End-to-end Service Quality Measurement Using Source-routed Probes

Fei Li
Department of Computer Science
Columbia University
New York, NY 10027
Email: lifei@cs.columbia.edu

Marina Thottan
Center for Networking Research
Bell Labs
Murray Hill, NJ 07974
Email: marinat@research.bell-labs.com

***Abstract*— The need to monitor real time network services has prompted service providers to use new measurement technologies, such as service-specific probes. Service-specific probes are active probes that closely mimic the service traffic so that it receives the same treatment from the network as the actual service traffic. These probes are end-to-end and their deployment depends on solutions that address questions such as minimizing probe traffic, while still obtaining maximum coverage of all the links in the network. In this paper, we provide a polynomial-time probe-path computation algorithm, as well as a 2-approximate solution for merging probe paths when the number of probes exceed a required bound k.**

Our algorithms are evaluated using ISP topologies generated via Rocketfuel. We find that for most topologies, it is possible to cover more than 98% of the edges using just 5% of the nodes as terminals. Our work also suggests that the deployment strategy for active probes is dependent on cost issues, such as probe installation, probe set-up, and maintenance costs.

I. Introduction

The emerging need to support real-time services and applications over a converged IP network has prompted service providers to monitor network wide service performance and service availability. In addition to having sufficient capacity to support the required throughput, service performance also depends on the impairments due to routing, such as reconvergence times and service disruption duration [21]. Thus, the performance and availability of a network service depends not only on available bandwidth, but also on router architectures and the design of the control plane. Evaluating the impact of network impairments on services can best be performed by end-to-end probes, which can mimic the behavior of the specific service being monitored. These end-to-end probes can track the changes of crucial performance parameters, such as network delay and loss. End-to-end probes fall in the general category of active probes.

Current measurement methods, those based on the standard Simple Network Management Protocol (SNMP)-based polling [35] or link-level measurements, cannot be used to model network services. SNMP only provides a device centric view of performance, while link-state metrics such as those based on *Hello* packets can only be used to detect link failures/availability. Unlike employing end-to-end probes, neither of these methods can be used directly to measure end-to-end delay, loss, and the impact that these performance metrics have on service quality. *In this work, we explore the issues surrounding the deployment of end-to-end network probes and design optimal probe deployment algorithms to detect service quality degradation over the IP path provisioned for the specific service.* For example, VoIP probes designed to monitor stringent end-to-end delay and loss requirements.

There are different types of end-to-end probe mechanisms [12], [26]: *one-packet* methods such as *pathchar* [22], the *packet pair* family [31], IP Management Protocol (IPMP) [27], and Internet Control Message Protocol (ICMP) [21]. *Pathchar* is used for estimating link bandwidths via round trip delays of packet sequences from successive routers [13]. Packet-pair methods can be used for estimating available bandwidth or the bottleneck link rate [9]. Using these 2 techniques to obtain end-to-end measurements requires that the measured data be correlated with the topology information obtained from *traceroute*.

IPMP is used for measuring one-way delay and is designed to overcome some of the limitations of the packet probes [27]. The IPMP protocol combines both path and delay measurements, thus alleviating the need for correlation with the *traceroute* measurement (path information is determined from the path record field in the IPMP packet, which is populated by the routers). In [28], the authors propose a user-based path diagnosis method that overcomes the limitations of *pathchar*. In that approach, two end-users of a flow cooperate to find performance faults that affect their flow. The users do not make use of any privileged network information, but make use of packet pair techniques such as ICMP. However the treatment received by probe packets, such as IPMP and ICMP at the individual routers, is dependent on the protocol used to encapsulate the probe packet. The network may prioritize this probe traffic differently from normal service traffic.

Thus, as noted in [28], any active measurement method that is out-of-band (not end-to-end); and does not look like application traffic might experience different network conditions as compared to the service traffic. Therefore, it cannot be used to measure service quality. An example of an out-of-band network measurement scheme was proposed by Padmanabhan et. al. [29] to identify faulty and malicious routing. In that scheme, the goal was to securely trace the path of existing traffic, thus preventing routers from misleading specialized

traceroute packets by treating them differently from normal traffic.

Our service monitoring framework comprises of source-routed probes crafted to mimic different network services (e.g. VoIP, Video-on-Demand) [8], [21]. Using the complete knowledge of network topology, the service provider can efficiently choose the MPLS probe paths that cover a set of interested edges, thus reducing the total probe traffic. We call this the *link-covering problem*. Service specific probes are sent along the computed probe paths at some predefined frequency, and checked for violations in terms of end-to-delay or loss metrics. Based on the probe paths that experienced performance degradation, a miscreant-link detection algorithm [30] can be initiated on the common set of links to isolate the link that is contributing to the performance degradation. Source-routed probes provide a deterministic map between the probe measurements and the associated links. Thus we can avoid the correlation problem with inaccurate *traceroute* data (due to transient congestions), as well as the limitation of path-record fields in IPMP.

Our framework assumes that, in general, most links have good operational performance and the goal is to identify the few links or nodes that may be encountering performance degradations. We believe that this approach will significantly reduce the total probing load on the network. The end-to-end probe design described in this work assumes complete knowledge of the topology and requires network support for source-routing mechanisms such as MPLS.

In this paper, we simultaneously address 2 *main issues related to deploying active network probes: (*$1.$*) to reduce the amount of probe traffic, and (*$2.$*) to minimize the cost of deploying probes*. The cost of probe deployments can be captured in terms of terminal costs (installing software for creating a probe terminal), as well as the path costs (capacity cost of using a specific link) [3].

We formulate our *link-cover problem* as a combinatorial graph problem where, we design a set of probes that can cover a specified set of edges that we are interested in, and every probe is *elementary* (does not traverse intermediate nodes more than once). The design of our minimal total cost probe-paths is obtained using a greedy approach. *This is a polynomial-time algorithm and is based on the idea of getting the minimal-cost source-destination path for each edge to be covered*. Notice that our algorithm generates probes without loops, and results in a set of probes with the minimal total cost. We also consider 2 variants of the link-cover problem: one is to minimize the maximal-cost of any probe while keeping the total number of probes be $\leq k$; the other is to minimize the number of probes while keeping the maximal-cost of any probe be $\leq l_{max}$. We show that these 2 variants are NP-hard; and, we design 2-approximation algorithms for them.

Our polynomial-time algorithm, as well as the 2 approximation algorithms are evaluated via the simulations on 5 of the largest Internet Service Provider (ISP) topologies obtained from the Rocketfuel project [34]. The evaluation criteria in each case are $(1.)$ the total cost of all probes designed, $(2.)$ the maximal cost of a probe when the total number of probes is fixed, as well as $(3.)$ the average cost of a probe.

We show that, our algorithms perform extremely close to the optimal solution when the probe *terminal* (the node that a probe should start from and end at) set is chosen to be the backbone nodes. In our prior work [30], we described and designed probes to be those paths (tours) derived from a Chinese Postman Tour covering all edges of a graph. However, the probes deployed in that solution had loops, which made the solution more tedious requiring a loop detection scheme and the use of heuristics to eliminate the loops. The elimination of loops is a necessary step for the practical implementation of the probes. On the contrary, *our current solution method explicitly accounts for the elimination of loops by finding probes that are elementary paths (tours) only — i.e. the probe path never intersects a transit (intermediate) node more than once*.

The rest of the paper is organized as follows: Section II presents related work, the probe models we consider, as well as our formal problem statement. In Section III, we introduce our algorithms along with the complexity analysis. Section IV presents the evaluation of the algorithms on ISP topologies, along with the discussion of the results and insights. We propose problems for future investigations, as well as the conclusions of our study in Section V.

II. Motivation and problem formulation

In this section we place our probe design work in the context of previous work and highlight the differences in our problem definition. We also provide an illustration of the the link-cover problem and provide a formal problem statement. Note that depending on context, we use the words edge and link interchangeably.

A. Related work

Network probing with low overhead has prompted a flurry of research activity in recent past. The IDmaps project [15] produces the latency maps of the Internet from which latencies of any arbitrary path can be obtained. However, since only a relatively few paths are actually monitored, it is possible to make errors in estimating the latencies of any arbitrary path. For the overlay network setting, the authors of [10] find the minimal set of paths to monitor so that the behavior of all paths can be inferred. Adler et. al. [1] provide a solution to compute the minimum cost set of multicast trees that can cover links of particular interest in the network.

