Distributed Network Monitoring with Bounded Link Utilization in IP Networks

Li Li
Center for Networking Research
Lucent Bell Labs
erranlli@dnrc.bell-labs.com

Marina Thottan
Center for Networking Research
Lucent Bell Labs
marinat@dnrc.bell-labs.com

Bin Yao
Center for Networking Research
Lucent Bell Labs
byao@dnrc.bell-labs.com

Sanjoy Paul
Center for Networking Research
Lucent Bell Labs
sanjoy@dnrc.bell-labs.com

Abstract— **Designing optimal measurement infrastructure is a key step for network management. In this work we address the problem of optimizing a scalable distributed polling system. The goal of the optimization is to reduce the cost of deployment of the measurement infrastructure by identifying a minimum poller set subject to bandwidth constraints on the individual links. We show that this problem is NP*-hard* and propose three different heuristics to obtain a solution. We evaluate our heuristics on both hierarchical and flat topologies with different network sizes under different polling bandwidth constraints. We find that the heuristic of choosing the poller that can poll the maximum number of unpolled nodes is the best approach. Our simulation studies show that the results obtained by our best heuristic is close to the lower bound obtained using LP relaxation.**

I. Introduction

With the deployment of QoS (Quality of Service) capable networks, research in network measurement infrastructure is gaining significance [9][2][12][10]. Accurate network measurement is essential for understanding network behavior and for providing Quality of Service (QoS) guarantees [5]. Most commercial network management software use the Simple Network Management Protocol (SNMP) [16] as the primary method for data collection. Using SNMP involves running SNMP agents on network devices and an SNMP manager polls these devices for management information.

SNMP-based polling systems have an inherent overhead in terms of processing load on network nodes and network bandwidth consumption. This overhead is further exaggerated when network devices are polled at a high frequency [18]. Such frequent polling is vital for providing QoS guarantees and fast failure detection. There is a considerable amount of work being done to improve the performance of the SNMP protocol by reducing the CPU load on the network node. For example, work has been done both to improve SNMP primitives [4] as well as to design better polling strategies by batching SNMP requests [6]. On the other hand, not much research has been done to reduce network bandwidth consumption. Bandwidth is a revenue generating resource and therefore service providers are reluctant to allocate large amounts of valuable bandwidth for network management purposes. Thus bandwidth constraint for network monitoring is an essential design criteria for any measurement infrastructure.

In a centralized measurement system (all network nodes are monitored from a central manager), poll responses have to be forwarded to a central location in the network. This provides a network-wide view but creates a potential for bandwidth bottleneck on links that are close to the central manager. On the other hand in a distributed measurement system [21], the polling load is shared among multiple pollers located at different points in the network. However, using distributed pollers could increase the cost of network management in terms of the number of pollers deployed and suffers from the lack of a network-wide view.

Taking into account the issues of scalability and network-wide view for large service provider networks, we envision an ideal measurement architecture to be a hierarchical system. A hierarchical system implies that there is a central manager but the resource intensive tasks such as polling are distributed among a set of polling nodes. Between the central manager and the polling nodes, there exists a set of aggregating nodes. The pollers are distributed and each poller is responsible for a polling domain consisting of a subset of the network nodes. Information gathered from the individual polling nodes are then aggregated [17] at the aggregators. The condensed information is then sent to the central manager that provides an overall view of network behavior. Such a hierarchical architecture reduces bandwidth overhead while still maintaining a network-wide view.

In the hierarchical polling-based measurement infrastructure, the bandwidth overhead is mainly composed of polling traffic to the network nodes. The amount traffic to the aggregator from the polling nodes, and to the central manager from the aggregator, is expected to be significantly smaller. Therefore, by just distributing the polling traffic the overall impact on network bandwidth can be significantly minimized. However, using a large number of distributed pollers will increase the cost of deployment and increase the complexity of the aggregation process. In this work, we identify a small subset of the nodes as the distributed poller locations that are required for a given network topology, with a known polling load, under a fixed per link bandwidth constraint. Our problem formulation includes the bandwidth constraint explicitly since it is extremely critical that the measurement system does not create a potential for bottleneck links in the network.

Our work focuses on optimizing a measurement system for a service provider network that supports QoS. The main contributions of our work are as follows: We first show that the problem of minimizing the number of pollers in a given network subject to bandwidth constraints is *NP hard*. We provide solutions to this problem by using heuristics based on the polling load and the maximum assignment of pollees. We show that the results obtained using the heuristic of maximum pollee assignment with re-shuffling is the most scalable in terms of both the number of pollers as well as the total bandwidth consumed by the measurement traffic. Based on empirical studies we show that our results are close to the lower bound obtained using *Linear Programming (LP) relaxation* by a factor of 4.5 in terms of the number of pollers and by a factor of 2 in terms of bandwidth consumed. We also provide detailed simulation studies for service provider networks with a hierarchical topology as well as for enterprise networks. The impact of QoS support on measurement systems is accounted for by including the bandwidth required for Multi-Protocol Label Switched (MPLS) tunnel monitoring [1].

The paper is organized as follows. In Section(II) we provide the integer programming formulation for finding the minimal set of pollers in a given network, as well as the assignments of individual nodes to the poller set, subject to bandwidth constraints. We show that this problem is NP-hard. In Section(III) we propose several heuristics that can provide reasonably good solutions to this problem and these solutions are evaluated based on the number of pollers chosen and the total bandwidth overhead incurred. The results obtained are also compared with the *LP relaxations*. Section(IV) provides simulation studies on both enterprise and service provider environments. We also evaluate our heuristics under different capacity constraints on the polling bandwidth.

II. System Model and Problem Formulation

Our model for the service provider network is an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the set of nodes or routers that must be polled and $e = (v_i, v_j)$ represents the edge that connects the nodes v_i and v_j and $e \in E$, where E represents the set of edges. Our model assumes that the graph edges are un-directed since we assume that the SNMP poll requests and poll responses are approximately equal in size (number of bytes) [2]. The graph model spans the entire service provider network from the aggregating nodes at the access points to the core nodes. Therefore $n = |V|$ is the total number of routers in the service provider network (including the access routers) and $m = |E|$ is the total number of links in the network but does not include the incoming links at the access routers.

In this work we assume that SNMP managers (pollers) can be co-located with network nodes. Each node v_i generates a polling traffic of w_i *bps*. This polling traffic is destined to the poller that has been assigned to this node. The polling load from each node is a function of the number of physical links as well as logical links (i.e. MPLS tunnels). Let $l(e)$ and $b(e)$ represent the actual polling bandwidth used and the amount of link bandwidth allocated for polling traffic for each of the edges. Bandwidth constraint is typically a fraction of the total capacity $c(e)$ on the link.

