Network Monitoring: Probe-Subset Selection Using the Constrained Coverage Problem[1]

Huseyin C. Ozmutlu[*a], Russell Barton[a], Natarajan Gautam[a], William J. Hery[b]

[a]Dept. of Industrial and Manufacturing Engineering, Pennsylvania State Univ, University Park, PA,16802
[b]Lucent Technologies, Bell Laboratories, Whippany, New Jersey 07981.

ABSTRACT

To predict the delay between a source and destination as well as to identify anomalies in a network, it is crucial 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 the 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.

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

1. INTRODUCTION

The tremendous growth of the Internet users and applications has resulted in severe congestion of the network[1,2]. This has led to the users demanding a Quality-of-Service (QoS) for their network applications. In particular, users are interested in the latency or delay and also in specifying a suitable route (or path; we will use path and route interchangeably) by identifying congestion bottlenecks, 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 delay in sending a message from a source to a destination, and, where are the "hot-spots" or anomalies in the network.

A way to answer the two questions is by continuously monitoring the network and making predictions about the end-to-end delay and the location of the "hot-spots"[3]. This monitoring requires sending probes between all sources and destinations in the network, which could potentially result in heavy traffic by just using the probes. A crucial decision is to choose an optimal subset of probes that is both small enough to handle (small files to store and browse through quickly for monitoring) and not contribute to the congestion as well as large enough to make accurate predictions[4].

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 "traceroute-like" 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.

This graph-theoretic problem that we call the "constrained coverage problem" is an important contribution to graph theory literature. We also 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 formulate it as a mathematical programming problem. In Section 4 we illustrate the algorithms and present examples. In Section 5 we state the conclusions of this work.

[1] To appear in the proceedings of the SPIE international symposium on Voice, Video and Data Communications (Performance and Control of Network Systems III), September 1999.
[*] Correspondence: Email: hco100@psu.edu; Telephone: 814 865 7601; Fax: 814 863 4745

2. PROBLEM DEFINITION AND INPUT ANALYSIS

The objective of this study is to select the optimal or sub-optimal subset of source and destination (S-D) pairs for continuous probing. The delay data obtained from the 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 of which provides different levels of details for the delay information. For example, ping[**] provides round-trip end-to-end delay, but no information about delays 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. Therefore, in this study, we use traceroute-like intelligent agents as the probe type. For sake of simplicity, we refer to such probes as, simply, traceroutes.

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:

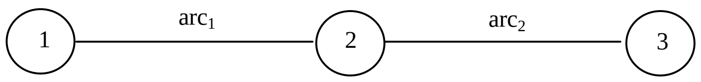

Figure 1
Topology of the Three-Node Example

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 the 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 reasons, we assumed that round-trip delays are same on the opposite directions of each path for the remaining part of this study.

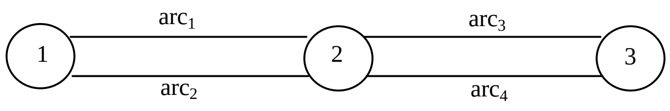

Figure 2
Topology of the Three-node example with one-directional arcs

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

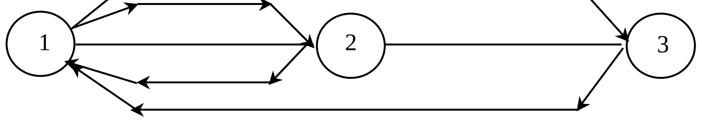

Figure 3
Delay data from traceroute between 1 and 3

[**] Unix version of ping (ping -s "destination")

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).

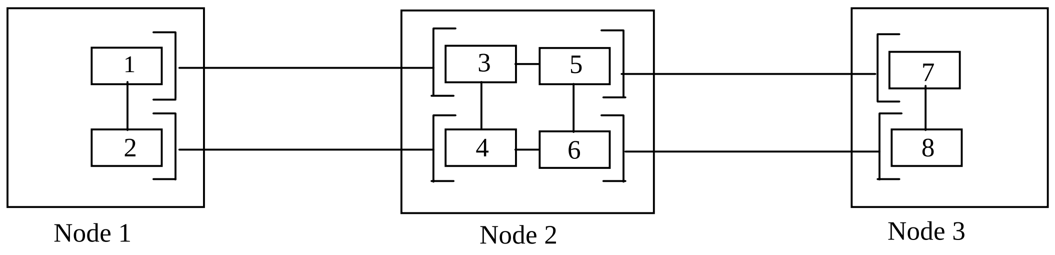

Figure 4
Buffers in the topology of the Three-Node Example

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$_{13}$. 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 buffers 1 and 3 at the same time, so delay due to 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 a very close estimate of the delay on traceroute$_{23}$.

$$\text{traceroute}_{23} = \text{traceroute}_{13} - \text{traceroute}_{12} \pm \varepsilon \qquad (1)$$

By sending enough traceroute probes that will 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 in the topology. Consequently, delay on any given path can be easily calculated.