Recently, the authors in [5] have provided algorithms for selecting probe stations such that all links are covered and computed the minimal set of probe paths that must be transmitted by each station, such that the latency of every link can be measured. However, the probe paths are computed via IP routes available from the probing stations. In [20], the authors consider the problem of probe-path design where they assume *local flexibility* — the probe-paths can be selected as either the current IP route or one of the current IP routes of the immediate neighbors. The efficient probe node (called *beacon*)

placement strategy proposed in [25] provides the minimum number of probe nodes required to deterministically monitor all network links even in the presence of dynamism in IP routes.

All of the above work on probe-paths and probe-node location has focused on IP routes as potential probe-paths. In the work presented here, the focus is on explicitly routed probe packets. A closely related work was presented in [6]. This work studies the problem of measuring path latencies through explicitly routed packet probes while minimizing the overhead imposed by the probe traffic. However, probe packets are generated using a different deployment architecture. In [6] all probes originate from a central point in the network. Our *link-cover algorithms focus on the design of probe paths and differs from [6], since we can choose probe paths (source-routed) that originate and terminate from any given set of* terminal nodes *in the network*. This new problem setting raises a series of natural questions: (1.) how to define a probe? (2.) how to find a minimum cost set of probes to cover a given set of edges? and (3.) what is the tradeoff between the number of probes and the cost of probes? In this paper, we address the above questions, by providing theoretical limits and qualitative evaluation of the algorithms through simulations over ISP topologies.

B. Problem description

Fig. 1 provides an illustration of the link-cover problem.

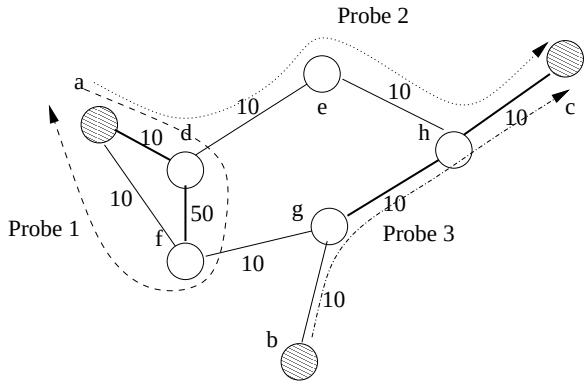

Fig. 1. An example showing the link-cover problem.

Consider the following scenario, in the given network, there are 8 nodes, $a - h$. a, b, and c act as the terminals and all others are regular nodes. Each probe we want to create must locate its end-points at either a, b, or c. Each network link has a cost (weight) as specified in the figure. The high-lighted edges are the network links of interest. We want to design a set of probes that covers all high-lighted (interesting) edges.

In this example, we have 3 probes, probe 1 goes from terminal a, traverses links (a, d), (d, f), and reaches back to terminal a; probe 2 goes from node a, traverses links (a, d), (d, e), (e, h) and arrives at terminal c. Probe 1 is a tour. Probe 3 starts from terminal b, runs across edges (b, g), (g, h) and ends at terminal c. All the links of interest are covered by these 3 probes. We use the sum of the weights of all edges of a path to represent the cost of the probe. Clearly, the total cost of the set of probes is $(10 + 50 + 10) + (10 + 10 + 10 + 10) + (10 + 10 + 10) = 140$, and the maximum cost probe is probe 1, whose cost is 70. From this example, link (a, d) is covered by 2 probes, and its weight counts twice to the total cost of these 3 probes. Note that a probe (elementary path) is allowed to have both its ends lie at the same terminal, like probe 1. Also, probe 2 is redundant in covering all specified edges. We can remove probe 2 and all interesting edges are still covered by probes 1 and 3, and the total cost is reduced to 100 from 140.

C. Problem formulation

In this subsection, we present a formal description of the *link-cover* problem. We model the network as a connected, undirected graph, $G = (V, E)$, V is the set of the vertices (which represents the nodes in the network), and E is the set of edges (which represent the network links); $|V| = n$ and $|E| = m$. Note that there is no self-loop in G: there is no edge that goes from one node and returns to the node itself, and G is not necessarily a planar graph. Without confusion, in graph G, we use node and vertex, link and edge interchangeably.

There is a cost function over each edge $e \in E$, $w_e : w_e \to \mathbb{R}^+$. This function denotes the cost of a probe when it uses a specific network link (edge). A set of specific nodes $\mathcal{T}$ in the graph are called *terminals*, $\mathcal{T} \subseteq V$. S is the set of edges that we are interested in, and $S \subseteq E$. Normally, $\mathcal{T} \neq \emptyset$ ($1 \leq |\mathcal{T}| \leq n$), and $S \neq \emptyset$ ($1 \leq |S| \leq m$). A *path* P is defined to be a set of concatenated edges between 2 nodes in V, and these 2 nodes are called *ends* of the path. If both ends of a path are the same node, the path can also be called a *tour*. Given a path P, we use $e \in P$ to denote the fact that path P contains (runs across) edge e. Except for end-nodes, all nodes that a path crosses are called *intermediate nodes*. Notice that, in the general case, a path may have *loops* — a loop is defined to be a sub-path that crosses an intermediate node more than once. *A path (resp. tour) not traversing any intermediate node more than once is called an* elementary *path (resp. tour); that is, an elementary path (resp. tour) is loop-less*. Also, the path from node $v_1 \in V$ to node $v_2 \in V$ is denoted as $v_1 \rightsquigarrow v_2$; and the cost of a path P is defined as $w(P) := \sum_{e \in P} w_e$.

A specific path called *probe* is defined to be an *elementary* path from one terminal $t_1 \in \mathcal{T}$ to another terminal $t_2 \in \mathcal{T}$. That is, both ends of a probe must be terminals. t_1 and t_2 are not required to be distinct; if $t_1 = t_2$, it results that the probe is a (loop-less) tour. The motivation of enforcing a probe to be elementary is that, typically, *a route with loops are not permitted in an IP network*; such a path will be rejected by the routers. Therefore, a feasible solution to the link-cover problem is to find a set of *elementary probe-paths* $\mathcal{P}$ such that every edge $e \in S$ ($\subseteq E$) is covered by at least one path $P \in \mathcal{P}$.

Input instance: given an undirected, connected weighted graph $G = (V, E)$. $S \subseteq E$ and $\mathcal{T} \subseteq V$. A mapping w_e exists, from each edge $e \in E$ to a non-negative number representing its cost, $w_e : w_e \to \mathbb{R}^+$; for each edge $e \in E$, $w_e \geq 0$. The set

of probe-paths $\mathcal{P}$ is defined as all such probes ($P = v_i \rightsquigarrow v_j$), $v_i, v_j \in \mathcal{T}$. For each edge $e \in S$, there exists at least one probe $P \in \mathcal{P}$, such that $e \in P$. Our optimization objectives can be represented as three subproblems:

1. Minimum-cost link-covering Problem (**LCP**): in which the goal is to minimize the total cost of the probes required. The cost of the probes is measured in terms of the traffic load on the network. That is,

$$\begin{aligned} \min \quad & \textstyle\sum_{P\in\mathcal{P}} w(P), \\ \text{subject to:} & \\ \forall e \in S, \quad & \exists P_j \in \mathcal{P}, e \in P_j, \\ \forall P_i \in \mathcal{P}, \quad & P_i \text{ has no loop.} \end{aligned}$$

2. Primal link-covering problem (**PLP**): in which the goal is to minimize the maximum cost of a probe. The cost of a probe can be measured in terms of latency, throughput, or length. That is,

$$\begin{aligned} \min \quad & \max_{P\in\mathcal{P}} w(P) \\ \text{where} \quad & |\mathcal{P}| \le k, \\ \text{subject to:} & \\ \forall e \in S, \quad & \exists P_j \in \mathcal{P}, e \in P_j, \\ \forall P_i \in \mathcal{P}, \quad & P_i \text{ has no loop.} \end{aligned}$$

   where k is a constant, and is the upper bound of the number of probes that can be used.
3. Dual link-covering problem (**DLP**): in which the goal is to minimize the total number of probes. The total number of probes reflects the cost of probe installation. That is,

$$\begin{aligned} \min \quad & k \\ \text{where} \quad & |\mathcal{P}| = k \\ \text{subject to:} & \\ \forall P \in \mathcal{P}, \quad & w(P) \le l_{max}, \\ \forall e \in S, \quad & \exists P_j \in \mathcal{P}, e \in P_j, \\ \forall P_i \in \mathcal{P}, \quad & P_i \text{ has no loop.} \end{aligned}$$

   where, l_{max} is the given maximal cost allowed for any probe, and k is the number of probes required.

