Distributed Network Monitoring and Multicommodity Flows: A Primal-Dual Approach

Baruch Awerbuch*
Johns Hopkins University.
baruch@cs.jhu.edu

Rohit Khandekar
IBM T.J. Watson Research Center.
rkhandekar@gmail.com.

ABSTRACT

A canonical distributed optimization problem is solving a Covering/Packing Linear Program in a distributed environment with fast convergence and low communication and space overheads. In this paper, we consider the following covering and packing problems, which are the dual of each other:

- *Passive Commodity Monitoring*: minimize the total cost of monitoring devices used to measure the network traffic on all paths.
- *Maximum Throughput Multicommodity flow*: maximize the total value of the flow with bounded edge capacities.

We present the first known distributed algorithms for both of these problems that converge to $(1+\epsilon)$-approximate solutions in poly-logarithmic time with communication and space overheads that depend on the maximal path length but are almost independent of the size of the entire network. Previous distributed solutions achieving similar approximations required convergence time, communication, or space overheads that depend polynomially on the size of the entire network. The sequential simulation of our algorithm is more efficient than the fastest known approximation algorithms for multicommodity flows, e.g., Garg-Könemann [14], when the maximal path length is small.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: [Non-numerical Algorithms and Problems]

General Terms

algorithms, theory

*Partially supported by NSF grants CCF 0515080, ANIR-0240551, CCR-0311795, and CNS-0617883.

PODC'07, August 12–15, 2007, Portland, Oregon, USA.

Keywords

distributed algorithms, multi-commodity flows, passive network measurement

1. INTRODUCTION

The distributed *Passive Flow Monitoring* (PFM) is used by network routers with attached storage devices to observe and record the packet traffic on operational network links, without injecting any traffic of its own onto the network. That is, the monitoring or logging device is *non-intrusive*. Our objective is to accomplish recording, at random, some percentage of network traffic without imposing too much overhead on the recording devices. The goal of monitoring is figuring out how to measure or log network activities, with the objective of covering all the paths using monitoring devices.

The problem can be reduced to the classical set cover problem, which involves an exponential number of elements (paths); thus this reduction is not efficient. This problem has not been considered in the theoretical computers science literature, but it has been well addressed in the applied community. Many conferences are dedicated to network monitoring and measurements, and many scientific and engineering papers have been published on the topic [20, 13, 21, 18, 15].

The *Maximum Throughput Multicommodity Flow* (MCF) is a classical optimization problem that directly addresses practically important issues of congestion and bandwidth management in connection-oriented network architectures. The objective here is to maximize the total flow that can be routed subject to edge-capacities. It can be formulated as a packing linear program which turns out to be the dual of the covering linear program of the PFM problem. In the theoretical computer science, it has been first introduced, in the same distributed model as ours, by Awerbuch and Leighton [7, 8].

1.1 Distributed ROUTERS Model

This paper deals with the "classic" distributed network ROUTERS model [2, 10, 9] which is based on an assumption that intelligence is embodied in the network routers and the computation proceeds via message exchanges between the neighboring routers. More precisely, we consider a network comprised of *routers*, connected to each other via communication links. The network graph $G = (V, E)$ has nodes V representing the routers and edges E representing the links. Each router can exchange message with its neighbors in the graph. Each router is also aware of all the commodities in

the network. The communication between neighbors takes exactly one time unit of a globally accessible clock [2].

This model does represent some actual routing protocols (e.g., BGP). For each destination, routers need to pick an adjacent edge over which the traffic toward that destination will be forwarded to (in contrast to source routing which specifies the whole path). Such protocols are also called "distance vector routing", because they maintain a "proximity" metric for each destination, with packets flowing in the direction of decreased metric, without knowing the structure of the graph.

Complexity measures. We evaluate our algorithms on the basis of the following complexity measures [2, 9, 10].

- The *approximation ratio* measures the quality of the output as compared to the optimum solution.
- The *message congestion* is the maximum number of messages traversing an edge.
- The *space complexity* is the maximum amount of space maintained per edge.
- The *convergence time* is the number of rounds needed to converge to a desired output.
- The *computational complexity* is the total number of computational steps needed in a centralized simulation of the distributed algorithm.

