its edges) from the graph and recalculate degrees.

Algorithm 5 Compute a beacon set using the approximate Vertex Cover algorithm.

Initialize B to be an empty set
Remove the edges in Z from G to get G'
while G' contains edges **do**
  Select* an edge from G'
  Include both its end nodes in B
  Remove the edge, its two end nodes and their edges from G'
end while
B is the required Beacon Set.

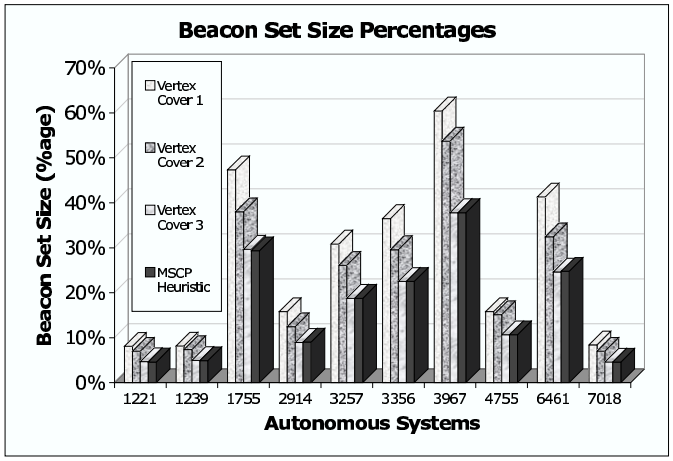


Fig. 10. Comparison of vertex cover heuristics for *simple* beacons

We next simulate these three heuristics on the Rocketfuel topology data and compare the resultant beacon sets with those obtained using Algorithm 3 for simple beacons (that latter is referred to as simply "simple beacons" below). The results are plotted in Figure 10.

We observe that even though all three vertex cover heuristics have a tight worst-case optimality bound (constant 2 when

compared to $1 + ln|E|$ for simple beacons), heuristic 3 and simple beacons yield the smallest beacon sets in our simulations. In fact, there is very little difference between simple beacons and heuristic 3.

The above empirical result also yield an interesting observation insight about the solution obtained by the best-performing heuristic (heuristic 3). We know that heuristic 1, being a vertex cover heuristic, never yields a beacon set larger than twice the optimal beacon set [17]. However, in most cases we find that the beacon set size yielded by heuristic 3 is quite close (but always greater) to half the beacon set size yielded by heuristic 1. This implies that heuristic 3 yields beacon sets with sizes close to the optimal beacon set, for all of the topologies considered in our experiments.

VIII. Incorporating Additional Network Scenarios

A. Incorporating Half-Duplex Links

Throughout the analysis presented so far, for brevity, we have used an undirected graph model to present our algorithms and proofs. In practice, networks may connect two nodes using a pair of half-duplex links, rather than a single full duplex link. A network monitoring infrastructure should ideally be capable of monitoring both half-duplex links as separate entities. Below, we illustrate that our beacon placement solutions can be extended to such networks with slight modifications.

Changes in Network Model: We modify our network model by replacing each undirected link by two half-duplex links between the end-nodes. More formally, if $G(V, E)$ is the undirected graph, we derive $G'(V', E')$ from G as follows:

- $V' = V$;
- For all $e \in E$, add directed edges $e_1 = x \rightarrow y$ and $e_2 = y \rightarrow x$ to E', where x and y are the nodes connected by e in G.

Changes in Definitions:

- *Node Arity:* The definition of node arity remains the same except that the "outgoing edges" of a node now refers to the links directed away from the node.
- *Path:* A path now consists of directed links. This also holds for the definition of physically connected paths.
- *DME:* The definition of a DME should now be considered with the new definition of paths.

Changes in Algorithms: The algorithms to find the DMES of a beacon (Algorithms 1 and 2) change in the following way. Prior to conducting a depth first search on node v to find all possible paths from the beacon u to node v, we need to reverse all links of the graph so that at the end of the search, we get paths from u to v; (in the undirected graph this didn't matter as the set of paths from beacon u to node v is the same as that from node v to beacon u). In practice, however, since all directed links are paired with another link in the opposite direction, we do not actually need to reverse the graph to find the subgraph of nodes—we should only ensure that the edge which finally gets included in the DMES of the beacon is the edge going into node v.

The pruning algorithm requires no changes since it is completely abstracted away from the beacon capabilities and network routing mechanisms used. To argue about the applicability of our optimizations to the new model we first present an observation for directed graphs.