III. Algorithms and analysis

In this section we describe the complexity of the three variants of the link-cover problem. Since **LCP** is not a NP-hard problem, we present a polynomial-time algorithm designed for **LCP** and a 2-approximation algorithm for **PLP** and **DLP** respectively. Before we proceed to introduce the algorithms, we first analyze the complexity (hardness) for **PLP** and **DLP**.

A. Complexity analysis for relaxations of PLP and DLP

Note that in our problem setting in Subsection II-C, we enforce a probe to be an elementary path or tour. *Even if we relax our problem setting, such that a probe is not necessarily an elementary path (can have loops),* **PLP** *and* **DLP** *are still NP-hard*. This can be easily generalized from the Min-Max k-Chinese Postman Problem (**MM** k**-CPP**), and the Capacitated Arc Routing Problem (**CARP**) respectively.

Briefly speaking, **PLP** is the generalization of **MM** k**-CPP** when we let $S = E$ and $\mathcal{T} = \{s\}$, where s is the depot node in **MM** k**-CPP**. At the same time, **DLP** is the generalization of **CARP** when we let $S = E$ and $\mathcal{T} = \{s\}$, where s is the depot node, and l_{max} is the vehicle capacity in **CARP**. We skip introducing **MM-**k**-CPP** and **CARP** since they are classical NP-hard problems and can be found in the references herein [2], [4], [16], [23].

B. NP-hardness of the primal link-cover problem: PLP

Please note that, if a probe is restricted to be defined as an elementary path (tour), we cannot induce **PLP**'s hardness directly from the Min-Max k-Chinese Postman Problem.

Now, consider the following problem (minimal *makespan* scheduling): given a set of tasks T, there are m *identical* machines (m is a fixed number), each task $t \in T$ requires time p_t to be finished, and any task can be run on any (identical) machine. The objective of this problem is to minimize the *makespan*, which is the maximal completion time for any machine in this schedule. This problem, minimizing makespan scheduling, is known to be NP-hard (even for m is 2) and there exists a simple greedy policy, which results in a 2-approximation algorithm [17], [36]. Also, a polynomial-time approximation scheme (PTAS) algorithm improves the approximation ratio to be $1 + \epsilon$ [19]. We will prove **PLP**'s hardness via a transformation to the problem of minimizing makespan scheduling.

Theorem 1: The decision version of **PLP** is NP-hard.

Proof: Theorem 1 is proved by the contradiction method via a transformation from the problem of minimizing makespan scheduling. Given any instance I of the minimum makespan scheduling problem, we shall prove that we can construct an instance I' to **PLP** (in polynomial-time), such that if I' satisfies **PLP** in polynomial-time, then I satisfies the decision version of the minimum makespan scheduling problem in polynomial-time. Therefore, assume there exists a polynomial-time algorithm for **PLP**, then, the problem of minimizing makespan over multiple machines would not be NP-hard since it can be solved via the algorithm for **PLP** in polynomial-time. Details of the proof follow:

Consider an instance I of the minimum makespan scheduling problem: there are m identical machines, and a set of jobs T. Each job $t \in T$ requires processing time of p_t. Now, we construct an instance I' to $\mathbf{PLP}$. In I', there is a graph $G = (V, E)$, $|V| = 2 \times |T| + 2$. Each job $t \in T$ corresponds to 2 nodes in graph G, and without loss of generality, we can view each job has a left node and a right node in the graph.

The weight of the edge connecting the left node and the right node is p_t.

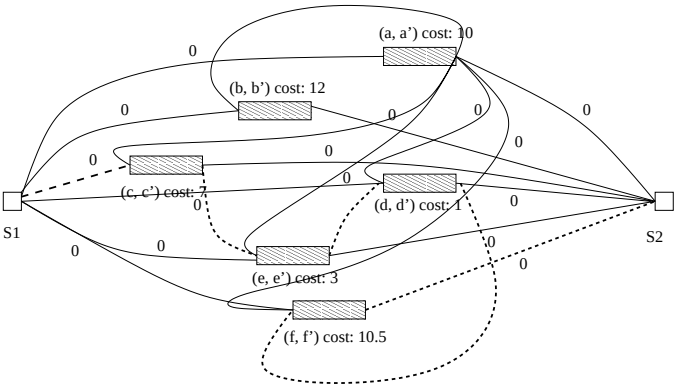

Fig. 2. A figure illustrating the proof of Theorem 1. Not all links are shown in this figure.

Suppose there are 2 specific nodes s_1 and s_2. $\mathcal{T} = \{s_1, s_2\}$. Now, we link s_1 to each job's left node and set the weight of the link to be 0; we also link each job's right node to s_2 and the weight assigned to that link is 0. Furthermore, we link each job's right node to all *other* jobs' left nodes and assign the weight of the link by 0. The corresponding edges (connecting a job's left node and right node) of all jobs consist of the set of interesting edges $S \subseteq E$. Till now, we create an instance I' to **PLP** in which only 2 terminals, s_1 and s_2, exist. Set S is the set of edges we want to cover and the number of probes to be used k is limited by m ($k \leq m$); and m is the number of identical machines in instance I to the minimum makespan scheduling problem.

Clearly, for this instance I', if we can construct a polynomial-time algorithm that will minimize the maximal cost of a probe, then, given the existence of $k = m$ probes (the more probes we use, the smaller the probe length), we see that in the instance I, all the edges belonging to the same probe lead to all corresponding jobs scheduled on one machine. Notice that there are $m = k$ machines to run these jobs. So, any instance to the minimum makespan scheduling problem can be converted in polynomial-time to an instance to **PLP**. Since the decision version of the minimum makespan scheduling problem is NP-hard, the decision version of **PLP** is NP-hard. So, Theorem 1 holds. ■

Refer to Fig. 2 as an example to the proof of Theorem 1. Given 5 edges to be covered, s_1 and s_2 are only 2 terminals in $\mathcal{T}$. The costs of traversing each edge are shown as those in the figure. Now, assume we limit the number of probes be $k = 2$, and 2 probes: $s_1 \rightarrow e(a, a') \rightarrow e(b, b') \rightarrow s_2$, and $s_1 \rightarrow e(c, c') \rightarrow e(e, e') \rightarrow e(d, d') \rightarrow e(f, f') \rightarrow s_2$ are found in a polynomial-time algorithm for **PLP**; with the maximal cost of a probe 22. So, correspondingly, in the minimum makespan scheduling problem, these 5 jobs can be optimally arranged in 2 identical machines, one is running jobs with processing time 10, 12; and one is running jobs with processing time 7, 3, 1, and 10.5. The optimal makespan is 22.

Next, we consider the NP-hardness of **DLP**. We notice that **DLP** is the counterpart of **PLP**, and, they have the same NP-hardness. Similarly, the hardness of **DLP** cannot be proved via a direct transformation from **CARP**. Consider a classical NP-hard problem: *bin-packing* problem. Given n items with sizes $a_1, a_2, \ldots, a_n \in \{0, 1]$, find a packing in unit-sized bins that minimizes the number of bins used. Bin-packing problem has been proved to be NP-hard [36]. From that, we have

Corollary 1: The decision version of **DLP** is NP-hard.

Proof: Corollary 1 can be proved via a transformation from the classical problem of bin-packing [11]: same as what we have constructed in the proof of Theorem 1, except that the weight of the edges we want to cover corresponds to the value of an item in the bin-packing problem. The maximal cost of a probe is limited by l_{max}; and $l_{max} = 1$ is the maximal capacity of a bin in the bin-packing problem.

