Managing End-to-end Network performance via

Optimized Monitoring Strategies

H. Cenk Ozmutlu

hco@uludag.edu.tr
Dept. of Industrial Engineering
School of Engineering and Architecture
Uludag University
Gorukle, Bursa, 16059, Turkey
++ 90 (224) 442 8178 Ext 1137, Fax ++90 (224) 442 9196

Natarajan Gautam,

ngautam@psu.edu
Dept. of Industrial and Manufacturing Engineering
Pennsylvania State University
University Park, PA, 16802, USA
++1 (814) 865-7601, Fax ++1 (814) 863-4745

Russell Barton

rbarton@psu.edu
Dept. of Industrial and Manufacturing Engineering
Pennsylvania State University
University Park, PA, 16802, USA
++1 (814) 865-7601, Fax ++1 (814) 863-4745

Suggested running head: Managing End-to-end Network performance via Optimized Monitoring Strategies

Managing End-to-end Network performance via Optimized Monitoring Strategies

H. Cenk Ozmutlu

hco@uludag.edu.tr
Dept. of Industrial Engineering
School of Engineering and Architecture
Uludag University
Gorukle, Bursa, 16059, Turkey

Natarajan Gautam, Russell Barton

ngautam@psu.edu, rbarton@psu.edu
Dept. of Industrial and Manufacturing Engineering
Pennsylvania State University
University Park, PA, 16802, USA

ABSTRACT

To predict the delay between a source and a destination as well as to identify anomalies in a network, it is useful to continuously monitor the network by sending probes between all sources and destinations. It is of prime importance to reduce the number of probes drastically and yet be able to reasonably predict delays and identify anomalies. In this paper we state and solve a graph-theoretic problem to optimally select a subset of traceroute-type probes to monitor networks. A sub-optimal algorithm, which works in polynomial time, is introduced as an alternative to the exponential-time graph-theoretic algorithm. The application of the constrained coverage problem on randomly generated topologies yielded an 88.1% reduction in the number of monitored traceroute-type probes on average. In other words, networks can be successfully monitored using only 11.9% of all possible probes. For these examples, the polynomial time sub-optimal algorithm has obtained the optimal (in 24 of 30 examples) or near optimal (second best solution in the remaining examples) solutions.

Keywords: Network management, quality of service, monitoring, end-to-end delay, graph theory

1. INTRODUCTION

The tremendous growth of Internet users and applications has resulted in severe congestion of the network [1],[2],[3]. This has led to users demanding a Quality-of-Service (QoS) for their network applications. In particular, users are interested in the end-to-end latency or delay and also in specifying a suitable route (or path; we will use path and route

interchangeably) that avoids by identifying congestion, if any, in the network. In this research, we concentrate on monitoring the network to answer two questions in real-time, namely, what will be the end-to-end delay in sending a message from a source to a destination, and, where are the "hot spots" or portions of the network with large delays.

A way to answer the two questions is by continuously monitoring the network and making predictions about the end-to-end delay as well as the location of the hot spots [4][5]. This monitoring requires sending probes between all sources and destinations in the network, which would not only produce redundant information but also potentially result in heavy traffic by using so many probes. A crucial decision is to choose an optimal subset of probes that is both small enough to handle (in terms of information management, since small files are easy to store and browse through quickly for monitoring) and not contribute to the congestion as well as large enough to make accurate predictions of delay localization of hot spots [6].

In this paper we use traceroute-like probes that give round-trip delay information between the source and every node in the route traversed by the probe from the source to the destination. Given these probes, the network topology, and the routes followed by packets between all sources and destinations, we formulate and solve a graph theoretic problem to optimally select a subset of probes.

We have not found any found any discussion in the literature on the graph-theoretic problem that we call the "constrained coverage problem". We develop a sub-optimal polynomial time algorithm to solve the problem. Using this algorithm we obtain a set of probes that can be used to continuously monitor the network to obtain information about the roundtrip delays between any two nodes and also to identify the hot spots in the network.

In Section 2 we describe the objective of this study, analyze the model's input variables and state the assumptions. In Section 3 we state the graph theoretic problem and formulate it as a mathematical programming problem. In Section 4 we introduced the sub-optimal algorithm and present examples. In Section 5 we state the conclusions of this work.

2. PROBLEM DEFINITION AND INPUT ANALYSIS

The objective of this research study is to select the minimum subset of source and destination (S-D) pairs from all possible S-D pairs for continuous probing. The delay data obtained from the subset of probes sent between the subset nodes will be used to calculate the delays between any S-D pair in the topology.