The optimal poller location and pollee assignment problem minPL can therefore be stated as follows: Given a network $G = (V, E)$, determine (1) a *minimum subset* of nodes $S \subseteq V$ on which to place pollers such that the the bandwidth constraint on each and every link $l(e) \leq b(e)$ is satisfied where $b(e)$, is the maximum bandwidth that can be used for polling on link e. (2) a mapping λ which maps a pollee to its poller. That is, for each node v_i, if $\lambda(v_i) = v_j$, then node v_i is assigned to the poller placed on node v_j. Note that, if $\lambda(v_i) = v_i$, then node v_i is being polled by itself.

In this paper, we assume that the routes between the poller and the pollee are fixed and symmetric. This assumption is reasonable since in a typical service provider network the routing paths are fairly stable. We assume that these routing paths are known and therefore considered as input to our poller placement problem.

A. Integer Programming Formulation for minPL

Given our assumption that routes are fixed between any given poller-pollee pair, the problem formulation presented above can be casted into an integer programming formulation.

Let $n = |V|$ be the total number of nodes in the network; The binary variable x_{ij}, indicates whether node v_i is polled by node v_j, where $v_i, v_j \in V$. The binary variable δ_e^{ij} indicates whether edge e belongs to the path P_{ij} between node v_i and v_j. Let w_i represent the polling bandwidth required to poll node v_i and b_e corresponds to the bandwidth constraint on the edge (physical link) e.

Our objective is:

$$Minimize \sum_{j=1}^{n} y_j \tag{1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \text{ for each } v_i \in V \tag{2}$$

$$x_{ij} \leq y_j, \text{ for each } v_i, v_j \in V \tag{3}$$

$$\sum_{i}^{n} \sum_{j}^{n} \delta_e^{ij} w_i x_{ij} \leq b(e) \text{ for each } e \in E \tag{4}$$

$$x_{ij} \in \{0, 1\}, \text{ for each } v_i, v_j \in V \tag{5}$$

$$y_j \in \{0, 1\}, \text{ for each } v_j \in V \tag{6}$$

The first constraint makes sure that each node v_i is assigned to exactly one poller. The second constraint guarantees that a node v_j must be a poller if some other node v_i is assigned to it. The third constraint ensures that the sum of the polling bandwidth used by all the poller pollee pairs on each link does not exceed its allocation.

B. Computational Complexity of the minPL Problem

Theorem 2.1: Computation of the optimal solution to the *minPL* problem is NP-hard.

Proof: The *minPL* problem is proven to be NP-hard via a reduction from the multi-processor scheduling problem, which

[1] Tunnel monitoring is done using MPLS traffic engineering MIB [15]

[2] Here we do not expect to poll for large tables since in this case there is a significant increase in the response size relative to the query size

is stated as follows. Given the set of tasks T, the number of available processors m, length $L(t)$ for each task $t \in T$ and a deadline $d < \sum_{t \in T} L(t)$, find a m-processor schedule that meets the overall deadline.

Given an instance of the m-processor scheduling problem, we create an instance of *minPL* problem. For each task $t \in T$, create a pollee node that has a polling demand of $L(t)$. We use Q to represent these nodes. Now pair-wise connect nodes in Q with links that have a polling bandwidth allocation of $\sum_{t \in T} L(t)$. Such a construction results in a clique of size $|T|$. For each processor in the scheduling problem, create a new node whose bandwidth demand is $\sum_{t \in T} L(t)$, and connect this node to an arbitrary node in Q with a new link that has d amount of bandwidth allocated for polling traffic. This step creates m additional nodes represented by P, and m additional links. Fig. 1 illustrate the $minPL$ instance constructed from a scheduling problem that has four tasks, two processors, and deadline d.

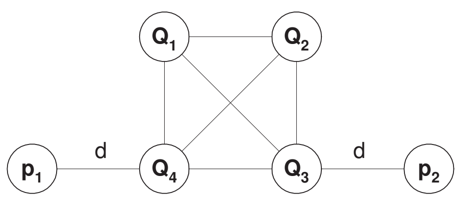


Fig. 1. Reduction from scheduling problem

Clearly, any node in P can only be polled by itself since its only link has bandwidth constraint d, which is less than its own demand, $\sum_{t \in T} L(t)$. Therefore, the solution to this *minPL* instance has at least m pollers. Consequently, if the *minPL* problem has exactly m pollers, all nodes in P and only nodes in P are pollers, with each poller handling no more than d amount of traffic from its pollees in Q. Since every node in Q is polled by a node in P, the poller-pollee relation between nodes in P and Q gives a m-processor schedule that satisfies the deadline d. On the other hand, suppose that the scheduling problem has a m-processor schedule, then a m poller solution can be constructed as follows: for a node $q \in Q$, find the processor on which the task represented by q is scheduled, let q be polled by the node $p \in P$ that represents that processor. This completes the proof. ■

III. Heuristic Algorithms for minPL

It is well-known that the Integer Programming formulation *minPL* described in Section(II) has an exponential running time in the worst case. In this section, we propose two sound heuristic algorithms. We also give a random poller placement scheme which serves as a base line for the evaluation of the two main heuristics employed. In addition, we use the Linear Programming (LP) relaxation as a crude lower bound to compare with our greedy heuristics.

Table I illustrates our greedy algorithm which consists of three steps. In the first step our algorithm greedily picks an additional poller (based on the greedy selection criteria described in Section III-A) if there are any nodes still present in the network that does not have a poller assigned to it. In the second step, after a poller is picked, the algorithm tries to assign the maximum number of un-polled nodes to the poller without violating bandwidth constraints (we refer to this sub-problem as the *maxPollee* problem)[3]. In the third step, the algorithm tries to reassign the pollees of other pollers to the new poller, if the overall bandwidth usage can be reduced when compared to the previous assignment. Care is taken to ensure that bandwidth constraints are not violated. This procedure is referred to as *ShuffleAndReduce*. The details of these steps are outlined in the next three subsections.