Clearly, for this instance I', if we can construct a polynomial-time algorithm that can minimize the number of the probes, then, given the existence of each probe cost $\leq l_{max}$, we see that in the instance I, the number of probes reflects the number of bins to be used. So, any instance to the bin-packing problem can be converted in polynomial-time to an instance to **DLP**. Since the decision version of the bin-packing problem is NP-hard, the decision version of **DLP** is NP-hard also. So, Corollary 1 holds. ■

Given the fact that **PLP** and **DLP** are NP-hard, there is no efficient algorithm solving them. So, in Subsection III-C, we will design approximation algorithms for solving them. Some previously well-known results that will be useful in understanding our algorithms are, (1.) finding a Chinese Postman Tour in a mixed graph is NP-hard. However, there exists an efficient algorithm to calculate a Chinese Postman Tour in an undirected graph; (2.) there is an efficient algorithm for calculating the shortest-path between any 2 nodes in an *undirected*, *connected*, and *non-negative weighted* graph. If the weight of an edge is allowed to be negative, the shortest path problem is NP-hard; and (3.) there is an efficient algorithm for breath-first search or depth-first search. Using the breath-first search algorithm, we can get a simpler shortest path algorithm (in linear time) on *un-weighted*, *undirected* graphs. Based on the above mentioned techniques 2 and 3, we design a polynomial-time algorithm for **LCP** and a 2-approximation algorithm for both **PLP** and **DLP**.

C. Algorithms

In the following, we will design a polynomial algorithm that can compute a set of elementary probes $\mathcal{P}$, such that all edges in S are covered by at least one path $P \in \mathcal{P}$, and the total cost of the probes is minimized. We will also present a 2-approximation algorithm for **PLP**; with a little modification on the algorithm for **PLP**, we get a 2-approximation algorithm for **DLP**.

Without loss of generality, we use $t_1, \cdots, t_{|\mathcal{T}|}$ to denote the set of terminals, and we use (a, b) to denote the edge connecting node a and node b; a is the left node for the edge and b is the right node for the edge. Remember that,

in undirected graphs, there is no difference between left node and right node of an edge. Here, we specify "left" and "right" only for ease of illustrating our algorithms. *In the following, whenever we mention a shortest path, we are talking about the shortest path on a weighted, undirected graph; we refer to the shortest path as the probe (path) that is with minimal cost*.

1) **A polynomial-time algorithm for LCP***:* Before we introduce our algorithm, we have one obvious lemma saying:

Lemma 1: Given an edge $e = (v_1, v_2)$, a set of terminals $\mathcal{T}$, the shortest path from one terminal to the nodes v_1 and v_2, say $p := t_i \rightsquigarrow (v_1, v_2) \rightsquigarrow t_j$, is either an elementary path, or p is a tour that has only one loop. Here, $t_i, t_j \in \mathcal{T}$, t_i, and t_j are not necessarily distinct.

Proof: Please note that the shortest path stated in Lemma 1 may not be a probe though both its ends are terminals — since a probe is required to be only an elementary path (tour). The proof of Lemma 1 is straightforward: given a path $p := t_i \rightsquigarrow (v_1, v_2) \rightsquigarrow t_j$, first of all, we notice that there is no other terminals in-between $t_i \rightsquigarrow v_1$, (v_1, v_2), and $v_2 \rightsquigarrow t_j$. Otherwise, the given path is not a shortest path from the nodes of the given edge (v_1, v_2) to a terminal in $\mathcal{T}$. The reason is that since we apply the shortest path algorithm (on non-negative weighted graph) in locating the terminal for v_1, so, once we meet a terminal for v_1, any other possible terminal will have a greater distance to v_1. Therefore, there is no terminal other than t_1 and t_2 in p.

If both nodes of the edge e, v_1 and v_2, have one same terminal as their destination of the shortest paths, exactly one loop is generated. Lemma 1 holds. ■

Lemma 1 serves the purpose of finding the shortest probe that covers only one edge. We notice that, given an edge $e = (v_1, v_2)$ ($e \in S$), and given the set of terminals $\mathcal{T} \subseteq V$, $|\mathcal{T}| \leq n$, there are at most $|\mathcal{T}|^2$ pairs of terminals that can serve as the probe-ends for a probe covering edge e. Also, if the shortest path to v_1 is given, say $P_1 = t_i \rightsquigarrow v_1$, in order to avoid generating a loop, other than $t_i \in \mathcal{T}$, any intermediate node in the path, $v \in P_1$ ($v \neq t_i$) cannot be one of the intermediate nodes in the path from v_2 to a terminal, $v_2 \rightsquigarrow t_j$. Therefore, once we get the shortest path from one terminal to v_1, to avoid generating loops, we need to remove all intermediate nodes V' in the path, and remove all edges associated with those intermediate nodes E' from the graph $G = (V, E)$. Then, given the remaining graph $G' = (V - V', E - E')$ (V' is the set of intermediate nodes in the shortest path from one terminal to v_1, and E' is the set of edges associated with those nodes), we apply the same procedure to find a shortest path P_2 from v_2 to another terminal $\in \mathcal{T}$. Clearly, if such a path exists, $P_1 \rightsquigarrow P_2$, it is an elementary path (tour).

Remember that, we should take care of the *order* of picking v_1 and v_2. Without loss of generality, we index all terminals, and if node v_1 has the same distance to more than one terminal in $\mathcal{T}$, we select the one that is with a smaller index number. With such an approach, the procedure for locating the nearest terminal for a given node is fixed in polynomial-time. Also, if P_2 cannot be found, then, there is no elementary path (tour) containing the edge e.

Now, given $|\mathcal{T}|^2$ pairs of terminals, we can get at most $|\mathcal{T}|^2$ different elementary path (tour) covering each edge $e \in S$. Then, among those $|\mathcal{T}|^2$ elementary paths (tour) we calculated, we can determine which one is the shortest one covering the edge $e = (v_1, v_2)$. We will select it as the shortest elementary path that we assign to e. Therefore, followed by Lemma 1, we have,

Lemma 2: Given an edge $e = (v_1, v_2)$, a set of terminals $\mathcal{T}$, there exists a polynomial-time algorithm to find the shortest (minimum-cost), and elementary path from $t_i \rightsquigarrow (v_1, v_2) \rightsquigarrow t_j$. $t_i, t_j \in \mathcal{T}$ and t_i, and t_j are not necessarily distinct.

Proof: Given a terminal $t_i \in \mathcal{T}$, there exists a polynomial-time algorithm to find the minimal-cost path from v_1 to t_i. Then, we remove all intermediate node in the path and their associated edges, from the remaining graph, we apply the same algorithm to get the minimal-cost path from v_2 to a terminal $t_j \in \mathcal{T}$. We conclude that the concatenated path (tour) (a path from a terminal to v_1, edge e, and a path from v_2 to a terminal) is elementary. Among all feasible concatenated paths or tours (at most $|\mathcal{T}^2|$ paths or tours since both v_1 and v_2 can be the first node in finding a path in the procedure), the minimal-cost one can be found and it serves as the probe covering edge (v_1, v_2).

Now we claim that, any shorter (less cost) path or tour that covers edge (v_1, v_2) will result in a loop. Assume a shorter path exists (say, an optimal path), it has 3 parts, without loss of generality, we assume it consists of a path from one terminal to v_1, edge e, and a path from v_2 to one terminal. So, the optimal path from one terminal to v_1 is the same as one we get; note that with the same distance to different terminals, the node will choose the one with the smallest indexed number.

Refer to Fig. 3 for an example illustrating the proof of Lemma 2.

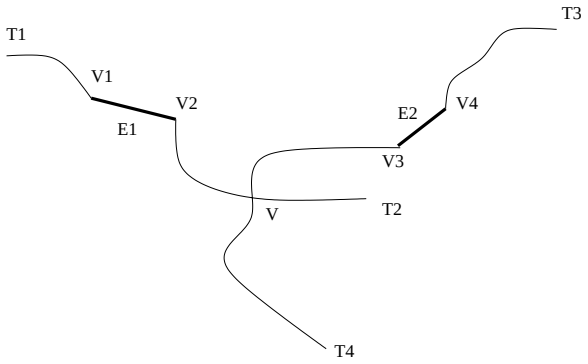

Fig. 3. A figure illustrating the proof of Lemma 2.