The probe type can be one of many alternatives (traceroute, ping, etc.) each providing different levels of detail on delay. For example, ping (Unix version of ping (ping -s "destination")) provides round-trip end-to-end delay, but no information about delays on specific arcs along its path from the source to the destination. On the other hand, traceroute provides round trip delays from the source node to all the nodes along its path, which can be used to calculate delays on each arc of the traceroute path. Therefore, in this study, we use traceroute-like agents as the probe type. For sake of simplicity, we refer to such probes as, simply, traceroutes.

2.1. Probing Using Traceroute-like Probes

It is important to note that some arcs might belong to more than one path. Assume an arc is in *n* paths. Sending probes over these *n* paths will provide *n* pieces of delay information about the arc. However, only one sample of delay information on each arc is sufficient for monitoring purposes at a given time. It is essential to arrange probes in such a way that all the arcs are covered at least once while minimizing the number of probes. The following Three-Node Example (Figure 1) shows how to extract delay data from each arc using traceroute probes:

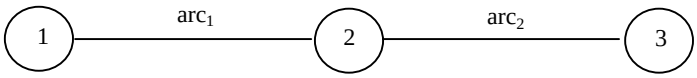

Figure 1: Topology of the Three-Node Example

Three probes are necessary (between 1 and 2, 1 and 3,and 2 and 3) to cover all the S-D pairs in the topology if round-trip delays are the same in opposite directions of any path. If round-trip delays are not the same, there will be six S-D pairs instead of three (between nodes 1-2, 2-1, 1-3, 3-1, 2-3, 3-2). Moreover, every arc should be replaced by two one-direction arcs as shown in Figure 2. Modeling the inequality of round-trip delays on opposite directions of paths will double

the number of arcs and S-D pairs in the topology. For simplicity, we assume that round-trip delays are same on the opposite directions of each path for the remaining part of this study.

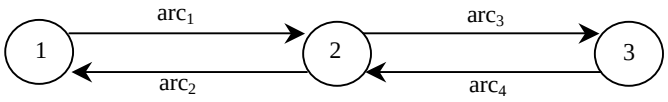

Figure 2: Topology of the Three-node example with one-directional arcs

If a traceroute is sent from Node 1 to Node 3, the following delay data will be obtained (see Figure 3):

a) delay over the trip Node1-Node2-Node1

b) delay over the trip Node1-Node2-Node3-Node2-Node1

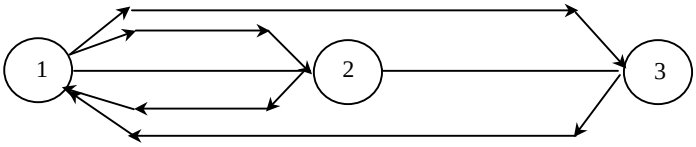

Figure 3: Delay data from traceroute between 1 and 3

The delay data in (a) is what we would have if we sent a traceroute between Node 1 and Node 2. So, the only data we need is the delay data corresponding to the traceroute between Node2 and Node 3. Let's examine the topology of the Three-Node Example in more detail (see Figure 4).

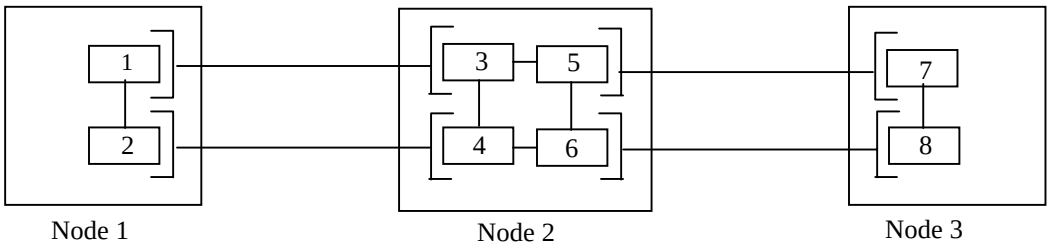

Figure 4: Buffers in the topology of the Three-Node Example

The delay information recorded using traceroutes between Node 1 and Node 3 ($traceroute_{13}$) is due to the wait in the buffers 1-3-5-7-8-6-4-2. $Traceroute_{23}$ has to use the buffers 5-7-8-6. Interesting enough, these are the four buffers in the

middle of $traceroute_{13}$. The remaining buffers (1-3 and 4-2) are the buffers used by $traceroute_{12}$ (the delay data in (a)). $Traceroute_{13}$ and $traceroute_{12}$ do pass through buffers 1 and 3 at the same time, so delay at those buffers are exactly same for both traceroutes. However, the delay at buffers 4 and 2 in $traceroute_{12}$ is not exactly the same as the delay at the buffers 4 and 2 in $traceroute_{13}$, because $traceroute_{12}$ and $traceroute_{13}$ do not enter (and leave) buffers 4 and 2 at the same time. Although, there will be an error factor, Equation (1) will provide us with a close estimate of the delay on $traceroute_{23}$.