TABLE I

Heuristic Greedy Algorithm for *minPL*

```
Algorithm GreedyPollerPlacement(G,w,b)
Input: G = (V,E) is the undirected network.
       w is the vector of polling load for
       each node.
       b is the vector of bandwidth constraints
       on polling traffic for each edge.
       T_u is the routing subgraph used by u to
       reach all v ∈ V.
Output: S ⊆ V a set of pollers and λ the
       poller-pollee mapping.

1. set A = V and S = ∅;
2. while A ≠ ∅ do
3.     u = GetNextPoller(G,λ,w,b);
4.     S = S ∪ {u};
5.     B = MaxPollee(G,T_u,λ,u,w,b);
6.     A = A − B − {u};
7.     ShuffleAndReduce(G,λ,u,w,b);
```

A. Heuristics for the Greedy Selection of Pollers: GetNextPoller

Our proposed greedy heuristics tackle the problem of picking the next poller, i.e. procedure $GetNextPoller$ in Table I. We present 3 different heuristics to solve this problem.

Poller Selection Based on Polling Load: *HmaxLoad* Our first heuristic is to choose additional pollers [4] based on polling load. We refer to this heuristic as *HmaxLoad*. It sorts the polling demand w_i of each node $v_i \in V - S$ in decreasing order. Then it picks a node with the highest polling demand. This greedy heuristic is based on the intuition that, by choosing pollers based on their own polling load we can minimize the impact on the bandwidth constraints on the individual links of the network. The difference in polling load for the different nodes in the network is due to the degree of the node as well as the nature of the information being polled. For example in the case of an MPLS network, if the information pertaining to the MPLS tunnels are only polled at the source and destination nodes then this will cause the edge nodes to have a significantly higher polling bandwidth than the core nodes.

Poller Selection Based on maximum number of Pollees: *HmaxPollee* Our second heuristic is to choose an additional poller u from the set $V - S$ such that u can poll the maximum

[3]Note that step 5 is not needed for our HmaxPollee heuristic discussed in Section III-A

[4]The first poller is also chosen using the polling load criteria, (i.e.) in the algorithm we start with $S = \emptyset$

number of additional pollers without violating bandwidth constraints. We refer to this heuristic as *HmaxPollee*. The intuition is that, if we assign as many pollees as possible to each poller, we can minimize the number of pollers required. Note that this heuristic requires first solving the *MaxPollee* problem which we discuss in Section(III-B). The *MaxPollee* problem finds the maximum number of un-polled nodes that can be assigned to a given poller without violating the imposed bandwidth constraints.

Random Picking of Pollers: *HRandom* The third heuristic is to pick a poller from $V - S$ randomly. This heuristic serves as a base-line comparison for the other greedy heuristics proposed earlier.

B. Computing the Set of Pollees for a Poller: MaxPollee

Once we select a poller, we need to determine the set of pollees to be assigned to that poller. Ideally, we would like to assign maximum number of un-polled nodes to a new poller. Here an un-polled node refers to those nodes that are not yet assigned to a poller. This problem is referred to as the *MaxPollee* problem: Given a poller u, we would like to find the maximum set of pollees that can be polled by u without violating the bandwidth constraint.

Let $G_u = (V, E_u)$ be the graph where E_u includes edges on the route from v to u, for each $v \in V - \{u\}$. The general problem where G_u is not a tree is NP-hard (see Appendix for the reduction from the Maximum Independent Set problem). Fortunately the problem can be solved optimally if G_u is a tree. Therefore we assume that the network graph forms a tree. This assumption is reasonable since the shortest path routes in a network form a tree. Table II shows the algorithm to solve this *RestrictedMaxPollee* problem. The algorithm first sorts

TABLE II

OPTIMAL ALGORITHM FOR RESTRICTED MAXPOLLEE PROBLEM

```
Algorithm RestrictedMaxPollee(G, T_u, λ, u, w, b)
Input: G = (V, E) is the undirected network.
       u is the poller.
       T_u is the tree containing the unique
       path from each un-polled node.
       v ∈ V − {s} to s.
Output: J ⊆ V the maximum set of pollees
       that can be polled by u.
1. set J = ∅
2. sort un-polled nodes in increasing demand
   w and their level in the tree
3. consider each node v in increasing rank
4.     if (canAssign(T_u, u, v, w, b))
5.         J = J ∪ {v}
6. return J
```

the nodes in increasing order based on demand and their level in the tree, i.e. if demands are the same for any two nodes, then the node with smaller level appears first in the ordering scheme. If the level is also the same, then node with the smaller identifier appears first. Then each node is considered in increasing rank of the ordering scheme. If a pollee v can be assigned to poller s without violating the bandwidth constraint on each link, then v is included in the solution set J.

Theorem 3.1: The *RestrictedMaxPollee* problem can be solved optimally in $O(n^2)$ time.

Proof: Assume that nodes are ordered in increasing demand and level. Ties are broken by node identifier. $\pi(v_i)$ gives the rank of node v_i in this ordering. Let the poller set given by $RestrictedMaxPollee$ be J. Let us order the nodes in J by their rank as follows: v'_1,v'_2, $\cdots$, v'_k where $k = |J|$. Denote this ordering as O_J. We consider the optimal solution which consumes minimum bandwidth. If there are more than one such optimal solutions, we consider the one with minimum total rank. If there are still ties, we consider the optimal solution which shares the largest prefix with O_J. Lets denote the optimal solution chosen in this process J^*. Let the ordering O_{J^*} be v^*_1,v^*_2, $\cdots$, v^*_t where $t = |O_{J^*}|$. Assume that the pollees are picked one at a time in this ordering. Suppose they start differing at the i-th node in O_J, (i.e.) $v^*_j = v'_j$, $\forall j < i$. It must be the case that $\pi(v'_i) < \pi(v^*_i)$ otherwise $RestrictedMaxPollee$ must have picked v^*_i instead of v'_i. Since v'_i does not appear in J^*, there must exist at least one bottleneck link from v'_i to the poller s. Lets consider the bottle neck link e that is the first to became bottleneck for v'_i and it is happened when the optimal algorithm assigns v^*_f to the poller. It must be the case that $\pi(v^*_f) > \pi(v^*_i)$ since the optimal algorithm pick pollees in their increasing rank. Since e is a bottleneck, no node v^*_j such that $\pi(v^*_j) > \pi(v^*_f)$ belongs to the subtree rooted at the endpoint of e that is further away from the root s (note that $w(v^*_j) \geq w(v^*_f)$). Replacing v^*_f with v'_i in O_{J^*}, we obtain an optimal solution whose total rank is less than J^*. Again this contradicts our assumption.