Assume there are 2 edges we want to cover $e_1 = (v_1, v_2)$ and $e_2 = (v_3, v_4)$. We get 2 elementary paths to cover them and one path covering e_1 starts from terminal t_1 and ends at terminal t_2; the other path covering e_2 starts from terminal t_3 and ends at terminal t_4. Assume these 2 paths interact at node v. Now, we consider the path covering e_1, when it goes from node v, it chooses terminal t_2 instead of terminal t_4 as its end

since the path length from v to t_4 is larger than the path length from v to t_2. At the same time, the path covering e_2 chooses t_4 instead of t_2 as its terminal, so, the path length from v to t_4 is shorter than that of the path from v to t_2. These 2 conclusions conflict, thus, we state that our assumption is wrong. *So, any* 2 *elementary paths calculated from the shortest-distance algorithm, cannot intersect with each other*.

In summary, if there exists a shorter path from v_2 to one terminal, then the path should intersect with one intermediate node in the path to v_1. Therefore, it turns out that the path is not elementary. One thing that we should note is, if path P_1 and P_2 share an edge, they must share at least one node. From the above analysis, Lemma 2 holds. ■

¿From Lemma 1 and Lemma 2, we immediately have,

Theorem 2: There is a polynomial-time algorithm for finding a set of probes $\mathcal{P}$, such that all interesting edges are covered by the probes and $\sum_{P\in\mathcal{P}}$ w(P) is minimized.

Proof: Theorem 2 can be proved by a construction method directly from Lemma 1 and Lemma 2. Lemma 1 and Lemma 2 provide a polynomial-time procedure to find an elementary path (tour) covering any given edge e. The above lemmas provide the background for finding the minimal total-cost probes covering all edges in S. Notice again, the minimal-cost path from one node to a given terminal can be calculated using any shortest-path algorithm over an undirected, non-negative weighted graph. Once a probe for edge e is located, all interesting edges in this probe can be removed from S. Among those paths with interested edges on it that are also covered by other probes, we remove that probe from $\mathcal{P}$. We repeat this until there is no such probe in the set $\mathcal{P}$. The above steps remove the redundancy of the probes covering all required edges. Notice that the greedy approach reduces the total cost of the probes in each step and each newly generated probe cannot be replaced by the set of all probes generated already. So, we can apply the same procedure to find the shortest path covering the remaining edges in S. We repeat this until set S is empty.

Then, we get a set of elementary probe paths $\mathcal{P}$ — a path covering one or more interesting edges. Finally, we claim the above procedure results in a polynomial-time algorithm for **LCP** in minimizing the total cost of all probes. Theorem 2 holds. ■

The algorithm describing how to find the set of probes with the minimal total cost of probes written in pseudo-code is presented in Algorithm Link-covering for **LCP**.

2) **A** 2**-approximation algorithm for PLP***:* Notice that the proof of Theorem 2 is a constructive proof. Assume the number of elementary shortest paths we get is k', we realize that if the number of probes for all edges in S, $k' \leq k$, we can say that we have found the optimal solution to **PLP**, which is the maximal-cost of the probe in the probe-sets we found. If $k' > k$, we need to reduce the number of probes by merging some of them until the number of elementary paths (tours) is k. In order to merge 2 elementary paths, we claim that, under a bounded cost for the maximal-cost probe, 2 elementary paths (probes) can be merged into one elementary path (probe).

Algorithm 1 Link-covering for **LCP**($G = (V, E)$)

```
Given a graph G = (V, E);
Index all terminals;
The set of interesting edges is S;
for each edge e := (v_l, v_r), e ∈ S do
  for each terminal t_i ∈ T do
    Find a shortest path P_1 = v_l ⇝ t_i along with the
    indexed distance of a node;
    Choose the nearest terminal, ties broken in favor of the
    smaller indexed terminal;
    Remove all intermediate node V' in P_1, between v_l
    and t_i from graph G;
    Remove all associated edges for node set V';
    On the remaining graph G = (V − V', E − E'), find
    the shortest path P_2 = v_r ⇝ t_j, t_j ∈ T;
  end for
end for
Among all elementary paths or tours, choose the one with
minimal total cost;
Denote it as probe P_e for edge e;
Associate each node in P_e with the difference between its
shortest distance to v_l and v_r and the total cost of the edges
it covers on that path;
Use associated distance as the shortest path calculation;
Remove all interesting edges e', S ← S − e', e' ∈ P_e;
Mark e and all removed edges as Y to P_e;
if f has been covered by other probes and f ∈ P_e then
  Mark f as N;
end if
for each probe P_i ∈ P do
  if ∀f ∈ P_i, f is marked N, f ∈ S then
    Remove the heaviest probe P_i;
  end if
end for
```

Before we proceed to our approximation algorithm, we give several definitions that help us illustrate our algorithms clearly and efficiently. Briefly speaking, for our approximation algorithm for **PLP**, we first get a series of elementary probes as we did in the algorithm for **LCP**, then, if the number of probes k' is more than what we can afford, say $k' > k$, we will merge probes such that the number of probes is reduced, while we sacrifice the maximal-cost of a probe in the probe set. The following definitions are related to the procedure of merging 2 probes.

Definition 1: **Merge distance**: the minimal cost of the final probe after connecting 2 probes via linking one node in one probe to another node in the other probe.

Clearly, from our discussion in Subsection III-B, there exists an efficient algorithm (polynomial-time) to identify the cost of linking 2 probes. Please note here, not every pair of probes can be merged into one longer probe.

Definition 2: **Shared edge**: if several probes ($\geq$ 2 probes) cross an interesting edge $e \in S$ that we want to cover, edge e

is said to be shared by all the probes that are incident on it. If an edge only has one probe to cover it, that edge is *necessary* to that probe.

Clearly, we know that given any 2 probes, we can link them together while leaving (possible) shared edges aside. Based on the above definitions, we have,

Lemma 3: Given any 2 elementary paths (tours) (there may exist other terminals on the path since such a path may be concatenated by several elementary paths covering a given edge), there exists a method that concatenates these 2 elementary paths and still get an elementary path.

Proof: Lemma 3 can be proved by the contradiction method. For each elementary path, the left or right node of an edge has the minimum-cost to the nearest terminals in the elementary path we find. If 2 elementary paths P_1 and P_2 have a node v that both of them traverse, we can conclude that for the given node v in these 2 paths, there are 2 terminals that both have minimal-cost to it. Refer to Fig. 4 for an illustration.

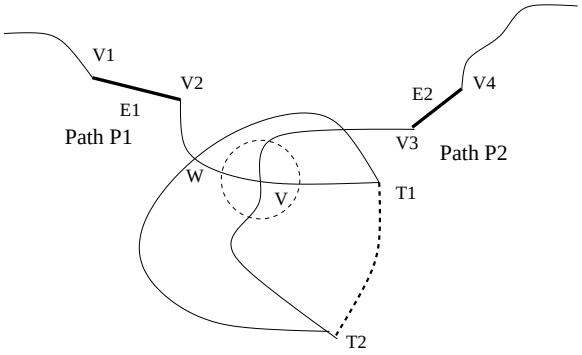

Fig. 4. An example showing the proof of Lemma 2.

Both path P_1 and path P_2 share the same node v, e_1 is the edge that P_1 is supposed to cover, and e_2 is the edge that P_2 is supposed to cover. $t_1 \in \mathcal{T}$ and $t_2 \in \mathcal{T}$ are the terminals for P_1 and P_2 end. Now, investigate the paths from v to t_1 and from v to t_2. Since when we locate the probe P_1 for edge e_1, P_1 ends at terminal t_1, so, $length(v \rightsquigarrow t_1) < length(v \rightsquigarrow t_2)$; otherwise, P_1 will end at t_2 based on the order of the terminals and the distance of the terminals we consider. In the same manner by analyzing the path P_2, we conclude $length(v \rightsquigarrow t_1) > length(v \rightsquigarrow t_2)$. This conflicts with the conclusion we made before. Therefore, the elementary paths P_1 and P_2 do not traverse the same node.