$$traceroute_{23} = traceroute_{13} - traceroute_{12} \pm \varepsilon \quad (1)$$

By sending enough traceroute probes to cover all the arcs in the topology (in the Three-Node Example, it is possible to achieve this with only one traceroute probe), sufficient information will be collected to make conclusions on the delay of any arc or node in the topology. Consequently, delay on any given path can be easily calculated.

2.2. The Looping Effect

It is known that sometimes packets take different routes from source to destination than from destination to source [2]. Such instances create a loop (or loops) in the traceroute path. We now explain, using a Six-Node Example (see Figure 5), the problem that arises due to the loops and suggest a solution to the problem.

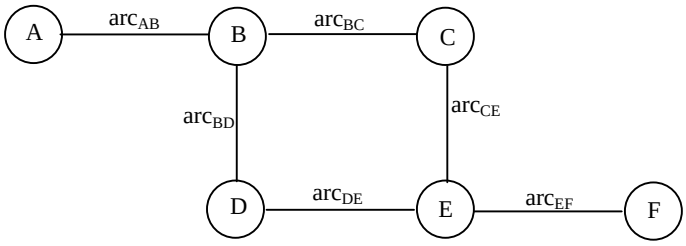

Figure 5: Topology of the Six-Node Example

As seen in Figure 5, there are alternative minimum-hop paths between B and E, which may create a loop. We assume that in the Six-Node Example, the packets from B to E take the path B-C-E, and the packets from E to B take the path E-D-B. Such traffic pattern will create a loop on the path between B and E. The traceroute from B to E will provide the following data:

a) B-C-B (will cover both directions of arc_{BC}, exactly same as $traceroute_{BC}$)

b) B-C-E-D-B (total delay data from one directional loop)

We can only use this traceroute delay data to predict the $traceroute_{BC}$ delays. It is impossible to make inferences about the round-trip delays experienced by $traceroute_{CE}$, $traceroute_{ED}$ or $traceroute_{DB}$, although $traceroute_{BE}$ travels on arc_{CE}, arc_{ED} and arc_{DB}. However, the second delay data (from the loop B-C-E-D-B) can be used to calculate the delay on any traceroute that will travel over the nodes B and E such as $traceroute_{AF}$, $traceroute_{AE}$ and $traceroute_{BF}$. The delay data of a loop cannot be used to calculate the traceroute values of single arcs, and traceroute delay data from single arcs cannot be used to calculate the one-directional loop delay. Therefore, in the final solution, there should be at least one path to cover each one-directional loop, in addition to the paths, which are selected to cover the arcs. The proposed formulation in Section 3 selects paths that will cover the arcs in the topology. To force the algorithm to select a path that contains the loop, an artificial arc should be added to the topology. The revised topology of the Six-Node Example can be seen in Figure 6.

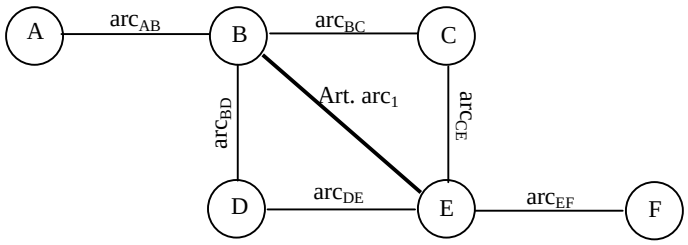

Figure 6: Revised topology of the Six-Node Example

Any example with loops can be solved as a topology without loops by adding one artificial arc for each loop in the topology. Notice that although $traceroute_{AF}$, $traceroute_{BF}$, $traceroute_{AE}$ and $traceroute_{BE}$ have loops in their traceroute paths, only one artificial link is created between B and E since all the loops are caused due to the loop between nodes B and E.