The sorting takes $O(nlogn)$. Each node is considered once in the iteration and the function $canAssign$ needs only to check whether there is bandwidth available from v to s. Therefore, the running time is $O(hn)$ for this step where h is the height of the tree. Therefore, the total running time is $O(n^2)$. ∎

C. Reducing Bandwidth Consumption: ShuffleAndReduce

TABLE III

PROCEDURE FOR SHUFFLEANDREDUCE

```
procedure ShuffleAndReduce(G, S, λ, u, w, b)
1. sort all the pollees in set V − S in
   decreasing demand.
2. for each v ∈ V − S
3.    if (ReduceBandWidth(G, v, u, w, b))
4.       λ(v) = u;
5. for each s ∈ S
6.    if (NumPollees(s)==0 and
         CanAssign(T_u, u, s, w, b))
7.       λ(s) = u;
8.       S = S − s;
```

The algorithm tries to shuffle the pollees that have been assigned to previously chosen pollers, provided that reassigning a pollee v to a poller u reduces the bandwidth usage and does not violate any bandwidth constraints. This step is illustrated in Line 1-4 of Table-III and we refer to this step as *PolleeShuffle*. This step is necessary to reduce the overall bandwidth consumption since pollees may have been assigned to previously chosen pollers that are further away than the current poller. To overcome the inefficiencies in bandwidth usage it is necessary to have the pollee shuffle mechanism. Note that, one might augment *PolleeShuffle* with additional mechanisms to reduce the number of pollers in the process. For example, if all the pollees of a given poller s have been reassigned, then one can try to reassign the poller s to the current poller u and remove s if the reassignment complies with bandwidth constraints. One can also do *ShuffleAndReduce* separately for each existing pollers s and its pollees. This increases the chance that a poller gets removed during the *ShuffleAndReduce* process. There are many other variations that can be used for shuffling, we tested the afore-mentioned augmentation in lines 5-8 in Table-III.

D. Computing Lower Bound from LP Relaxations

Our primary goal is to minimize the set of pollers for a given network topology. We would like to know how the solution computed by our heuristic algorithms differ from the optimal. Since we cannot compute the optimal solution to the Integer Programming formulation of the *minPL*, we would like to compare our solution with some lower bound. We choose to use the solution from the LP relaxation of the integer programming formulation in Section II-A as the lower bound. Specifically, the LP relaxation gives the lower bound on the number of pollers required for a given topology.

To obtain the lower bound on the bandwidth consumption we make the assumption that the number of pollers is given. We would like to compare the total bandwidth consumed by our solution to the minimum possible bandwidth consumption assuming a fixed number of pollers. This problem (referred to as the *OptimalPolleeAssignment* problem) is also NP-hard via a reduction from the *BinPacking* problem(see Appendix for proof). The lower bound on bandwidth consumption in this case is obtained by solving the LP relaxation for the following integer programming problem:

$$Minimize \sum_{e \in E} \sum_{i}^{n} \sum_{j}^{n} \delta_e^{ij} w_i x_{ij} \tag{7}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \text{ for each } v_i \in V \tag{8}$$

$$x_{ij} \leq y_j, \text{ for each } v_i, v_j \in V \tag{9}$$

$$\sum_{i}^{n} \sum_{j}^{n} \delta_e^{ij} w_i x_{ij} \leq b(e) \text{ for each } e \in E \tag{10}$$

$$\sum_{j=1}^{n} y_j \leq k \tag{11}$$

$$x_{ij} \in \{0,1\}, \text{ for each } v_i, v_j \in V \tag{12}$$

$$y_j \in \{0,1\}, \text{ for each } v_j \in V \tag{13}$$

where k is the number of pollers given by the heuristic algorithm we choose to compare.

E. Time Complexity of the proposed algorithms

The procedure $GetNextPoller$ is implemented differently for different heuristic algorithms. For the *HmaxLoad* algorithm, we sort the nodes in decreasing load once. Therefore this step takes $O(nlogn)$. For the *HmaxPollee* algorithm, the procedure $GetNextPoller$ needs to find the node which can poll maximum additional unassigned nodes. This step requires invoking the *maxPollee* algorithm n times. Therefore the order is $O(n^3)$ since the *maxPollee* algorithm has an order of n^2. For the *HRandom* algorithm, it takes O(1) to randomly pick a node.

The procedure $MaxPollee$ takes $O(n^2)$ for the pollee assignment since it makes use of the *RestrictedMaxPollee* algorithm. The algorithm goes through k iterations where k is the number of pollers chosen. The $RestrictedMaxPollee$ is used by the *HMaxLoad* and *HRandom* heuristics. The *HMaxPollee* heuristic does not require another *RestrictedMax-Pollee* algorithm since the choice of the poller was already determined using the *RestrictedMaxPollee* algorithm.

The *ShuffleAndReduce* step has a complexity of $O(nlogn)$ with k iterations. Taking into account all three steps, the total running time for our proposed algorithm under each of the heuristic schemes are as follows: $O(kn^2)$ for *HmaxLoad*, $O(kn^3)$ for *HmaxPollee* and $O(kn^2)$ for *HRandom*. The *HmaxPollee* heuristic has the highest running time.

IV. Simulation Results

In this section, we evaluate the performance of the various heuristic algorithms on several different topologies and parameter settings. For simplicity, we make the reasonable assumption of shortest path routing. The focus of our work is to optimize the measurement infrastructure for service provider networks. However in the interest of discussing the performance of the algorithm we also present results for enterprise networks.

A. Simulation Setup

Since the service provider network is inherently hierarchical with metro and core networks, we chose to generate hierarchical network topologies. We use the Tier topology generator [8] for service provider networks. In the service provider network there are three types of links: (1) intra-core link; (2) intra-metro link; and (3) metro-to-core link. In our simulation we have three scenarios with different capacity allocation schemes as shown in Table-IV. These schemes are reasonable approximations of today's provider networks. For each bandwidth tier we evaluate our algorithms at network sizes of 200, 400 and 800 nodes.

	Link Type		
	1	2	3
Tier 3	OC48	OC3	OC12
Tier 2	OC48	OC12	OC12
Tier 1	OC192	OC48	OC48

TABLE IV

BANDWIDTH ASSIGNMENT SCHEMES FOR SERVICE PROVIDER NETWORKS

For enterprise networks we assume a flat topology and all the links are assumed to have the capacity of fast Ethernet. We generate the flat network topology using the GT-ITM topology generator[22]. The network graph used in this study is the Waxman model [20].