Even though it is highly undesirable in practice for a variety of reasons, such as security, fault tolerance and confidentiality, our model considers collecting all the information into a single router and applying a centralized solution, e.g., a Linear Program, to be a perfectly legitimate solution. Essentially, a distributed algorithm for a network problem such as MST, shortest path, or maximum flow must compete with such a centralized solution. Typically, the centralized solutions for graph problems have a large space or message congestion overhead, and thus are considered inferior to the distributed solutions with (typically) near-constant space and message congestion overheads, e.g., [3, 2, 4].

1.2 Passive Commodity Monitoring Problem

We model the network by a directed graph $G = (V, E)$ where V is the set of nodes while E is the set of links (routers) between them. There are k *commodities* each specified by a source node s_i, a sink node t_i, and a demand $d_i > 0$.

There is a router associated with each network link that is capable of doing traffic monitoring. The router for a link $e \in E$ needs to decide a frequency $x_e \geq 0$ of monitoring the traffic going through e. At this frequency, the monitoring device of router e makes x_e measurement probes per unit time. We assume that it costs $c_e x_e$ to measure the traffic at frequency x_e on link e, where $c_e > 0$ is a constant. Obviously, if all the routers measure all the flow passing through them at all times, i.e., $x_e = 1$, then all the flow gets recorded. However, in this case the effort exerted by the routers is highly redundant, e.g., the same flow gets recorded by many routers.

Since the links on a path measure the traffic going through them independently with respect to each other, the frequencies along a path add up to give the total monitoring frequency on that path.

Monitoring only short paths. Input to the problem is also an integer $L > 0$. We assume that each commodity i routes its flow only along s_i-t_i paths that have hop-length at most L. Thus only such paths need to be monitored. In many applications, L is essentially a constant, e.g., $L = 1$ in bi-partite case.

Objective. *Find minimum-cost set of the monitoring frequencies x_e on the links so that each path of at most L hops from s_i to t_i for each commodity i gets measured at a total frequency that is at least the demand d_i.*

This problem can be cast as a "covering" linear program as follows. Let $\mathcal{P}_i$ be the set of paths of at most L hops between s_i and t_i. For a technical reason, we also include non-simple s_i-t_i paths of hop-length at most L in $\mathcal{P}_i$. Note that the hop-length of a non-simple path counts edges with their multiplicities.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e & & & \\ \text{s.t.} \quad & \sum_{e \in p} x_e & \geq & d_i & \forall i \text{ and } p \in \mathcal{P}_i \\ & x_e & \geq & 0 & \forall \text{ links } e \in E \end{aligned} \tag{1}$$

Note that this problem can be thought of as a set cover problem, with paths being elements and routers (edges) being sets. The problem with this reduction is that there is an exponential number of elements to be covered. Aggregating all the paths for a single commodity into a single commodity "super-element" allows a possibility of a polynomial overhead solution, but does not let us use the set cover framework in a direct way.

1.3 Maximum Multicommodity Flow Problem

We again model the network by a directed graph $G = (V, E)$ where each link e has a capacity $c_e > 0$. There are k *commodities* each specified by a source node s_i, a sink node t_i, and a profit $d_i > 0$. Routing a unit flow between s_i and t_i accrues a profit of d_i.

Flows only along short paths. Input to the problem is also an integer $L > 0$. We assume that each commodity i routes its flow only along s_i-t_i paths that have hop-length at most L. Again in many practical applications, L is much smaller than the total number of vertices.

Objective: *Route the flows for these commodities so that the aggregate profit is maximized while the edge-capacities are satisfied.*

Let $\mathcal{P}_i$ denote the set of (perhaps non-simple) s_i-t_i paths of hop-length at most L. We can formulate the maximum multicommodity flow problem as a "packing" linear program as follows:

$$\begin{aligned} \max \quad & \sum_i d_i \sum_{p \in \mathcal{P}_i} f_p & & & \\ \text{s.t.} \quad & \sum_{p: e \in p} f_p & \leq & c_e & \forall \text{ edges } e \in E \\ & f_p & \geq & 0 & \forall \text{ paths } p \in \cup_i \mathcal{P}_i \end{aligned} \tag{2}$$

This packing formulation has a number of variables that is exponential in L. However, aggregating the paths for the ith commodity allows a possibility of a polynomial overhead solution.