Lemma 6: The existence of a high arity node in the directed graph as constructed in Section VIII-A implies that we have a cycle. Conversely, a cycle implies the existence of a high arity node.

Proof: Let u be high arity with respect to v. This means that we have distinct paths p_1 and p_2 from u to v both of which start from different outgoing edges of u. Since every directed link is paired with an oppositely directed link between the same nodes, we have paths p_1' and p_2' from v to u. Let us assume that we chose a v such that the first edges in paths p_1' and p_2' differ. Now paths p_1 and p_2' (and also paths p_1' and p_2) combine to form a cycle.

Now consider the case when we have a cycle in the graph. We break the cycle into any two paths p_1 and p_2. Like the argument above, we would have a reverse path for both p_1 and p_2 which makes the nodes at which we broke the cycle, high arity. ■

The above formulation keeps our single-arity network optimization intact. Having no high arity nodes means that our network has no cycles. This is even stronger than a directed acyclic graph as, because of the way links are connected, any physical loop in the network becomes a cyclic path. The graph looks like a tree except that all edges are actually paired.

The above formulation also keeps the optimization of considering just the high arity nodes for the beacon set, intact. In the proof, a high arity node is said to imply a cycle which has already been proved above for directed graphs.

B. Incorporating Non-Transit Nodes

Our formulation so far assumes that any simple physical path between two nodes may be selected by the routing protocol as the network route taken by packets sent between the two nodes. While this is a reasonable assumption for a single autonomous network, it might not be true in a backbone network which consists of nodes (that provide access to customer sub-networks) which refuse to carry any *transit* traffic—that has both source and destination outside the customer sub-network. Note that while the backbone ISP does not monitor the customer sub-network, it would want to monitor the access links to such non-transit customers. In order to incorporate such backbone networks, we extend our DMES-finding algorithms in this section.

We model non-transit networks in the form of a node in the network. The invariant to be modeled is that any valid *routable* path in the network can have a non-transit node only either in the beginning or the end of the path. In the following, we do so by modifying our DFS-based traversals to stop the "depth-recursion" as soon as a non-transit node is encountered.

The change in the algorithm is in the depth first search; if the depth first search algorithm encounters a non-transit node, it doesn't proceed depth-wise further, but returns back to its parent node in the depth first search tree. This essentially means that non-transit nodes are always *leaves* in a depth first search tree. We claim that the above algorithm gives us a valid

Algorithm 6 Find the DMES of a simple beacon u in a graph $G(V, E)$ containing non-transit nodes.

Consider a modification of algorithm 1.

```
Initialize S to be an empty set
for all edges l neighboring u do
    Include l in S
end for
for all nodes v in V do
    Do a modified depth first search from v such that on
    visiting a non-transit node we don't proceed further
    recursively but we retract to the parent node in the DFS
    tree; we don't stop at the root node if the root is a non-
    transit node. (we get a set of subgraphs each connected
    to v by one or more edges)
    if u lies in a subgraph connected to v by only a single
    edge e then
        Include the edge in S
    end if
end for
S is the DMES of u
```

optimal DMES for the beacon u in $G(V, E)$ containing non-transit nodes.

Proof: (Proof of correctness) Since non-transit nodes don't allow transit traffic, any valid path in $G(V, E)$ now can contain a non-transit node only at the start or the end. Consider a graph G' which is the same as $G(V, E)$ minus all non-transit nodes and its neighboring edges. Note that G' might not be connected even if G is. Let us denote the set of paths between the node pair (x, y) by the symbol $P_{x,y}$. Because of the characteristic of the non-transit nodes, the set of paths for a node pair (x, y) in $G(V, E)$ can be described as;

- if x and y are not non-transit nodes, then the set of paths $P_{x,y}$ remains the same in both G and G'.
- if x is a non-transit node, then the set of paths $P_{x,y}$ in G is equal to $\bigcup_{x'}[(x, x') + P_{x',y}\ in\ G']$ where x' is not a non-transit node and adjacent to x.
- if x and y are non-transit nodes, then the set of paths $P_{x,y}$ in G is equal to $\bigcup_{x',y'}[(x, x') + P_{x',y'}\ in\ G' + (y', y)]$ where x' is a normal node and adjacent to x and y' is a normal node and adjacent to y.