In our simulations, we assume that each interface or MPLS tunnel requires a polling bandwidth of 4kbps/sec. The number of MPLS tunnels per node is randomly chosen from the range of [1,1000]. To the best of our knowledge these parameters are typical of service provider environments. Since we expect the number of MPLS tunnels to be large, for scalability reasons we assume that the tunnels are only monitored at the source and destination nodes. We choose not to poll core nodes for MPLS specific information. For the enterprise network, the number of MPLS tunnels per node is randomly chosen from the range of [1,500].

For each tier of service provider networks we generate 5 topologies. In the case of enterprise networks we generated different network topologies by varying the parameter β for a fixed α. The varying β gives topologies with degree of connectivity ranging from 3 to 8. Each link allocates a certain fraction of their bandwidth for polling traffic based on the capacity constraint imposed for the link. This fraction is the same for all links in our simulation. We call this parameter LINKF. The results presented are averaged over the different topologies.

The goal of our simulation study is to see the power of the heuristics to compute a "good" solution that can be used in practice to design network measurement infrastructure. Our performance metrics are (1) total number of pollers required, and (2) fraction of total bandwidth consumed for polling.

B. Performance Comparison Among Greedy Heuristic Algorithms

	Link Fraction for Polling	Number of Nodes 200	400
HmaxLoad	5%	(15±3, 0.91%)	(32±7, 0.85%)
	10%	(7±3, 1.50%)	(16±2, 1.38%)
HmaxPollee	5%	(6±2, 0.94%)	(15±4, 0.97%)
	10%	(2±1, 1.45%)	(6±2, 1.33%)
Random	5%	(13±2, 0.84%)	(29±2, 0.89%)
	10%	(6±1, 1.33%)	(13±4, 1.35%)

TABLE V

PERFORMANCE OF HEURISTIC ALGORITHMS FOR TIER 2 SERVICE PROVIDER NETWORKS. EACH TUPLE IS (NUMBER OF POLLERS REQUIRED WITH VARIANCE, FRACTION OF TOTAL BANDWIDTH USED.)

Table-V compares the performance of the *HmaxLoad*, *HmaxPollee* and *HRandom* algorithms. Each tuple in the Table V represents the average number of pollers used with the variance term and the fraction of bandwidth used for polling. From Table V, we see that in the Tier 2 service provider architecture, the *HmaxLoad* performs worse than *HRandom*. *HmaxPollee* performs much better than the other two heuristics in terms of the number of pollees required with some increase in polling bandwidth. When LINKF = 5%, *HmaxPollee* needs only 6 pollers whereas *HmaxLoad* and *HRandom* needs 9 and 7 additional pollers respectively. When LINKF = 10%, *HmaxPollee* needs just 2 pollers whereas *HmaxLoad* and *HRandom* needs 5 and 4 additional pollers. Note that in each case there is a clear tradeoff between the number of pollers and the bandwidth consumed. Although *HmaxPollee* requires fewer number of pollers, it consumes more bandwidth than the other two heuristics. The performance of the *HmaxPollee* with respect to the *HRandom* heuristic remains the same regardless of the capacity constraint. This implies that in order to reduce the number of pollers required, choosing a poller based on the ability to poll maximum number of nodes is the best scheme (note that *HmaxPollee* and *HRandom* differ only at this step of picking the poller). However, the reduction in the number of pollers comes at the cost of bandwidth consumed.

We also observe that as expected by increasing the capacity reserved (LINKF parameter) for polling we can always reduce the number of pollers regardless of the heuristics used. This further highlights the importance of the bandwidth constraint in the formulation of this problem.

	Link Fraction	Number of Nodes 200	400
shuffle	5%	(6±2, 0.94%)	(15±4, 0.97%)
	10%	(2±1, 1.45%)	(6±2, 1.33%)
no shuffle	5%	(7±5, 1.21%)	(20±9, 1.32%)
	10%	(2±1, 1.55%)	(6±3, 1.76%)

TABLE VI

EFFECT OF POLLEE SHUFFLE USING HMAXPOLLEE

Effect of *ShuffleAndReduce*: In our study we found that the procedure $ShuffleAndReduce$ (shown in Table III) is very effective. Table VI illustrates this using *HmaxPollee* as an example. Note that the effect of the pollee shuffle is primarily aimed at reducing the bandwidth consumed. Without ShuffleAndReduce, bandwidth usage would be 29% more in the 200 node networks when LINKF = 5%. Also as shown in Table VI for 400 node networks without shuffling we need 5 more pollers when LINKF = 5%; In addition, bandwidth usage would increase by 36%. *ShuffleAndReduce* reduces the number of pollers because it clusters the pollees around the closest poller. This results in reduced bandwidth consumption at each step, which in turn reduces the total number of pollers needed.

	Link Fraction	Number of Nodes 200	400
HmaxLoad	5%	(42±8, 0.66%)	(89±13, 0.59%)
	10%	(23±5, 1.25%)	(53±7, 1.13%)
HmaxPollee	5%	(27±3, 0.89%)	(65±10, 0.75%)
	10%	(12±5, 1.53%)	(29±7, 1.36%)
Random	5%	(44±4, 0.73%)	(101±10, 0.69%)
	10%	(22±5, 1.22%)	(51±9, 1.31%)

TABLE VII

PERFORMANCE OF HEURISTIC ALGORITHMS FOR TIER 3 SERVICE PROVIDER NETWORKS.

Effect of Link Capacity: From Table VII and Table VIII,

	Link Fraction	Number of Nodes	
		200	400
HmaxLoad	5%	(3±1, 0.41%)	(7±1, 0.56%)
	10%	(2±0, 0.43%)	(4±2, 0.62%)
HmaxPollee	5%	(1±0, 0.40%)	(1±0, 0.49%)
	10%	(1±0, 0.40%)	(1±0, 0.54%)
Random	5%	(3±1, 0.42%)	(6±1, 0.49%)
	10%	(2±1, 0.46%)	(3±0, 0.58%)

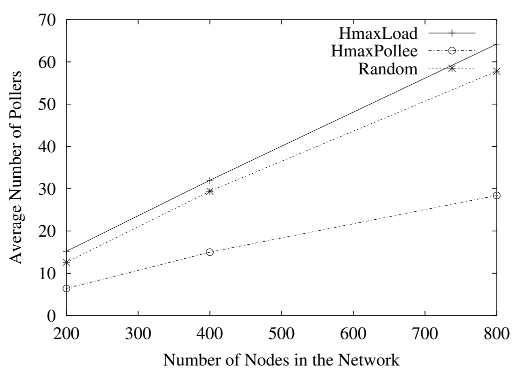
TABLE VIII
PERFORMANCE OF HEURISTIC ALGORITHMS FOR TIER 1 SERVICE PROVIDER NETWORKS