REMARK 1.1. *The linear programs (1) and (2) are LP duals of each other.*

1.4 Our results

Let $n = |V|$, $m = |E|$, and $\mathrm{P} = n^L$ be an upper bound on the total number of paths of hop-length at most L. Let $C = \max_e c_e / \min_e c_e$ and $D = \max_i d_i / \min_i d_i$.

THEOREM 1.2 (MAIN). *The algorithms in Sec. 3 and 4 are $(1+\epsilon)$-approximation algorithms for the passive commodity monitoring (1) and the maximum multicommodity flows (2) in the* ROUTERS *model and have the following features:*

- *The* convergence time *is*
$$O\left(L^3 \cdot \frac{\log^2(n\mathrm{C}/\epsilon)}{\epsilon^4} \cdot \log \frac{mCD}{\epsilon}\right).$$
- *The* space per router *is* $\tilde{O}(k \cdot L)$.
- *The* messages per router *are* $\tilde{O}\left(k \cdot L^3\right)$.
- *The* total computation overhead *is* $\tilde{O}\left(m \cdot k \cdot L^3\right)$.

See Figure 1 and subsection 1.5 for a comparison of our results with existing work. Note that router space and messages per router are *linear* in the number of commodities k and is *poly-logarithmic* in the network size m. Since there are k flows going through a router, $\Omega(k)$ is a lower bound on these quantities. Surprisingly, all existing distributed flow algorithms have either space or message congestion bounds or convergence bounds above $\Omega(m)$. One of our major contribution in this work is to break this barrier. Our secondary contribution is to develop a solution to PFM problem, which has been the original motivation for this line of research. Surprisingly, focusing on the PFM problem, which is the dual of MCF, enables us to achieve improvements in both *distributed* and *sequential* models for classical maximum flow algorithms thus improving on best known solutions for MCF in [7, 8] and [14, 12, 22], respectively, on some graph instances.

1.5 Existing work

Sequential algorithms. Garg-Könemann [14] and Fleischer [12] have developed the best known multicommodity flow algorithms for the *centralized setting*. A sequential simulation of our distributed algorithm is significantly faster than [14, 12] on graphs with short flow paths (see Figure 1).

Existing Flow algorithms for our model. Multicommodity flow algorithms in ROUTERS model are presented in [7, 8]. Their algorithm cannot be used in our model because the actual flows sent in this algorithm violate flow preservation at intermediate nodes, accumulating flow excess at nodal queues. In our model, flows must obey conservation laws. However, our algorithm dominates this algorithm in space, times, communication and computational complexities. It is worth pointing out that [7, 8] have additional fault-tolerance feature, such as *statelessness*, which is extremely important feature, that is absent in our solution.

BILLBOARD **Flow algorithms.** The BILLBOARD model is used in essentially all of the work on "congestion games", where "flow agents" optimize their routes while observing congestion of the links over the global "billboard" [1, 19, 11, 5]. In this model, there exist distributed solutions to multicommodity flows such as [11, 6, 5]. The choice between BILLBOARD versus ROUTERS model for flow optimization is actually a major architectural decision. Essentially, this is the choice of using *packet routing* with all the intelligence at the routers and "dumb" end-to-end users (flow agents) versus *source routing* with "dumb" routers and all the intelligence at the end-to-end users (flow agents). In the BILLBOARD model, one assumes that routers are dumb, i.e., each packet sent by the source carries the full description of the path, and routers just forward the packet to the next hop. The routing algorithms for this model are also called "Link-State", because they maintain at an end-point (source) of the flow the database of all the network links, *including* current utilization of that link. Maintaining link state database at all the flow sources means that utilization of all the links must be continuously reflooded through the network, resulting in a larger maintenance cost.

Just like the BILLBOARD-based flow algorithm in [6], we use a primal-dual approach. Adapting BILLBOARD-based algorithms to ROUTERS model is un-attractive since it either increases space complexity or convergence time to $\Omega(m)$. Direct implementation of link-state requires communication and space overheads of $\Omega(m)$ at the sources of the flows. Indirect implementations using distance vector approaches slows down the algorithm by $\Omega(m)$. In contrast, router-based algorithms described in this paper require space overhead of $O(k \cdot L)$ where k is the number of commodities, and L is the maximal path length. (In practice, $L < 15$ on the Internet.)