2.3. Assumptions

For the analysis in Sections 3 and 4, the following assumptions are made:

i) The topology is known

ii) No breakdowns occur on the links or the nodes. If a breakdown occurs, there is a chance that at least one of the paths of the selected S-D pairs can change. Any change on the paths of the selected subset may cause some arcs to be left uncovered (i.e. no probe traversing the arc). Subsequently, no delay information can be gathered about a path containing the uncovered arc. Such cases can be handled by solving the Constrained-Coverage Problem (explained in Section 3) whenever the status of the network changes (arcs going up or down). However, this issue will not be discussed any further in this study. For the sake of simplicity we will assume that there will be no breakdowns.

iii) The routes between any S-D pair are given, and every packet sent between an S-D pair takes the same path every time. This assumption is valid as long as there are no breakdowns in the network. The routes taken by the probes are based on a static routing table. The routes can be obtained in practice by sending traceroutes between all possible S-D pairs (in both directions) and recording the paths.

iv) Each arc is covered at least by one path. If all the paths for the network traffic are included in the analysis, then an arc that is not covered by any of the paths does not carry any traffic. In such cases, any arc without a traffic load can be discarded from the formulation.

v) Round-trip delays are the same for the paths between S-D pairs node A-node B and node B-node A. This assumption can be relaxed easily, by adding node B-node A as a new S-D pair and adding the path in-between as a new path (which would be the opposite of the path from node A to node B). However, this will cause significant increase in the number of variables and constraints.

3. THE GRAPH THEORY FORMULATION FOR THE CONSTRAINED COVERAGE PROBLEM

Given a topology and the paths between all S-D pairs, the Constrained Coverage Problem (explained below) selects a subset of S-D pairs with paths that will traverse all the arcs in the topology, while minimizing the size of the subset. The Constrained Coverage Problem has predefined paths hence differs from the classical coverage problem [7][8][9]. There is a unique path ($path_j$ for j=1… number of paths) for each S-D pair in topology. Therefore, minimizing the number of paths is equivalent to minimizing the size of the subset of the S-D pairs.

The Constrained Coverage Problem can be stated as a mathematical programming problem, more precisely a binary integer programming formulation as follows:

Notation:

N = number of nodes

M = number of arcs

n_paths = number of paths

= N*(N-1)/2

$path_j$ = path number j, j = 1..n_paths

$$= \begin{cases} 1 & \text{if } path_j \text{ is selected to be in the subset} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{i,j} = \begin{cases} 1 & \text{if } arc_i \in path_j \\ 0 & \text{otherwise} \end{cases}$$

$$\min \quad z = \sum_{j=1}^{n\_paths} path_j$$

s.t.

$$\sum_{j=1}^{n\_paths} a_{i,j} * path_j \geq 1 \qquad i = 1....M$$

$$path_j = 0 \text{ or } 1 \quad \forall \; j = 1....n\_paths$$

Therefore the objective is to select a subset of paths with minimum cardinality from a maximum of N (N-1)/2 paths subject to satisfying the constraint of having at least one path covering each arc. We illustrate the formulation on an example below

3.1. Example for the application of the integer programming formulation

The integer programming formulation is illustrated using the Three-Node Example in Section 2. There are three S-D pairs, which are 1-2, 1-3 and 2-3 (in other words there will be three paths, i.e. *n_paths* = 3). In Table I, the paths and the corresponding variables are given.

Table I: Integer programming variables for Three-node example

S-D	Path	Variables
1-2	arc_1	$path_1$
1-3	arc_1-arc_2	$path_2$
2-3	arc_2	$path_3$

The Integer Programming formulation can be stated as:

Min $z = path_1 + path_2 + path_3$

s.t.

$1*path_1 + 1*path_2 + 0*path_3 \geq 1$

$0*path_1 + 1*path_2 + 1*path_3 \geq 1$

$path_1, path_2, path_3 = 0$ or 1

The unique optimal solution to the problem is:

$z = 1,\ path_2 = 1,\ path_1 = path_3 = 0$

The result implies that only $path_2$ is chosen for probing. In other words, a probe will be sent only between nodes 1 and 3 and its path covers all the arcs in the topology.

4. IMPLEMENTATION OF THE CONSTRAINED COVERAGE PROBLEM

In this section, after discussing the complexity and implementation issues of the integer programming formulation, a polynomial time sub-optimal algorithm is proposed and compared with the integer programming formulation. Later, both algorithms for the Constrained Coverage Problem are compared on a real topology.