	Link Fraction	Number of Nodes	
		100	200
HmaxLoad	5%	(10±4, 1.38%)	(9±2, 0.74%)
	10%	(5±1, 1.82%)	(4±1, 0.81%)
HmaxPollee	5%	(6±1, 1.63%)	(5±0, 0.79%)
	10%	(3±1, 1.72%)	(3±0, 0.82%)
Random	5%	(12±1, 1.49%)	10±2, 0.78%)
	10%	(5±1, 1.87%)	(5±1, 0.84%)

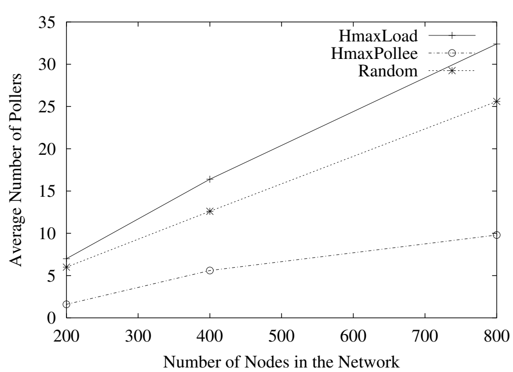
TABLE IX
PERFORMANCE OF HEURISTIC ALGORITHMS FOR ENTERPRISE NETWORKS (100 NODES: WAXMAN α = .2, $\beta = 0.15$).

we see that the number of pollers needed is inversely proportional to the network capacity. In tier 1 where network capacities are less than in tier 3 we require more pollers. However as expected as the number of pollers increases the bandwidth consumed decreases as well. This result further justifies our distributed polling architecture when compared to a centralized system. Furthermore, the *HmaxPollee* algorithm still performs the best among the three heuristics.

The trend on the number of pollers as a function of network size is shown in Figure 2. We see that *HmaxPollee* scales better than the other two algorithms. The graph for LINKF=5% shows that there is almost no distinction between *HmaxLoad* and *HRandom* suggesting that greedily picking the next poller based purely on polling load is not a good heuristic. It is only as good as randomly picking the next poller.



(a) LINKF=5%



(b) LINKF=10%

Fig. 2. Minimum number of Pollers as a function of Network Size

Application to Enterprise Networks: From Table-IX we see that the performance of the *MaxPollee* heuristic is the best among the three algorithms. The impact of the average degree of connectivity on the performance of the *MaxPollee* heuristic is presented in Table-X. The average degree of connectivity varies from 3.5 to 8 as β varies from 0.08 to 0.2 for a fixed α of 0.2. Note that the performance of the *HMaxPollee* heuristic improves as the degree of connectivity increases.

C. Comparison with LP Relaxations

Ideally we would like to see how far our solutions are from the optimal solution. We are unable to obtain optimal solution for any reasonable sized network since our integer programming solver [1] could not compute the optimal solutions even for a small network of just 50 nodes. Therefore we use the LP relaxation of our problem to compute a crude lower bound on the optimal solution. The LP solution can be computed in reasonable time for relatively large networks using ALPO [19]. Since *HmaxPollee* performs the best among the three greedy heuristics, we only need to compare *HmaxPollee* with the *LP* lower bound. We only present the results on the topology that gives the worst approximation ratio. The approximation ratio is defined as the ratio between the number of pollers given by *HMaxPollee* and the best possible estimate of the lower bound. A tie is broken by using the largest fraction of bandwidth usage. Table-XI presents the comparison between our results and those obtained using the LP bound. Note that, the second item in the tuple for LP bound is the fraction of bandwidth used given that the number of pollers used is the same as *HmaxPollee* in the respective cases (not the fraction of bandwidth consumed for the LP relaxation of minPL problem.).

	Link Fraction	Number of Nodes	
		200	400
HmaxPollee	5%	(7, 0.70%)	(18, 0.91%)
	10%	(2, 1.62%)	(7, 1.43%)
LP bound	5%	(2, 0.52%)	(-, -)
	10%	(1, 1.24%)	(-, -)

TABLE XI
COMPARISON OF HMAXPOLLEE WITH LP LOWER BOUND: TIER 2 SERVICE PROVIDER NETWORKS

For the 200 node tier 2 network[5], the lower bound on the number of pollers is 2 when LINkF = 5% and 1 when

[5] Our LP solver cannot solve the 400 node case in reasonable time.

	Link Fraction	β				
		0.08	0.1	0.15	0.18	0.2
HmaxPollee	5%	(9, 1.72%)	(8, 1.35%)	(5, 0.73%)	(5, 0.58%)	(4, 0.57%)
	10%	(4, 2.03%)	(3, 1.52%)	(3, 0.75%)	(2, 0.62%)	(2, 0.56%)

TABLE X

PERFORMANCE OF *HmaxPollee* FOR ENTERPRISE NETWORKS (200 NODES: WAXMAN α = .2).

LINKF=10%. Given the number of pollers to be 7 and 2 for LINKF = 5% and 10%, the lower bound on the fraction of bandwidth used is 0.52% and 1.24% which is very close to the bandwidth fraction (0.7% and 1.62%) required for *HmaxPollee*.

	Link Fraction	Number of Nodes	
		100 ($\beta = 0.15$)	200 ($\beta = 0.08$)
HmaxPollee	5%	(6, 1.79%)	(9, 1.72%)
	10%	(4, 1.65%)	(4, 2.03%)
LP bound	5%	(1, 1.02%)	(1, 0.99%)
	10%	(1, 1.24%)	(1, 1.38%)

TABLE XII

COMPARISON OF HMAXPOLLEE WITH LP LOWER BOUND: ENTERPRISE NETWORKS

The LP lower bound for enterprise networks is shown in Table XII. The relative difference between *HmaxPollee* and LP lower bound is similar to that in the case of service provider networks. From the Table we see that the LP solution has 1 poller while the *HmaxPollee* solution has 6 pollers. Note that the optimal number of pollers needed is at least two. The reason is that *RestrictedMaxPollee* gives the optimal solution if only one poller is needed. Since the solution obtained from the LP relaxation is 1 and the optimal is at least 2 in this problem instance, we can say that the integrality gap of the *minPL* problem is at least 2. Integrality gap is defined as the ratio of the optimal solution to the solution obtained using LP relaxation[14].