2. ALGORITHMS FOR SET-COVER AND ITS DUAL

In Section 2.1, we first present an algorithmic framework for computing a $(1 + \epsilon)$ approximation to the fractional set-cover problem and its dual. Later, in Section 2.2, as an instance of this framework, we present our distributed $(1+\epsilon)$-approximation algorithm with poly-logarithmic running time. The framework presented in these sections is based on the parallel algorithm of Luby and Nissan [16] for packing/covering linear programs and similar results by Garg-Könemann [14] and Young [22].

Our novel technical contributions, presented in Sections 3 and 4, are in extending this framework to poly-logarithmic solutions for our problems in the distributed model, with only *polynomial* overhead, and using only *local* information.

2.1 Algorithmic Framework for Set-cover and its Dual

In the set cover problem, we have a set $\mathcal{U}$ of clients (elements) to be served (covered) and a collection of servers (sets) $\mathcal{S}$. Each server can serve only some clients, as described by a binary relation $\mathcal{R} \subset \mathcal{U} \times \mathcal{S}$ where tuples $(u, S) \in \mathcal{R}$ indicates that server (set) $S \in \mathcal{S}$ can serve (cover) a client (element) $u \in \mathcal{U}$. A server is also associated with a cost $c_S \geq 0$ of usage. The objective in the classical *fractional* set-cover problem is to pick the servers S to "extents" $x_S \geq 0$ such that each client $u \in \mathcal{U}$ is served to a total extent of at least one: $\sum_{(u,S)\in\mathcal{R}} x_S \geq 1$ while minimizing the total cost $\sum_S c_S x_S$. It is formulated as the following "covering" LP.

Problem	Reference	Rounds	Messages	Space	Computation
MCF	[14, 12, 22]	$\tilde{O}(m+k)$	$\tilde{O}(m+k)$	$\tilde{O}(m+k)$	$\tilde{O}(m(m+k))$
MCF	[6, 5]	$\tilde{O}(m \cdot L)$	$\tilde{O}(k \cdot L)$	$\tilde{O}(k)$	$\tilde{O}(m^3 \cdot k \cdot L)$
MCF	[7, 8]	$\tilde{O}(m \cdot L)$	$\tilde{O}(m \cdot k \cdot L)$	$\tilde{O}(m \cdot L)$	$\tilde{O}(m^2 \cdot L)$
MCF & PFM	[this paper]	$\tilde{O}(L^3)$	$\tilde{O}(k \cdot L^3)$	$\tilde{O}(k \cdot L)$	$\tilde{O}(m \cdot k \cdot L^3)$

Figure 1: Our result vs. existing work for the Multicommodity flow (MCF) and Passive monitoring (PFM) problems in ROUTERS model. $\tilde{O}(\cdot)$ absorbs factors polynomial in $\frac{\log(mkCD)}{\epsilon}$. L denotes the maximal path length; $L \ll m$ in many practical networks.

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} c_S x_S \\ \text{s.t.} \quad & \sum_{(u,S) \in \mathcal{R}} x_S \geq 1 \quad \forall u \in \mathcal{U} \\ & x_S \geq 0 \quad \forall S \in \mathcal{S} \end{aligned} \tag{3}$$

Its dual "packing" linear program is given below.

$$\begin{aligned} \max \quad & \sum_{u \in \mathcal{U}} y_u \\ \text{s.t.} \quad & \sum_{(u,S) \in \mathcal{R}} y_u \leq c_S \quad \forall S \in \mathcal{S} \\ & y_u \geq 0 \quad \forall u \in \mathcal{U} \end{aligned} \tag{4}$$

The algorithmic framework is given in Figure 2. We use the following notations: $|S|$ is the number of elements covered by S, $N = |\mathcal{U}|$ is the number of clients to be covered, and M is the total number of servers. By scaling, we assume that $\min_S c_S = 1$ and $\max_S c_S = C$.