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.

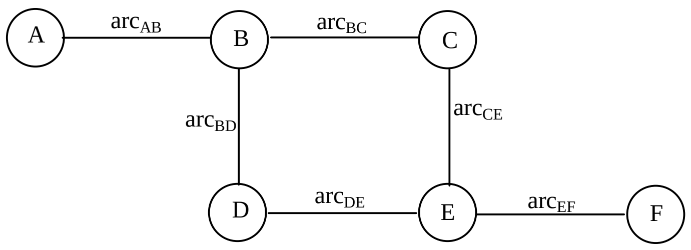

Figure 5
Topology of the Six-Node Example

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 E and B. 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.

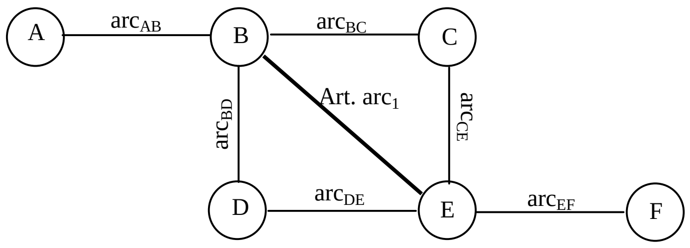

Figure 6
Revised topology of the Six-Node Example

After adding one artificial arc for each loop in the topology, any example with loops can be solved as a topology without loops. 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.

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. 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) repeatedly 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 between $Node_A$ 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[5,6]. 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 $\qquad j = 1..n\_paths$,

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

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

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

$s.t.$

$$\sum_{j=1}^{n\_paths} a_{i,j} * path_j \geq 1 \qquad i = 1....M \tag{2}$$

$$path_j = \quad 0\ \ or\ \ 1 \qquad \forall\ j = 1....n\_paths$$

Example

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). In Table 1, the paths and the corresponding variables are given.

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

Table 1
Integer Programming variables for Three-node example

The Integer Programming formulation can be stated as:

Min $\quad z = path_1 + path_2 + path_3$

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 to send a probe. In other words, a probe will be sent only between nodes 1-3 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 complexity of the integer programming formulation is of the order of $O(2^{n^2})$. The Constrained Coverage Problem should be solved each time the routes changes due to breakdowns. Since the complexity of the integer programming algorithm is in exponential time, there is a need for a polynomial-time algorithm. We hence 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. At each step of the algorithm, an S-D pair that will cover most of the 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 the arc i. The column sum S_j, depicts the number of arcs covered by path_j. Choosing the path with the most covered arcs is a reasonable step to begin the algorithm. After selecting the first path, the matrix A should be updated, such that the column sum S_j should give us the number of uncovered arcs that path_j will cover if selected. The S_j values for the updated matrix A is 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 $=\emptyset$ and $A^0=A$, where p is the counter for the number of columns chosen and the LIST is the set of the selected paths.

Step 2: Calculate the column sum S_j of A^p .

$$S_i = \sum_{j=1}^{a} a_{ij}^{p}$$

Step 3: Choose the largest nonzero S_j, $j=1,\dots,n\_paths$ (ties are broken arbitrarily). Let $S_k=\max(S_j)$, $j=1,\dots,n\_paths$.

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 $\{\text{path}_k\}$.

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

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

Example:

The sub-optimal algorithm is demonstrated by applying it to the Three-Node Example:

		S_D PAIR		
		1-2	1-3	2-3
A R	Arc1	1	1	0
C S	Arc2	0	1	1

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

	S_D PAIR		
	1-2	1-3	2-3

ARCS	Arc1	1	1	0
	Arc2	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 remaining columns to form matrix A^1.The modifications are as follows:

		S_D PAIR		
		1-2	1-3	2-3
ARCS	Arc1	Max{(1-1),0}	Max{(1-1),0}	Max{(0-1),0}
	Arc2	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	Arc1	0	0	0
	Arc2	0	0	0

Goto Step2.