V. RELATED WORK

The problem formulation presented in this paper is unique when compared to prior work done in the area of network measurements. In [3], the authors solve the problem of identifying the optimal set of nodes that must be monitored for measuring bandwidth utilization on the network. This problem is approximated by a *minimum vertex cover* formulation. In the same work they also address the question of identifying the optimal set of probes for latency measurements. They show that this problem can be formulated as a *facility location* problem. In both cases there is an assumption that there is only a single poller in a given network. In our work, we require that the network has distributed pollers. This assumption allows for more fine-grained monitoring on all the nodes.

Similar to our approach, the work done by Jamin et. al [11], has a distributed system flavor. This work addresses the problem of optimal placement of measurement instrumentation. The objective of this problem is very similar to our minimum pollers problem however, the two formulations differ in the constraints imposed. In Jamin's work they propose two graph theoretic approaches to determine the number and placement of the measurement instrumentation. Both of these methods require *a priori* knowledge of network topology. In the first approach they solve an instrumentation location problem that is constrained by a maximal center to node distance. This problem is solved using *k-hierarchically well-separated trees*. In the second approach they assume that the number of measurement centers is given and one needs to decide their location such that the maximum distance between the node and the center is minimized. This problem is solved by mapping to a *minimum k-center* problem. Liotta [13] also formulated the problem of computing agent locations as a *minimum k-center* and *minimum k-median* problem. He proposed several novel approximate solutions to reduce the computational overhead of existing solutions [7]. Our problem formulation is different since it poses a constraint on the measurement bandwidth on any given link regardless of the distance between the poller and the pollees. Thus the first approach proposed in [11] does not apply to our case. Furthermore, since the goal of our problem is to minimize the number of pollers required in a given network topology, the second approach which assumes that the number of measurement centers is known is also not suitable.

VI. DISCUSSION

The primary motivation for our work is based on the assumption that in order to design good measurement infrastructure it is necessary to have a scalable system at a reduced cost of deployment. With this view we proposed a *hierarchical measurement architecture*. The key feature of this architecture was to distribute the resource intensive tasks across the network. One such task is the polling of individual nodes. The number and choice of these pollers has a significant impact on the cost of the measurement infrastructure. The cost is due to both the deployment of the instrumentation as well as the bandwidth consumed by the measurement traffic. Our study shows that it is possible to design such an infrastructure and attain significant reduction in bandwidth overhead as well as in the cost of deployment of the pollers.

Based on our simulation studies we see that the total number of pollers for a given network topology can be reduced using the *MaxPollee assignment scheme*. The MaxPollee assignment scheme aggressively chooses the next poller with the intent of assigning all of the remaining nodes to the new poller. However, the reduction in the number of pollers comes at the cost of the bandwidth consumed. We also observe that by increasing the capacity reserved (LINKF parameter) for polling we can always reduce the number of pollers regardless of the heuristics used. This further highlights the importance of the *bandwidth constraint* in the formulation of this problem. One other factor that affects the performance of the maxPollee algorithm is the degree of connectivity in the network. We observed that for enterprise network with high degree of connectivity fewer pollers were required as compared to the service provider network.

Our results show that both in the case of the service provider network as well as the enterprise network we are able to identify a poller set and a corresponding assignment scheme without violating any bandwidth constraints. Only in 6% of the cases was it necessary to use more than 90% of the allocated bandwidth. We believe that we were able to attain such efficient use of bandwidth due to the *ShuffleAndReduce* that was performed for every new additional poller chosen. Without the explicit consideration of bandwidth issues it is highly likely that we could create bandwidth bottlenecks due to network measurements.

The distributed poller location scheme presented here can be used in the design phase of a network management system. For a network of about 200 nodes we can obtain the poller set and the assignment scheme in just a few minutes. Thus with little effort the network operations center can identify the placement of their measurement instrumentation. When choosing pollers we assume that the pollers have sufficient power to handle the polling load for the assigned pollees.

VII. Conclusion and Future Work

In this paper, we have addressed the problem of optimizing the distributed polling system with respect to the number of pollers as well as the overall bandwidth consumption due to network measurements. To the best of our knowledge this work is the first formulation that includes per-link bandwidth constraints as part of optimizing instrumentation location. Furthermore by including the bandwidth constraint, our problem formulation has wider applicability to areas such as identifying optimal server locations.

Of the different heuristics that were used, the maximum pollee assignment scheme performed the best in terms of the minimum number of pollers required as well as scalability with respect to network size. Interestingly, there was almost no distinction between the heuristics of choosing pollers based on the polling load and a random pick. We believe that the polling load bias among the nodes has to be significantly higher for load based criteria to be effective. There is on-going work to explore this issue in more detail.

Our future work is to increase the scope of our problem formulation by accounting for delay constraints. Currently we assume that routes are fixed between any given pair of nodes. In an MPLS enabled network, one can choose different routes between any given pair of nodes by setting up Label Switched Paths (LSP) between them. Therefore, we hope to use this to our advantage by including the computation of the best routes between poller-pollee pairs as part of the pollee assignment scheme. In addition, we would also like to model the computational power by limiting the number of pollees that can be assigned to a poller.

References

[1] Michel Berkelaar. lp_solve 3.0 software. ftp://ftp.es.ele.tue.nl/pub/lp_solve/.

[2] J. C. Bolot. End to end packet delay and loss behavior in the Internet. In *Proc. of ACM SIGCOMM*, pages 289–298, 1993.

[3] Y. Breitbart, C. Y. Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz. Efficiently monitoring bandwidth and latency in IP networks. In *Proc. IEEE Infocom*, pages 933–942, 2001.

[4] D. Breitgand, D. Raz, and Y. Shavitt. SNMP GetPrev: An efficient way to access data in large MIB tables. *IEEE Journal of Selected Areas in Communication*, 20(4):656–667, 2002.

[5] R. Caceres, N. G. Duffield, A. Feldmann, J. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. Kalmanek, B. Krishnamurthy, D. Lavelle, P. P. Mishra, K. K. Ramakrishnan, J. Rexford, F. True, and J. E. van der Merwe. Measurement and analysis of IP network usage and behavior. *IEEE Communications Magazine*, pages 144–151, May 2000.

[6] M. Cheikhrouhou and J. Labetoulle. An efficient polling layer for SNMP. In *Proc. of the IFIP/IEEE Network Operations and Management Symposium*, pages 477–490, 2000.