Furthermore, we can link t_1 and t_2 to generate an elementary path for both paths if there is no shared intermediate node (or no shared edges) in path P_1 and in path P_2 (as illustrated in Fig.2). If path P_1 and path P_2 have shared edges, there is no way to reduce the number of probes by removing P_1 or P_2 only (remember that, for any probe all of whose interested edges are shared has been removed from the probe set $\mathcal{P}$ by Algorithm Link-covering **LCP**). Based on the above analysis, Lemma 3 holds. ■

In the following, we present an approximation algorithm for **PLP**. Let $\mathcal{L}$ denote the maximum-cost of a probe path in a graph $G = (V, E)$ under the restriction that there are at most k probe paths (our solution). We use OPT to denote an optimal solution for the $\min w(P)_{P \in \mathcal{P}}$. The number of probes is $\leq k$, and A returns the maximum length of a probe.

First of all, we know that before we merge any probes together, after the Algorithm Link-covering for **LCP**, we get a series of elementary paths (tours) as the probes for the interesting edges. Assume the number is k'. If $k' \leq k$, we return the optimal maximal-cost of a probe. If $k' > k$, we should merge elementary paths in order to reduce the number of probes; while, we have the maximal-cost of the probe as the lower bound of OPT. The merge part is similar to the bin-packing problem if we regard the probe-length as the bin's load and the number of probes is limited as the number of bins, k. With the goal of reducing the complexity of the algorithm, we apply *optimal suit* method [11] to merge any 2 probes: if their *probe distance* is the smallest among all pairs of probes, these 2 elementary paths are merged if possible. The algorithm for **PLP** goes as follows.

Algorithm 2 Link-covering for **PLP** ($G = (V, E), k$)

```
Given a graph G = (V, E), # of probes ≤ k;
Apply Algorithm Link-covering for LCP;
if k' ≤ k then
   Return the maximal-cost P_max ∈ P;
else
   Calculate the merge distance for each pair of probes
   P_i, P_j ∈ P;
   while k' > k do
      Merge(P_i, P_j) with the minimal merge distance;
      k' ← k' − 1;
   end while
end if
```

The procedure for merging 2 elementary probes, we call Merge(P_1, P_2). The nearest nodes are defined to be the node pair $v_1 \in P_1$ and $v_2 \in P_2$, linking these 2 nodes makes the final probe connecting P_1 and P_2 the minimal newly generated probe cost.

Algorithm 3 Merge(P_1, P_2) in Algorithm Link-covering for **PLP** ($G = (V, E), \mathcal{P}$)

```
for ∀ shared edge e ∈ P_1 and e ∈ P_2 do
   if e is shared by another probe P then
      Link P_1 and P_2 at the connecting node v;
   else
      Return false for Merge(P_1, P_2);
   end if
end for
if P_1 and P_2 have no shared edges between them then
   Link the nearest node between them;
end if
```

Theorem 3: The algorithm *Link-covering for* **PLP** is 2-approximation.

Proof: Theorem 3 states a fact that the algorithm Link-covering for **PLP** works the same way as the optimal suit algorithm for bin-packing [36], if we regard the number of probes as the number of bins, and the maximal-cost of a probe as the maximal-load of a bin. Directly following the bin-packing algorithm's analysis, we can prove Theorem 3. The detailed proof goes as follows.

The lower bound of OPT (remember that we use OPT to denote the optimal maximal-cost of probe, instead of the optimal algorithm itself) corresponds to a set of k probes, such that each interesting edge $e \in S$ is covered by at least one probe, and the maximal-cost of a probe in such probe set is minimized to be OPT. Notice that, in each step we merge 2 elementary paths, the increase of the probe's cost is bounded by OPT as we choose the minimal merge distance. Also, we note the sum of the total cost of the covered edges to the nearest terminal is the lower bound of $k \times OPT$, and the cost of each probe to be merged is also the lower bound of OPT. So, in each step, the cost of any probe we generate is still bounded by $2 \times OPT$. Therefore, Theorem 3 holds. ■

3) **A 2-approximation algorithm for DLP***:* In the following, we introduce a 2-approximation algorithm for **DLP**. Remember that the variant **DLP** is the counterpart of the variant **PLP**.

Assume the maximal cost of a probe is limited by l_{max}, our goal is to minimize the number of probes to be employed. First of all, with a fixed initial number $k' = 1$ (as the number of probes used), we will apply the Algorithm Link-covering for **PLP** to estimate the maximal length of a probe we generate. If in the resulting probe set $\mathcal{P}$, we get the maximal cost of a probe is l_p, and if $l_p \leq l_{max}$, we claim that k' is the minimal number of probes to be used. If $l_p > l_{max}$, we double k' (k' is increased by 1 time), $k' \leftarrow 2 \times k'$, and then, apply the same algorithm for **PLP** to locate a set of probes and the maximal-cost of a probe in it. We repeat doing this until we end up with a set of probes $\mathcal{P}$ in which the maximal-cost of probe $\leq l_{max}$, we return k' and claim that $k' \leq 2 \times k$ (to be proved in Theorem 4).

Algorithm 4 Link-covering for **DLP** ($G = (V, E), l_{max}$)

```
Given a graph G = (V, E), the maximal-cost of a probe l_max;
k' = 1; l = ∞;
while l > l_max do
  k' = 2 × k;
  Apply Link-covering for PLP (G = (V, E), k');
  Get a probe set P and the maximal-cost l;
  if S is covered by P (a reasonable solution) then
    Return l;
  else
    l = ∞;
  end if
end while
```

Using Theorem 3 and following the same idea, we have,

Theorem 4: The algorithm *Link-covering for* **DLP** is 2-approximation.

Proof: Theorem 4 can be proved via following Theorem 3 directly. From Theorem 3, we know that, if the number of probes used is increased by 1 time, then, the Algorithm Link-covering for **PLP** returns the lower bound of the optimal maximal-cost of a probe, when the optimal algorithm has k probes to be used; our algorithm is allowed to employ $2 \times k$ probes. So, in Algorithm Link-covering for **DLP**, in the step $l \leq l_{max}$, the number of probes increases up to 2 times the number of probes that an optimal algorithm will use. So, Theorem 4 holds. ■

IV. Simulation results

Our *link-cover* solutions are evaluated on different realistic ISP topologies obtained from Rocketfuel project [34]. We choose to use the largest 5 topologies since they provide the most interesting analysis in terms of the complexity of the network being monitored. For each set of simulations we performed 5 runs (in some cases up to 10 runs) to obtain statistically significant results. Each link in the network is assigned a cost which is its inferred latency (link cost). In each case, we choose (a fraction of) the backbone nodes as the potential set of terminals.

Since for ISP topologies, the number of terminals (backbone nodes) is relatively large when compared with the total number of nodes (almost half), we restrict the choice of number of terminals to be 5%, 10%, and 15% of the total number of nodes $|V|$ of that topology. However, the chosen terminals were placed at randomly chosen backbone node locations. Also, the set of interested edges to be covered is chosen to be 25%, 50%, 75%, and 100% of the total number of edges in the network.

Table I shows the results of our polynomial-time algorithm **LCP** on different ISP topologies. The algorithm is used to compute the number of probes required to cover the interested edges using 5% and 15% of the total nodes as terminals. Two sets of interested edges comprising of 50% and 100% of the total edges are considered. In each case, we obtain the maximal-cost of a probe, the average-cost of a probe, as well as the total cost of all the probes. An important scenario to note is where (for Telstra and Sprintlink networks), not all edges randomly selected can be covered by a probe since the probe is required to be an elementary path linking 2 terminals. The specific question of the impact of the number of probe terminals on coverage is studied using the Telstra topology and the results are presented in Table II.

In the discussion of our algorithms and analysis, we considered the general case of undirected, and non-negative weighted graphs. So, the cost of probe traversal is assigned as a non-negative real number. Without loss of generality, and for simplicity in experimental design, we use the number of hops to denote the cost of the path. That is, we calculate the shortest path on an un-weighted graph.

¿From Table I and Table II, we see that as expected the number of probes almost doubles as the number of interested

edges increases from 50% to 100% of all edges in the network. Similar effect is observed in the total probe costs: covering additional edges with the same number of terminals increases the total probe costs, while increasing the number of terminals decreases total probe cost. We also observe that the maximal cost and the average cost of the probe decreases as the number of terminals increases. Furthermore, the values of these metrics are found to be comparable for all 5 topologies.