4.1. Complexity Discussion and an Alternative Sub-optimal Algorithm

There are at most N*(N-1)/2 variables and M constraints in the integer program of the Constrained Coverage Problem. Since all the variables are binary, there are at most $2^{N(N-1)/2}$ alternative solutions. So the complexity of the integer programming formulation is of the order of $O(2^{N^2})$. Since the Constrained Coverage Problem should be solved each time the routes changes due to breakdowns, and since the complexity of the integer programming algorithm is exponential in the number of nodes, there is a need for approximate solution via a polynomial-time algorithm. Hence we develop a sub-optimal polynomial time algorithm for the problem. Although the optimal solution is not guaranteed, the goal to determine a subset of S-D pairs to cover all the arcs in the topology is satisfied. A greedy heuristic is used at each step of the algorithm, an S-D pair that covers the greatest number of uncovered arcs is selected.

Let $A=[a_{ij}]$, where a_{ij} is defined in Section 3. The sum of row i of the matrix A gives us the information of how many paths cover arc i. Column sum S_j, depicts the number of arcs covered by $path_j$, j=1,…,n_paths. Choosing the path with the most covered arcs is a reasonable step to begin the algorithm. After selecting the first path, A should be updated, such that column sum S_j should give us the number of uncovered arcs that $path_j$ will cover if selected. S_j, j=1,…, n_paths, values for the updated A are used to select the next path to be included in the subset. The sub-optimal algorithm for the Constrained Coverage Problem can be summarized as follows:

Step 1: Set p=0, LIST =∅ and $A^0 = A$, where p is the counter for the number of columns chosen and LIST is the set of the selected paths.

Step 2: Calculate S_j of A^p, j=1,…,n_paths

$$S_j = \sum_{1=1}^{M} a_{ij}^{p}$$

Step 3: Choose the largest S_j, j=1,…,n_paths (ties are broken arbitrarily). Let S_k=max (S_j), j=1,…,n_paths.

If S_k=0, then terminate. The paths in the LIST will constitute the sub-optimal subset of the S-D pairs.

Else p=p+1, LIST =LIST U {$path_k$}.

Step 4: Modify the matrix with respect to column k of matrix A^{p-1} to form A^p as:

$a_{ij}^{p} = \max\{0, a_{ij,j\neq k}^{p-1} - a_{ik}^{p-1}\}$. Go to Step 2.

4.2. Example application of the sub-optimal algorithm

Following is the example used for the integer programming formulation, the sub-optimal algorithm is also demonstrated on the Three-Node Example in Section 2:

Step 1: p=0, LIST =∅ and A^0 =A as shown below

		S_D PAIR		
		1-2	1-3	2-3
ARCS	Arc_1	1	1	0
	Arc_2	0	1	1

Step 2: Calculate S_j, i=1,2,3

		S_D PAIR		
		1-2	1-3	2-3
ARCS	Arc_1	1	1	0
	Arc_2	0	1	1
S_j		1	2	1

Step 3: S_2=max (S_j), j=1,2,3. S_2>0, LIST={1-3}, go to Step 4

Step 4: Modify the columns to form A^1. The modifications are as follows:

		S_D PAIR		
		1-2	1-3	2-3
ARCS	Arc_1	Max{(1-1),0}	Max{(1-1),0}	Max{(0-1),0}
	Arc_2	Max{(0-1),0}	Max{(1-1),0}	Max{(0-1),0}

The new matrix A^1 is as follows:

	S_D PAIR		
	1-2	1-3	2-3

ARCS	Arc_1	0	0	0
	Arc_2	0	0	0

Goto Step2.

Step 2: Calculate S_j, i=1,2,3.

		S_D PAIR		
		1-2	1-3	2-3
ARCS	Arc_1	0	0	0
	Arc_2	0	0	0
S_j		0	0	0

Step 3: S_2=max(S_j)=0, j=1,2,3. Terminate.

The path between nodes 1 and 3 ($path_2$) is selected to cover the topology of the three-node example. The solution of the sub-optimal algorithm is the same as the solution of the optimal integer programming algorithm. In order to compare the sub-optimal and integer programming algorithms from computational point of view, first we have to calculate the computational complexity of the sub-optimal algorithm.

At each step, the proposed sub-optimal algorithm chooses one path that will cover the most number of uncovered arcs. Note that the selected path should cover at least one uncovered arc. If there are no such paths, then the algorithm will terminate. Due to assumption (iv) in Section 2.1., all the arcs will be covered when the algorithm terminates. In the worst case, only one arc is covered by the selected path at each step, meaning that the algorithm may have a maximum of M iterations. At each iteration, the matrix A_{M*n_paths} is updated. The dimensions of the matrix can be at most M*[N(N-1)/2]. The worst case computational effort needed for this algorithm is M*[M*N*(N-1)/2]. In other words, the complexity of the algorithm is in the order of O(M^2N^2).