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

		S_D PAIR		
		1-2	1-3	2-3
ARCS	Arc1	0	0	0
	Arc2	0	0	0
S_j		0	0	0

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

Path 1-3 is selected to cover the topology of the three-node example. The solution of the sub-optimal algorithm is same as the solution of the optimal integer programming algorithm. In order to compare, sub-optimal and integer programming algorithms, first we have to calculate the computational complexity of the sub-optimal algorithm.

At each step, the proposed 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 should 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 both methods on an Internet topology.

4.2. Application of the constrained coverage problem formulations to the vBNS Network

To compare the performance of the integer programming algorithm and the polynomial algorithm, we consider the vBNS (very high Backbone Network Service) (Figure 7). 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.

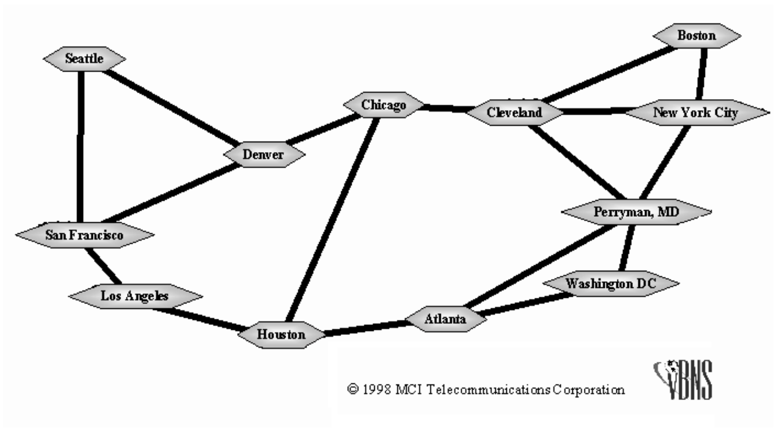

Figure 7
vBNS backbone[10]

After solving the problem with the IP 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.

5. CONCLUSION

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

We state an integer programming problem for the graph-theoretic formulation. We compare the complexity and the accuracy of the classical integer programming solution techniques and a sub-optimal algorithm we developed. The example we considered resulted in a drastic (approximately 91 percent) reduction in the number of probes by both techniques. In most cases the sub-optimal polynomial time algorithm performs as well as the optimal exponential time integer programming algorithm.

Now using the final subset of probes, the network can be continuously monitored and the delay across each arc in the network can be stored. 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 observing any anomalies in the delay across each arc.

ACKNOWLEDGMENTS

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

REFERENCES

1. R. Govindan, and A. Reddy, "An analysis of internet inter-domain topology and routing stability," *IEEE Infocom*, pp. 7b.4.1-7b.4.8., 1997.
2. V. Paxson, "End-to-end routing behavior in the Internet," *Proceedings of SIGCOMM '96*, pp. 25-38, 1996.
3. K. Thompson, G. J. Miller, and R. Wilder, "Wide-area Internet traffic patterns and characteristics," *IEEE Network Magazine*, November/December 1997, pp.10-23, 1997.
4. K. Varadhan, D. Estrin, and S. Floyd (1997). "Impact of network dynamics on end-to-end protocols: case studies in TCP and reliable multicast," Submitted to *IEEE Infocom*, 1998.
5. R.K. Ahuja, T.L. Magnanti. and J.B. Orlin, *Network flows: theory, algorithms, and applications*, Prentice Hall, 1993.
6. G.L. Nemhauser, and L.A. Wolsey, *Integer and combinatorial optimization*, John Wiley & Sons, New York, 1998.
7. M.S. Bazara, J.J. Jarvis, and H. D. Sherali, *Linear programming and network flows, second edition*, John Wiley & Sons, New York, 1990.
8. N. Jamison, et al, "VBNS: not your father's Internet," *IEEE Spectrum*, July 1998, pp. 38-46, 1998
9. V. Paxson, and S. Floyd, "Wide-area traffic: the failure of Poisson modeling". IEEE/ACM Transactions on Networking 3, pp. 226-244. 1995.
10. http://www.vbns.net/