We have to prove that if and only if two nodes x, y in $G(V, E)$ are connected by a network route, then there exists a path in the (modified) depth first tree which connects the two nodes. We consider the following cases:

- *Case 1:* If x and y are normal nodes, then from the above, existence of a non-empty $P_{x,y}$ in G' implies a non-empty $P_{x,y}$ in G. Also, any path in $P_{x,y}$ doesn't contain non-transit nodes. Hence, our modified depth first search algorithm connects these two nodes.
  Conversely, if $P_{x,y}$ in G' is empty, then (because initially G was connected), we can conclude that all paths in $P_{x,y}$ had non-transit nodes in the middle. Our modified depth first search algorithm will not connect x and y as on all paths it finds a non-transit node on and halt further depth-wise graph search from that point onward.
- *Case 2:* If any of x and/or y are non-transit nodes, then derivation of $P_{x,y}$ in G from $P_{x',y'}$ in G' is given above. This enumerates all paths that can exist between x and y in G assuming these are the only non-transit nodes present in the network. Without loss of generality, let us assume that x is a non-transit node; y may or may not be a non-transit node. If $P_{x,y}$ is non-empty in G, then there exists a non-empty $P_{x',y'}$ as described above. A non-empty $P_{x',y'}$ implies that x' and y' are connected by our modified depth first search algorithm in G. This follows directly from Case 1. Since our modified depth first search algorithm connects x to x' (depth first starting from root doesn't stop at root because the root is a non-transit node), it also connects x to y'. Also, since our modified depth first search algorithm connects any non-transit node to a normal neighbor node, it joins y' and y. Thus our modified depth first search algorithm connects x and y.
  Conversely, if $P_{x,y}$ is empty in G, it means that for all neighbors of x (and possibly y if y is a non-transit node), $P_{x',y'}$ is empty. From Case 1, we can conclude that our modified depth first search algorithm doesn't connect any of those x' and y'. Thus, x and y are not connected by our modified depth first search algorithm.
- *Case 3:* A trivial border case which is not easily evident from the above is about two non-transit nodes x and y, directly connected. Our modified depth first search algorithm correctly connects them.

Using the above connectivity proof, along with the proof of correctness of Algorithm 1, we establish the correctness of Algorithm 6. Though the above is shown only for simple beacons, the same connectivity argument also holds for the DMES algorithm for locally-flexible beacons. ■

C. Robustness to k beacon failures

A tomographic monitoring infrastructure should ideally be robust to beacon failures; in particular, there should be no links in the network that can be monitored by only a single beacon. To achieve robustness to beacon failures, therefore, a monitoring infrastructure may wish to instantiate additional (possibly redundant) beacons in the network. A beacon set is said to be robust to k beacon failures if any subset of the beacon set containing k less beacons is able to monitor the complete network.

In [4], the authors have presented a technique for computing such robust beacon sets, using an extended version of the Min Set Cover problem. We believe their solution is general enough to be applicable to the beacon-placement framework presented in this paper. A detailed treatment of this scenario is, however, beyond the scope of this paper.

IX. Concluding Remarks

In this paper, we have presented a generic framework for addressing the problem of beacon placement for network monitoring using tomography, efficiently and under a generic set of policy constraints. Our framework incorporates routing dynamism and monitoring policy constraints by defining the

generic concept of Deterministically Monitorable Edge Sets, and then using the DMESs to find beacon sets that are within a bounded-range of the optimal size for monitoring a given set of network edges. We extend our basic framework to incorporate networks consisting of half-duplex links as well as non-transit sub-networks.

We supplement our theoretical analysis with empirical results obtained from running beacon placement strategies on Rocketfuel versions of real-world ISP topologies. We find that our algorithms perform significantly better than previous techniques and reduce the beacon set size by about 50 to 80%. We also find that the best-performing heuristic used for placement of "simple" beacons, yields beacon sets fairly close to optimal.

Acknowledgments: The authors thanks the anonymous JSAC reviewers for their valuable feedback and suggestions.