[7] M. S. Daskin. *Network and Discrete Location: Models, Algorithms and Applications*. John Wiley and Sons, Inc., New York, 1995.

[8] M. Doar. A better model for generating test networks. In *Proc. of IEEE Globecom*, pages 86–93, November 1999.

[9] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewicz, and Y. Jin. An architecture for global Internet host distance estimation service. In *Proc. of IEEE Infocom*, pages 210–217, 1999.

[10] Van Jacobson. *pathchar* - a tool to infer characteristics of internet paths, ftp://ftp.ee.lbl.gov/pathchar.

[11] S. Jamin, C. Jin, Y. Jin, D. Raz anf Y. Shavitt, and L. Zhang. On the placement of Internet instrumentation. In *Proc. of IEEE Infocom*, pages 295–304, 2000.

[12] K. Lai and M. Baker. Measuring bandwidth. In *Proc. of IEEE Infocom*, pages 235–245, 1999.

[13] A. Liotta. Towards flexible and scalable distributed monitoring with mobile agents. PhD Thesis, University College London, London, UK, August 2001.

[14] M. Pal, E. Tardos, and T. Wexler. Facility Location with Nonuniform Hard Capacities. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pages 329–338, 2001.

[15] C. Srinivasan, A. Viswanathan, and T. D. Nadeau. MPLS traffic engineering MIB. *Internet Draft*, July 2002.

[16] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison Wesley Longman, Inc, 3 edition, 1999.

[17] W. Theilmann and K. Rothermel. Dynamic distance maps of the Internet. In *Proc. of IEEE Infocom*, pages 275–284, 2000.

[18] M. Thottan and C. Ji. Using network fault prediction to enable IP traffic management. *Journal of Network and Systems Management*, 9(3):327–346, 2001.

[19] R. J. Vanderbei. Alpo: Another linear program optimizer. *ORSA J. Computing*, 5(2):134–146, 1993.

[20] B.M. Waxman. Routing of multipoint connections. *IEEE Journal of Selected Areas in Communication*, 6(9):1617–1622, 1988.

[21] Y. Yemini, G. Goldschmidt, and S. Yemini. Network management by delegation. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management*, pages 95–107, 1991.

[22] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE Infocom*, pages 594–602, 1996.

Appendix

A. The Generalized maxPollee Problem is NP-hard

Theorem 1.1: Given a graph $G = (V, E)$ and a distinct poller node $s \in V$. Each node v has a demand $w(v)$. Each edge $e \in E$ has a capacity $b(e)$. For each node $v \in V - \{s\}$, a unique path P_{vs} specifies the route from v to s. Each edge along P_{vs} consumes $w(v)$ amount of resources if v's polling demand is shipped to s. Finding the maximum set of nodes J such that their polling demands can be shipped to s along the fixed paths without violating the capacity constraint on each edge is NP-hard.

Proof: We reduce the Maximum Independent Set problem to the maxPollee problem. An *independent set* in a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that, for all $u, v \in V'$, the edge (u, v) is *not* in E. The Maximum Independent Set problem asks that, for a given graph $G = (V, E)$, find an independent set V' such that for any other independent set V'' of G, $|V'| \geq |V''|$.

Given a Maximum Independent Set problem, we construct an instance of the maxPollee problem using the following steps. (1) For each node v_i, create $d_i - 1$ additional nodes where d_i is the node degree of v_i. Lets number these d_i nodes as $v_i^0, v_i^1, \cdots, v_i^{d_i-1}$ where v_i^0 in the new graph $G^- = (V^-, E^-)$ corresponds to node v_i in G. (2) For each node $v_i \in G$, order the nodes that share an edge with it. $Rank(i,j)$ gives the rank of node v_j according to v_i's ordering. For each edge $(v_i, v_j) \in E$, connect node $v_i^{Rank(i,j)}$ to $v_j^{Rank(j,i)}$. (3) For each node $v_i \in G$, construct paths P_i from v_i^0 to node s such that P_i must contain edge $(v_i^{Rank(i,j)}, v_j^{Rank(j,i)})$ for each node $v_j \in G$ which shares an edge with v_i in G. (4) Assign demand 1 to each node v_i^0 and assign demand ∞ to each node v_i^k where $k > 0$. Set $b(e) = 1$ for each edge in E^-.

Note that this construction satisfies the following property: for each node $v_i^0 \in G^-$ which corresponds to a node $v_i \in G$; its path P_i to s uses all the links corresponding to those incident on $v_i \in G$. Therefore, if v_i is chosen in the solution set, then none of the nodes who share a link with v_i in G can be chosen in the solution set. Therefore, the solution set to the maxPollee problem corresponds to a solution set in the Independent set problem. Therefore, if we can determine the maximum pollee set, then we can determine the maximum independent set. ■

B. The Optimal Pollee Assignment Problem is NP-hard

Theorem 1.2: Given a graph $G = (V, E)$ and a distinct poller set $S \in V$. Each node v has a demand $w(v)$. Each edge $e \in E$ has a capacity $b(e)$. For each node $v \in V - S$ and $s \in S$, a unique path P_{vs} specifies the route from v to s. Each edge along P_{vs} consumes $w(v)$ amount of resources if v's demand is chosen to ship to s. It is NP-hard to determine the best λ such that for each node $v_i \in V - S$, $\lambda(v_i)$ gives the poller it is assigned to, and the total bandwidth consumption is minimal subject to the capacity constraint on each link.

Proof: We reduce the *bin packing* problem to our Pollee Assignment problem. Given a finite set U of items, a size $t(u) \in Z^+$ for each $u \in U$, a positive integer bin capacity B, and a positive integer K. The bin packing problem asks for a partition of U into disjoint sets $U_1, U_2, \cdots, U_K$ such that the sum of the sizes of the items in each U_i is B or less.

Given an instance of bin packing problem, we create an instance of our Pollee Assignment problem: for each bin i, create a poller s_i and an extra node v_i; Assign zero demand to s_i and v_i. Connect s_i and v_i with an edge of capacity B. For each items $u \in U$, create a pollee u with demand $t(u)$ and connect u and each node v_i with an edge of capacity ∞.

Clearly, if Pollee Assignment problem has solution with less than KB total bandwidth consumption, then items in U can be partitioned into K disjoint sets with the size constraint. On the other hand, if the bin packing problem can be partitioned into K disjoint set, a corresponding optimal Pollee Assignment solution can be constructed by assigning a pollee to the poller that represent the bins it is assigned to. Therefore, our Pollee Assignment problem is NP-hard. ■