In terms of complexity, the polynomial algorithm has advantages over the integer programming algorithm. Since there is no guarantee on the proximity of the solutions from the sub-optimal (polynomial) algorithm and the integer programming algorithm, we should test the solution performance of both methods on existing and random topologies.

4.3. Application to the vBNS Network

To compare the performance of the integer programming algorithm and the polynomial algorithm, we consider the vBNS (very high speed Backbone Network Service) (Figure 7) [10]. There are 12 nodes and 17 arcs in the network. If we want to observe all the round trip delays on all the S-D pairs in the topology, then we have to send probes on 66 paths. The route between an S-D pair was selected arbitrarily using the minimum-hop path. In some cases, alternative paths (with the same number of hops) were observed between some S-D pairs, which might cause loops as mentioned in Section 2. However, in this example we assumed no loops, since we provided a method to handle loops in Section 2.

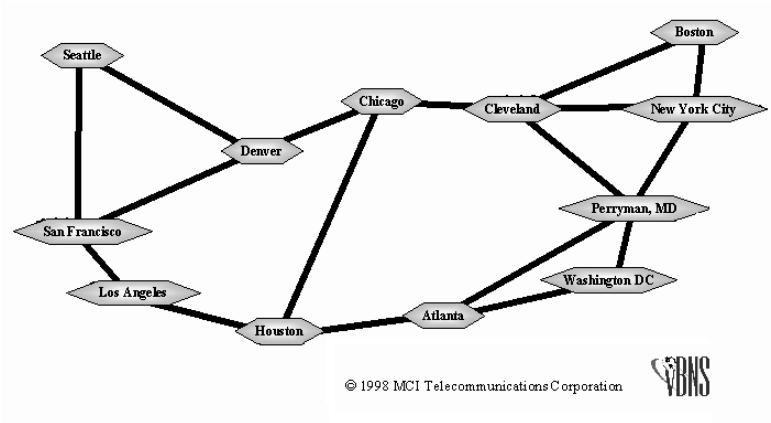

Figure 7: vBNS backbone (Source:[10])

After solving the problem using the integer programming formulation, we achieved an optimal solution of six paths (Boston-Atlanta, Boston-San Francisco, NY City-Cleveland, Cleveland-Washington DC, Washington DC-Seattle, Houston-Seattle; note that the paths for Washington DC-Seattle and Houston-Seattle have alternative shortest paths). By sending traceroutes between the six selected paths, we can calculate the necessary information for all 66 paths, thus obtaining a 90.91 percent reduction in the number of observed S-D pairs. The same problem was also solved using the polynomial algorithm, yielding a subset of six paths (Boston-Washington DC, Boston-San Francisco, NY City-Cleveland, Cleveland-Atlanta, Washington DC-Seattle, Houston-Seattle). However, the new subset has different paths compared to the subset obtained from the integer programming formulation due to multiple optimal solutions. The polynomial algorithm performed as efficiently as the integer programming formulation in this case. The effectiveness of the sub-optimal algorithm is satisfactory for the vBNS example, however we have to compare performances of integer programming formulation and

the sub-optimal algorithm using randomly generated topologies. The results of the performance testing of the two algorithms are summarized in the next section.

4.4. Application of the constrained coverage problem formulations to randomly generated topologies

The topologies of existing networks generally are not publicly available especially if they are commercially owned and operated. The unavailability of real topologies increases the importance of randomly generated topologies. In this study, we have selected the random-hop approach [11] for generating random topologies. The parameters for the random-hop approach are set so that the random topologies should resemble the publicly available backbone topologies NGNET 3 [12], ABILENE [13], vBNS (old) and vBNS (current) [10]. Thirty random topologies are generated and solved by integer programming formulation and the sub-optimal algorithm. The results are summarized in Table II.