The evaluation of the 2-approx **PLP** algorithm is presented in Table III. Here, we are targeting the fixed k as $1/2$ of the number of probes obtained using **LCP**. The table provides a comparison of probe characteristics (number of probes, maximal-cost of a probe) obtained by the polynomial algorithm **LCP** and the number of probes obtained after merging using the **PLP** algorithm.

A. Discussion

As described in the introduction the goal of this work was to design optimal probe paths to cover the selected edges in a network. The optimality criteria was evaluated using metrics such as the total cost of the probes deployed, maximal cost of the probe, and the number of probes required to cover the edges. Under these evaluation metrics, we find that the **LCP** algorithm provides very good performance.

We note that with just 5% of the nodes as terminals we can cover almost 100% of the edges. Increasing the number of terminals to 15% does not provide any additional edge coverage, and reduces the total cost of the probes on the average by only 1.08%. This is an important finding since using fewer terminal nodes implies that fewer nodes need to be enhanced to host probe generation software (probe installation costs are minimized).

Using 5% of the nodes as terminals, and 100% coverage of links, the average cost of a probe is found to be 3.62; suggesting that the load induced by probing on the different network links is minimal. The goal of the **PLP** algorithm was to reduce the number of probes by merging the probe paths obtained using **LCP**. We find that after merging, the average percentage reduction in the number of probes is around 56%. This implies that the probe paths identified by our **LCP** and **PLP** algorithms are close to optimal.

We also note that, from Table II, for any given topology, there is *no linear relation between the number of terminals and the total number of probes or probe costs*. We believe that this effect is a consequence of the shortest path computation used to determine probe paths as well as some experimental factors involved with the random choice of the interested edges.

Figure 5 shows the relationship between the maximal cost of a probe and the number of terminals in the Telstra topology. We observe that, as the number of terminals increases, the maximum probe cost decreases. However, we reduce the maximal probe cost by 33% by increasing the number of terminals from 15% to 25%. Thus, it suggests that even in terms of minimizing the maximal probe cost, there is no need to use more than 15% of the nodes as terminals. Thus, we claim that for the topologies considered here, the gain

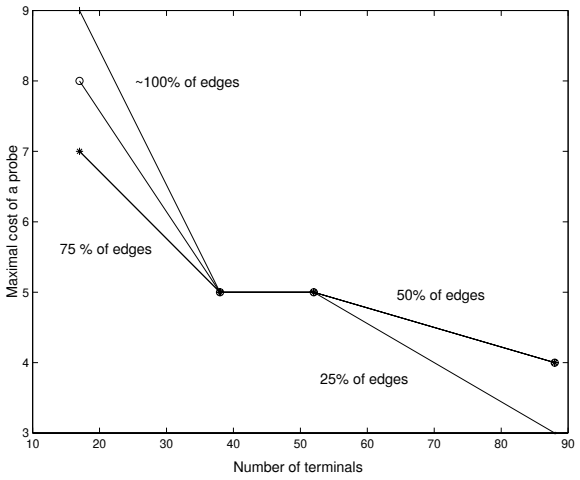

Fig. 5. Maximal-cost of a probe v.s. the number of terminals for the Telstra topology.

expected by solving the dual problem of **PLP**, namely **DLP**, may not be significant. Thus the two step design for probe based monitoring will consist of the **LCP** algorithm followed by **PLP**.

B. Impact from network topology

From Figure 6, we find that the number of probes required to cover the interested edge set is almost linearly dependent on the number of edges in the network. The observation is found to be true regardless of the number of interested edges or the number of terminals used. However, from Table I, we see that the number of nodes does not have any characterizable impact on the number of probes. We also observe that in the case of covering all edges, in two of the 5 topologies, increasing the number of terminals did not improve the edge coverage. The same number of links are left uncovered in both cases. We believe that this is a consequence of the location of these edges in the network with respect to the position of the (randomly) chosen terminal nodes.

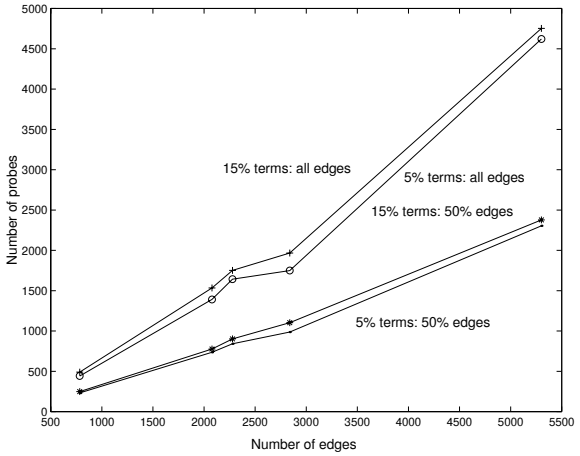

Fig. 6. The total number of edges in the network v.s. the total number of probes.

TABLE I

SIMULATION RESULTS FOR **LCP** (NUMBER OF TERMINAL NODES, AND PERCENTAGE OF EDGES COVERED VERSUS THE OPTIMAL TOTAL PROBE COST, AVERAGE COST PER PROBE, MAXIMAL COST OF PROBE AND THE NUMBER OF PROBES. T IS THE SET OF TERMINALS; E IS THE SET OF EDGES).

Name	$\|V\|$	$\|E\|$	used T (as %)	covered E (as %)	probe cost total	probe cost average	probe cost maximal	# of probes
Telstra (Australia)	351	784	17 (5%)	392 (50%)	934	3.98	8	235
				769 (98.1%)	1614	3.66	8	441
			52 (15%)	392 (50%)	707	2.79	6	254
				769 (98.1%)	1305	2.67	6	489
Sprintlink (US)	604	2279	30 (5%)	1139 (50%)	3340	3.93	10	851
				2277 (99.9%)	6250	3.84	10	1627
			90 (15%)	1139 (50%)	2550	2.85	6	894
				2277 (99.9%)	4883	2.80	8	1741
Verio (US)	972	2839	48 (5%)	1419 (50%)	3824	3.96	11	966
				2839 (100%)	6937	3.97	11	1748
			145 (15%)	1419 (50%)	3005	2.87	8	1048
				2839 (100%)	5445	2.774	8	1964
Level3 (US)	624	5301	31 (5%)	2650 (50%)	7592	3.23	11	2351
				5301 (100%)	15501	3.36	11	4662
			93 (15%)	2650 (50%)	6584	2.76	8	2384
				5301 (100%)	13107	2.76	7	4756
AT&T (US)	631	2078	31 (5%)	1039 (50%)	3223	4.28	10	755
				2078 (100%)	5802	4.15	12	1300
			94 (15%)	1039 (50%)	2457	3.00	8	820
				2078 (100%)	4534	2.97	7	1527

TABLE II

SIMULATION OF **LCP** ON TELSTRA TOPOLOGY WITH $|V| = 351$ AND $|E| = 784$. T MEANS THE SET OF TERMINALS; E MEANS THE SET OF EDGES).

used T (as %)	covered E (as %)	probe cost total	probe cost average	probe cost maximal	# of probes
17 (5%)	196 (25%)	452	3.80	7	120
	392 (50%)	934	3.98	8	235
	588 (75%)	1381	4.15	9	334
	769 (98.1%)	1614	3.66	8	441
38 (10%)	196 (25%)	416	3.12	6	134
	392 (50%)	739	3.02	7	245
	588 (75%)	1144	3.20	7	358
	769 (98.1%)	1388	2.96	6	469
52 (15%)	196 (25%)	397	2.94	6	134
	392 (50%)	704	2.78	6	255
	588 (75%)	1037	2.76	6	375
	769 (98.1%)	1326	2.72	6	487
88 (25%)	196 (25%)	325	2.23	4	146
	392 (50%)	614	2.22	4	276
	588 (75%)	885	2.20	4	403
	769 (98.1%)	1148	2.18	4	527

TABLE III

SIMULATION OF **PLP** ALGORITHM USING 15% OF NODES AS TERMINALS, AND COVERING ALL EDGES, AND $k \leq 1/2$ OF THE PROBES OF **LCP**. FOOTNOTE 1: DUE TO CIRCUIT TECHNOLOGY [34].