REFERENCES

[1] CAIDA, "Skitter," http://www.caida.org/tools/ measurement/skitter.
[2] M. Coates, A. Hero, R. Nowak, and B. Yu, "Internet tomography," *IEEE Signal Processing Magazine*, May 2002.
[3] K. Claffy, T. Monk, and D. McRobb, "Internet tomography," *Nature*, January 1999.
[4] Y. Bejerano and R. Rastogi, "Robust monitoring of link delays and faults in ip networks," in *Proceedings of the 2003 ACM INFOCOM*, April 2003. [Online]. Available: citeseer.ist.psu.edu/bejerano03robust.html
[5] N. Duffield and F. L. Presti, "Multicast inference of packet delay variance at interior network links," in *INFOCOM*, vol. 3, 2000, pp. 1351–1360.
[6] N. Duffield, J. Horowitz, F. L. Presti, and D. Towsley, "Network delay tomography from end-to-end unicast measurements," *Lecture Notes in Computer Science*, vol. 2170, pp. 576–595, 2001.
[7] J. Horton and A. Lopez-Ortiz, "On the number of distributed measurement points for network tomography," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference (extended abstract)*, October 2003, pp. 204–209.
[8] R. Kumar and J. Kaur, "Efficient beacon placement for network tomography," *(extended abstract) in Proceedings of Internet Measurement Conference*, October 2004.
[9] P. Barford, A. Bestavros, J. Byers, and M. Crovella, "On the marginal utility of network topology measurements," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, November 2001, pp. 5 – 17.
[10] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *Proceedings of ACM SIGCOMM*, vol. 32, August 2002. [Online]. Available: citeseer.ist.psu.edu/spring02measuring.html
[11] P. et. al., "Rfc 791: Internet protocol," *Request for Comments*, pp. 17–18, 1999.
[12] H. Nguyen and P. Thiran, "Active measurement for multiple link failures diagnosis in ip networks," in *Proceedings of Passive and Active Measurement Workshop (PAM)*, April 2004.
[13] T. Cormen, C. Stein, R. Rivest, and C. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
[14] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.
[15] T. Cormen, C. Leiserson, and R. Rivest, "Theorem 35.4, minimum set cover," *Introduction to Algorithms, Second Edition*, 2001.
[16] R. Teixeira, K. Marzullo, S. Savage, and G. Voelker, "In search of path diversity in isp networks," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, October 2003.
[17] T. Cormen, C. Leiserson, and R. Rivest, "Theorem 35.1, vertex cover," *Introduction to Algorithms, Second Edition*, 2001.

APPENDIX

Proof: (Proof of Correctness) Consider the case when u is high arity. Then there exists a node (say v) to which there are more than one paths from u with different outgoing edges from u. In a depth first search from u, one of these edges will be selected first for traversal. In the depth first search run, consider that we are on the first edge from u which has a path to v, with an alternating path to the same node v as defined above. Because of the alternating path, there exists a cycle with nodes u and v on it. Since the edge on the alternating path outgoing from u is not yet traversed, it will become a back edge and will be detected as outlined in the algorithm.

Now consider that node u is not a high arity node. Also assume that we detect a back edge in the way outlined in the algorithm. Since the back edge was to node u itself, it means that node u is part of a cycle with two outgoing edges participating in the cycle. Thus, there is at least one node in the network (from the same cycle) which has two paths to u each path having a different outgoing edge from u. This means that u is a high arity node. This is a contradiction. Hence if u is an arity one node then one cannot detect a back edge as described in the algorithm. ■

Time Complexity: The complexity of depth first search is $\Theta(|E| + |V|)$. Detecting a back edge involves going through all the neighbors of u. If the degree of u is k, then the cost for checking for a back edge is $\Theta(k)$. Since k is bounded by $|E|$, the time complexity of our algorithm is $\Theta(|E| + |V|)$.

Algorithm 7 Finding out if a node, u in $G(V, E)$ is high arity or not.

```
Do a depth first search from u;
if there is a back edge to u from any other node then
    declare u high arity
else
    declare u single arity
end if
```

Ritesh Kumar is a Graduate Student in the Department of Computer Science at the University of North Carolina at Chapel Hill. His research interests lie in computer networks, and especially in the design of end-to-end transport protocols.

Ritesh received his B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, in 2003.

Jasleen Kaur is an Assistant Professor in the Department of Computer Science at the University of North Carolina at Chapel Hill. Her research interests lie in the design and evaluation of networks and operating systems. Jasleen is a recipient of the NSF CAREER Award, 2004, and the UNC Junior Faculty Development Award, 2004.

Jasleen received her B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, in 1997, where she was also awarded the Motorola Student of the Year Gold Medal. She earned an M.S. and Ph.D. in Computer Sciences from the University of Texas at Austin in 1999 and 2002, respectively. She is a recipient of the J.C.Browne Graduate Fellowship and the M.C.D. Fellowship from the University of Texas at Austin.