Table II: Solutions of Integer Programming and sub-optimal algorithm for the 30 random topologies

Ex	*N*	*M*	# probes	# probes in the optimal solution	# probes in the solution of the sub-optimal algorithm	Ratio of the solution of sub-optimal algorithm to all probes	Difference from the optimal solution
1	7	8	21	4	4	19.05%	0
2	8	10	28	5	5	17.86%	0
3	10	12	45	5	5	11.11%	0
4	9	11	36	4	5	13.89%	1
5	9	10	36	4	4	11.11%	0
6	9	10	36	4	4	11.11%	0
7	7	8	21	4	4	19.05%	0
8	8	9	28	4	4	14.29%	0
9	10	11	45	4	4	8.89%	0

10	8	10	28	5	5	17.86%	0
11	11	14	55	5	5	9.09%	0
12	15	21	105	9	9	8.57%	0
13	9	9	36	4	4	11.11%	0
14	10	10	45	3	3	6.67%	0
15	14	18	91	5	6	6.59%	1
16	12	18	66	8	8	12.12%	0
17	8	8	28	3	3	10.71%	0
18	9	9	36	3	3	8.33%	0
19	13	17	78	5	5	7.69%	0
20	14	17	91	5	5	5.49%	0
21	13	18	78	8	8	10.26%	0
22	12	18	66	6	6	9.09%	0
23	12	19	66	8	9	13.64%	1
24	12	18	66	6	6	9.09%	0
25	12	18	66	6	7	12.12%	1
26	10	14	45	5	6	13.33%	1
27	9	14	36	8	8	22.22%	0
28	11	15	55	7	7	12.73%	0
29	12	17	66	6	7	10.61%	1
30	10	15	45	6	6	13.33%	0

The application of the sub-optimal algorithm on randomly generated topologies yield an average reduction of 88.1% in the number of monitored paths. In other words, the number of probes necessary to be observed is 11.9% of all probes. Six of the thirty examples (examples 4, 15, 23, 25, 26 and 29) have resulted in non-optimal solutions when the sub-optimal algorithm is applied. Although, the sub-optimal algorithm yields non-optimal solutions, the resulting subsets are second best solution in all examples. Whether the solution of the sub-optimal algorithm is optimal or near optimal, the sub-optimal algorithm performs extremely well in reducing the size of the monitored probes. In the examples in Table II, we observe that the success of the sub-optimal algorithm and the integer programming formulation increase as the size of the topology increases.

We have provided an example in Figure 8 to demonstrate a situation where the sub-optimal algorithm fails to obtain an optimal solution.

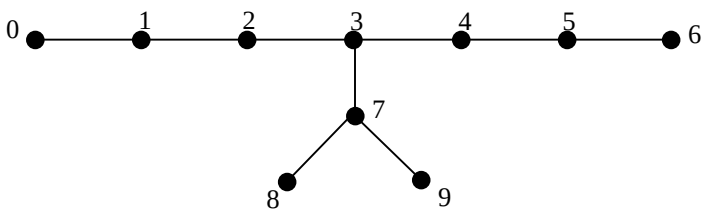

Figure 8: An example where the sub-optimal algorithm fails to yield the optimal probe subset

Path (0-6) is the only path with six arcs. Selecting path (0-6) in the first step of the sub-optimal algorithm will yield the solution path (0-6), path (0-8) and path (0-9). However, path (0-6) is not in any of the optimal solutions. In other words, once path (0-6) is selected, it is guaranteed that an optimal solution cannot be reached. For the example in Figure 8, the topology can be covered by two probes (such as path (0-8) and path (6-9)) and this solution does not have any paths with six arcs.

As in the example shown in Figure 8 and the other six cases in Table II (4, 15, 23, 25, 26 and 29), the sub-optimal algorithm yielded the next best results. Obtaining near optimal solutions is a reasonable trade-off for gaining a polynomial-time algorithm. The importance of a polynomial-time algorithm becomes more critical as the size of the problem increases (such an example is a multi-layer hierarchical network where there are thousands of nodes). In addition, the coverage problem is repeated each time there is a breakdown, which increases the importance of the complexity of the algorithm.

5. CONCLUSION

Given a network topology, the routes followed by packets between all S-D pairs, and the traceroute-like probes, we have formulated a graph-theoretic problem, namely the constrained coverage problem, of minimizing the number of source-destination probes such that every arc in the network is covered.

In this study, we presented an integer programming formulation for the graph-theoretic problem, which constitutes a new problem definition in graph theory. In addition, the complexity and the accuracy of the classical integer programming solution technique have been compared against a sub-optimal algorithm that we have developed. The comparison has yielded that the constrained coverage problem solutions of an existing topology (the vBNS backbone topology) have

resulted in a drastic (approximately 91 percent) reduction in the number of probes by both techniques. Additional thirty random topologies generated and solved using the integer programming formulation and the sub-optimal algorithm. In most cases the sub-optimal polynomial time algorithm has performed as well as the optimal exponential time integer-programming algorithm. On the remaining cases, the sub-optimal algorithm has yielded the second the best solution if not optimal. On average, the application of the sub-optimal algorithm on randomly generated topologies yields 88.1% reduction in the number of monitored paths, while achieving an optimal solution in twenty-four out of thirty examples.