The algorithm starts by setting all x_S to zero. We then increase the values x_S according to certain rules till each client u_0 is covered: $\sum_{(u_0,S) \in \mathcal{R}} x_S \geq 1$. We associate a "residual requirement" $r_u > 0$ with each client $u \in \mathcal{U}$. At any time, the residual requirement of $u \in \mathcal{U}$ is given by

$$r_u = \left(\mathcal{B}^{1/\epsilon}\right)^{-\sum_{(u,S) \in \mathcal{R}} x_S} \tag{5}$$

where $\mathcal{B} = NC/\epsilon$.

We increase x_S values while always satisfying two important rules: "increasing least pricey servers" and "step-size constraint" as given below.

- **Increasing least pricey servers**. Define the *price* $\alpha(S)$ of a server S as the ratio of its cost to the sum of current residual requirements of the clients served by that server:

$$\alpha(S) := \frac{c_S}{\sum_{(u,S) \in \mathcal{R}} r_u}. \tag{6}$$

  Call a server $(1+\epsilon)$-least pricey if its price is at most $(1+\epsilon)$ times that of the least pricey server:

$$\alpha(S) \leq (1+\epsilon)\alpha \quad \text{where} \quad \alpha = \min_{S'} \alpha(S'). \tag{7}$$

  According to the first rule, at any point in the algorithm, only $(1+\epsilon)$-least pricey servers S are allowed to increase their x_S values.

- **Step-size constraint**. Let Δx_S be the increase in the x_S value of a server S in a step. The second rule requires that the total increase corresponding to servers serving any fixed client u, such that $\sum_{(u,S) \in \mathcal{R}} x_S \leq 1+\epsilon$ currently, is at most $\epsilon^2 / \log \mathcal{B}$:

$$\sum_{(u,S) \in \mathcal{R}} \Delta x_S \leq \frac{\epsilon^2}{\log \mathcal{B}}.$$

The dual variables y_u are also increased as given in Step 3c. Here Δr_u is the decrease in the requirement of u in this step and $\alpha = \min_S \alpha(S)$ as defined in (7). In the end, the algorithm outputs the final x_S and y_u values to be the fractional solutions. The main result of this section is as follows.

LEMMA 2.1. *Assuming that the algorithm in Figure 2 terminates, let its outputs be $\{x_S\}$ and $\{y_u\}$. Then $\{x_S\}$ forms a $(1+O(\epsilon))$-approximate feasible solution to (3). Also $\{y_u\}$ forms a $(1+O(\epsilon))$-approximate $(1+O(\epsilon))$-feasible solution (i.e., $\sum_{(u,S) \in \mathcal{R}} y_u \leq (1+O(\epsilon))c_S$ for all S) to (4).*

Before proving Lemma 2.1, we prove some important properties of the above algorithm.

LEMMA 2.2. *During the algorithm, we have $\alpha = \min_S \alpha(S) \leq \frac{C}{\mathcal{B}^{-1/\epsilon}}$ where $\alpha(S)$ is the price of S as defined in (6).*

PROOF. Let u be the last client that got covered completely. Thus during the algorithm, $\sum_{(u,S) \in \mathcal{R}} x_S < 1$ and $r_u > \mathcal{B}^{-1/\epsilon}$. For any server S such that $(u,S) \in \mathcal{R}$, since $c_S \leq C$, it is easy to see that $\alpha(S) \leq C/\mathcal{B}^{-1/\epsilon}$. This completes the proof. □

Call a client u "active" in a particular step if $\sum_{(u,S) \in \mathcal{R}} x_S \leq 1+\epsilon$. We denote the set of active clients by $\mathcal{A}$.

LEMMA 2.3. *Once a client stops being active, the further increase in y_u till the end of the algorithm is less than $\epsilon^2 / N \log \mathcal{B} < \epsilon/N$.*

PROOF. The increase in y_u is given by $\Delta y_u = \alpha \cdot \Delta r_u \cdot \epsilon / \log \mathcal{B}$ (see Step 3c in Figure 2). Since, from Lemma 2.2, $\alpha \leq C/\mathcal{B}^{-1/\epsilon}$ while the further decrease in r_u is $\Delta r_u \leq r_u < \mathcal{B}^{-1-1/\epsilon}$, the proof is complete. □

Now we prove Lemma 2.1. Assume that the algorithm in Figure 2 terminates. It is clear from the stopping condition that $\{x_S\}$ forms a feasible solution to the primal. To complete the proof we show that the primal value is not much larger than the dual, i.e., $\sum_S c_S x_S \leq (1+O(\epsilon)) \sum_u y_u$ and that $\{y_u\}$ satisfies $\sum_{(u,S) \in \mathcal{R}} y_u \leq (1+O(\epsilon))c_S$ for all S. This, in turn, implies near-optimality of the two solutions.