Name	$\|V\|$	average degree	$\|T\|$	# probes before merge	# probes after merge	maximal cost before merge	maximal cost after merge	Gain of **LCP** (percentage reduction in # of probes)
Telstra (Australia)	351	2.336	52	485	211	6	8	56.5%
Sprintlink (US)	604	3.77	90	1742	752	7	8	56.8%
Verio (US)	972	2.92	145	1964	863	8	8	56.0%
Level3 (US)	625	8.41[1]	93	4750	1963	7	10	58.7%
AT&T (US)	631	3.29	94	1534	682	7	8	55.5%

We also observe that the percent reduction obtained *using* **PLP** in the number of probes and the maximal cost of the probe is dependent on the degree of the network. We find that for the network with the highest degree of 8.41, the reduction after **PLP** is significantly smaller than the other 4 topologies which have an average degree in the range 2.336 to 3.29. Thus, we can say that *for networks with higher degree the polynomial-time* **LCP** *algorithm provides a nearly optimal solution*.

V. Conclusion

Based on our simulation study, we conclude that for any network topology, for covering almost 100% edges, only 5% of the nodes need to be assigned as terminal nodes. This is a significant finding for probe-based monitoring systems. Since there is no edge coverage advantage by adding additional probe terminals, the design of the probe-based monitoring system can be optimized based on just deployment costs. The deployment cost are in terms of the cost of terminal installations and the cost of setting up and maintaining probe paths. Furthermore, using the **LCP** algorithm we can obtain probe paths that provide nearly optimal results in terms of minimizing all three criteria: the number of probes, the maximal cost of the probe, as well as the average probe cost.

As part of our future work, we would like to explore the use of the **DLP** algorithm and also take into account topological issues, such as degree of network connectivity for further improvement of the **PLP** algorithm. The **PLP** and **DLP** algorithms presented in this paper can be easily mapped on to the minimum makespan scheduling problem, and bin-packing problem. Therefore, we believe that a $(1+\epsilon)$-approximation algorithm can be found, when k is a constant.

VI. Acknowledgement

The authors would like to thank Neil Spring, Tian Bu, Rajeev Rastogi, and Vahab Mirrokni for the comments and suggestions on the initial draft.

References

[1] M. Adler, T. Bu, R. Sitaraman, and D. Towsley, *Tree layout for internal network characterizations in multicast networks*, in Proc. of NGC, London, UK, Nov 2001.

[2] D. Ahr and G. Reinel, *New heuristics and lower bounds for the Min-Max k-Chinese Postman Problem*, in Proceedings of the 10th Annual European Symposium of Algorithms (ESA), September, 2002.

[3] S. Agrawal, P. P. S. Narayan, J. Ramamirtham, R. Rastogi, M. Smith, K. Swanson and M. Thottan, *VoIP Service Quality Monitoring Using Active and Passive Probes* in Proc. of First International Conference on Communication System Software and Middleware (COMSWARE) 2006, New Delhi, India.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation*, Springer, 1999.

[5] Y. Bejerano and R. Rastogi, *Robust monitoring of link delays and faults in IP networks*, in Proc of IEEE INFOCOM San Francisco, CA, USA. Mar 2003.

[6] Y. Breitbart, C. Y. Chong, M. Garofalakis, R. Rastogi, and A. Silberschatz, *Efficiently monitoring bandwidth and latency in IP networks*, in Proc of IEEE INFOCOM Tel Aviv, Israel, Mar 2000.

[7] T. Bu, N. Duffield, F. Lo Presti, and D. Towsley, *Network tomography on general topologies*, in Proc. ACM SIGMMETRICS, 2002.

[8] G. Prabhakar, R. Rastogi, and M. Thottan, *OSS architecture and requirements for VoIP networks*, Bell Labs Technical Journal, Vol. 10, No. 1, 2005.

[9] R. L. Carter and M. E. Crovella, *Measuring bottleneck link speed in packet switched networks*, Performance Evaluation, Vol. 27 & 28, pp 297-318, 1996.

[10] Y. Chen, D. Bindel, H. Song, and R. H. Katz, *An algebraic approach to practical and scalable overlay network monitoring*, in Proc of ACM SIGCOMM Portland, Oregon, USA. Aug 2004.

[11] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: a survey*, Approximation algorithms for NP-hard problems, 1997.

[12] C. Dovrolis, P. Ramanathan, and D. Moore, *What do packet dispersion techniques measure?*, in Proc. IEEE INFOCOM, Anchorage, Alaska, USA, April 2001.

[13] A. B. Downey, *Using Pathchar to estimate Internet link characteristics*, in Proc. ACM SIGCOMM, Cambridge, MA 1999.

[14] W. Fernandez and G. S. Lueker, *Bin packing can be solved within* $1+\epsilon$ *in linear time*, Combinatorica, 1981.

[15] F. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewwicz, and Y. Jin, *An architecture for global Internet host distance estimation service* in Proc. of IEEE INFOCOM 1999, New York City, NY.

[16] G. N. Frederickson, M. S. Hecht, and C. E. Kim, *Approximation algorithms for some routing problems*, SIAM Journal of Computing, Vol. 7, No. 2, May 1978.

[17] R. L. Graham, *Bounds on certain multiprocessing anomalies*, Bell System Technical Journals, 45:1563 - 1581, 1966.

[18] R. L. Graham, M. Grotschel, and L. Lovasz, edited, *Handbook of combinatorics*, Elsevier, 1995.

[19] D. S. Hochbaum and D. B. Shmoys, *Using dual approximation algorithms for scheduling problems: theoretical and practical results*, Journal of the ACM, 34:144 - 162, 1987.

[20] J. D. Horton and A. Lopez-Ortiz, *On the number of distributed measurements points for network tomography*, In Proc. of Internet Measurement Conference, IMC 2003.

[21] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, *Analysis of link failures in an IP back bone*, Internet Measurement Workshop, 2002.

[22] V. Jacobson, *Congestion avoidance and control*, in Proc. ACM SIGCOMM, Stanford, CA, USA, 1988.

[23] D. Jungnickel, *Graphs, networks, and algorithms*, Springer, 1999.

[24] N. Karmakar and R. M. Karp, *An efficient approximation scheme for the one-dimensional bin packing problem*, IEEE FOCS, 1982.

[25] R. Kumar and J. Kaur, *Efficient beacon placement for network tomography*, In Proc of Internet Measurement Conference, IMC 2004.

[26] K. Lai and M. Baker, *Measuring link bandwidths using a deterministic model of packet delay*, in Proc. ACM SIGCOMM, Stockholm, 2000.

[27] M. J. Luckie, A. J. McGregor, and H. W. Braun, *Towards improving packet probing techniques*, Internet Measurement Workshop, 2001.

[28] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, *User-level Internet path diagnosis*, in Proc of ACM SOSP, Bolton Landing, NY, USA. Oct 2003.

[29] V. N. Padmanabhan and D. R. Simon, *Secure traceroute to detect faulty or malicious routing*, in Proc of HotNets-I Princeton, NJ, USA. Oct 2002.

[30] S. Parthasarathy, R. Rastogi, and M. Thottan, *Efficient design of end-to-end probes for source-routed network*, Bell Labs Technical Memo. April, 2005.

[31] A. Pasztor and D. Veitch, *Active probing using packet quartets*, in Proc. Internet Measurement Conference, 2002.

[32] A. Reddy, R. Govindan, and D. Estrin, *Fault isolation in multicast trees*, in Proc. ACM SIGCOMM, 2000.

[33] Y. Shavitt, X. Sun, A. Wool, and B. Yener, *Computing the unmeasured: an algebraic approach to Internet mapping*, in Proc. IEEE INFOCOM 2000, Tel Aviv, Israel, Mar 2000.

[34] N. Spring, R. Mahajan, and D. Wetherall, *Measuring ISP topologies with rocket fuel*, in Proc of ACM SIGCOMM 2002.

[35] W. Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*, Addison-Wesley Longman Inc. 1999, (Third Edition).

[36] V. V. Vazirani, *Approximation algorithm*, Springer, 2003.

[37] D. B. West, *Introduction to graph theory*, Prentice Hall, 1996.