Given the topology and routing information, and the subset of traceroute-like probes, the network can be continuously monitored and the delay across each arc in the network can be determined. Then the delay between any source and destination can be obtained by adding up the delays across the arcs in the route and possibly a safety factor to account for random network behavior. Also, the hot spots can be identified by periodically observing any anomalies in the delay across each arc.

ACKNOWLEDGMENTS

This work is partially supported by DARPA (Contract number F30602-97-C-0274). We also thank Lucent Technologies for their generous support.

REFERENCES

1. R. Govindan, and A. Reddy, An analysis of internet inter-domain topology and routing stability, An analysis of internet inter-domain topology and routing stability. *Proceedings of the 16th IEEE INFOCOM'97Annual Joint Conference of the IEEE Computer and Communications Societies*, 2, pp. 850-857, 1997.
2. V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM transactions on networking*, Vol. 5, No. 4, pp. 601-615, 1997.
3. V. Paxson, and S. Floyd, "Wide-area traffic: the failure of Poisson modeling". IEEE/ACM Transactions on Networking, Vol. 3, No. 3, pp. 226-244, 1995.
4. K. Thompson, G. J. Miller, and R. Wilder, "Wide-area Internet traffic patterns and characteristics," *IEEE Network Magazine*, Vol. 11, No. 6, pp.10-23, 1997.
5. N. Jamison, et al, "VBNS: not your father's Internet," *IEEE Spectrum*, Vol. 35, No. 7, pp. 38-46, 1998.
6. K. Varadhan, D. Estrin, and S. Floyd, Impact of network dynamics on end-to-end protocols: case studies in TCP and reliable multicast, *Third IEEE symposium on computers and communications, ISCC'98*, pp147-153, 1998.
7. R.K. Ahuja, T.L. Magnanti. and J.B. Orlin, *Network flows: Theory, Algorithms, and Applications*, Prentice Hall, New York, 1993.
8. G.L. Nemhauser, and L.A. Wolsey, *Integer and combinatorial optimization*, John Wiley & Sons, New York, 1988.
9. M.S. Bazara, J.J. Jarvis, and H. D. Sherali, *Linear programming and Network Flows, 2nd edition*, John Wiley & Sons, New York, 1990.
10. http://www.vbns.net/

11. H.C. Ozmutlu, N. Gautam and R. Barton (Forthcoming) Random-hop approach for network topology generation.
12. http://www.ngnet3.net/
13. http://www.ucaid.edu/abilene/

Huseyin C. Ozmutlu was born in Ankara, Turkey, in 1971. He received his B.S. degree in Management Engineering in June 1993 from Istanbul Technical University, Istanbul. He began working as a teaching and research assistant in Uludag University and earned a scholarship from Uludag University in 1994 for graduate study in the U.S. He received his M.S. degree in Operations Research at the George Washington University in 1996. He began his Ph.D. education at the Columbia University Department of Industrial Engineering and Operations Research in 1996, and continued at the Pennsylvania State University Department of Industrial and Manufacturing Engineering. He is expected to graduate from Pennsylvania State University and begin working as an Assistant Professor in Uludag University, Turkey, in August 2001. His research interests include efficient monitoring tools for Internet, telecommunication networks, Internet search engine user behavior analysis, mathematical programming and applied optimization.

Natarajan Gautam is an Assistant Professor in the Harold and Inge Marcus Department of Industrial and Manufacturing Engineering at the Pennsylvania State University. His research interests are in the areas of modeling, analysis and performance evaluation of computer and communication networks as well as information systems. He is a member of IEEE, INFORMS and MAA, and a senior member of IIE. He received both his Ph.D. and M.S. in Operations Research from the University of North Carolina at Chapel Hill.

Russell Barton is a Professor in the Harold and Inge Marcus Department of Industrial and Manufacturing Engineering at the Pennsylvania State University. He received a B.S. in Electrical Engineering from Princeton University and M.S. and Ph.D. degrees in Operations Research from Cornell University. At Penn State, he has worked to increase the practice component of engineering education. He is Secretary-Treasurer for the Informs College on Simulation, and he was Proceedings Co-editor for the 2000 Winter Simulation Conference. His current interests include graphical methods for experiment design, design of experiments and statistical models of product and process behavior.