1. Let $\mathcal{B} = NC/\epsilon$.
2. Initialize $x_S \leftarrow 0$ for $S \in \mathcal{S}$ and $y_u \leftarrow 0$ for $u \in \mathcal{U}$.
3. **While** $\exists\ u_0 \in \mathcal{U}$ such that $\sum_{(u_0,S)\in\mathcal{R}} x_S < 1$ do:
   (a) Update
   $$r_u = \left(\mathcal{B}^{1/\epsilon}\right)^{-\sum_{(u,S)\in\mathcal{R}} x_S} \quad \text{for } u \in \mathcal{U}, \quad \alpha(S) := \frac{c_S}{\sum_{(u,S)\in\mathcal{R}} r_u} \text{ for } S \in \mathcal{S}, \quad \alpha = \min_{S\in\mathcal{S}} \alpha(S).$$
   (b) Increase x_S values by Δx_S satisfying:
   - "increasing least pricey servers" rule:
   $$\Delta x_S > 0 \quad \text{only if} \quad \alpha(S) \le (1+\epsilon)\alpha.$$
   - "step-size constraint": $\forall u$ such that $\sum_{(u,S)\in\mathcal{R}} x_S \le 1+\epsilon$, we have
   $$\sum_{(u,S)\in\mathcal{R}} \Delta x_S \le \frac{\epsilon^2}{\log \mathcal{B}}.$$
   (c) For all $u \in \mathcal{U}$ do: Let $\Delta r_u \leftarrow r_u \left(1 - \mathcal{B}^{-\sum_{(u,S)\in\mathcal{R}} \Delta x_S/\epsilon}\right)$ be the reduction in r_u in this step and update
   $$y_u \leftarrow y_u + \alpha \cdot \Delta r_u \cdot \frac{\epsilon}{\log \mathcal{B}}.$$
4. Output x_S for all S and y_u for all u.

Figure 2: An algorithmic framework for the set-cover problem and its dual

Now note that the change in the requirement of a client $u \in \mathcal{A}$ in a step satisfies:

$$\begin{aligned} \Delta r_u &= r_u \left(1 - \mathcal{B}^{-\sum_{(u,S)\in\mathcal{R}} \Delta x_S/\epsilon}\right) \\ &\ge (1-\epsilon) r_u \cdot \frac{\log \mathcal{B}}{\epsilon} \cdot \sum_{(u,S)\in\mathcal{R}} \Delta x_S \end{aligned} \tag{8}$$

where Δx_S is the increase in the x_S values of a server S. This follows from the step-size constraint (which is satisfied for active clients $\mathcal{A}$) and the elementary fact that $\exp(-\delta) \le 1 - \delta(1-\delta)$ for small $\delta > 0$. Now from (8), we conclude the following. The total change in $\sum_{u\in\mathcal{A}} y_u$, in a single step is

$$\begin{aligned} \sum_{u\in\mathcal{A}} \Delta y_u &= \alpha \cdot \frac{\epsilon}{\log \mathcal{B}} \cdot \sum_{u\in\mathcal{A}} \Delta r_u \\ &\ge \alpha(1-\epsilon) \cdot \sum_{u\in\mathcal{A}} r_u \sum_{(u,S)\in\mathcal{R}} \Delta x_S \\ &= \alpha(1-\epsilon) \cdot \sum_{S} \Delta x_S \sum_{u\in\mathcal{A}:(u,S)\in\mathcal{R}} r_u \\ &\ge (1-O(\epsilon)) \sum_S c_S \Delta x_S. \end{aligned} \tag{9}$$

The last inequality follows from two observations: (i) "increasing least pricey server" rule implies that $\Delta x_S > 0$ only for the servers S with $(1+\epsilon)\alpha \ge \frac{c_S}{\sum_{(u,S)\in\mathcal{R}} r_u}$, and (ii) $\alpha \sum_{u\notin\mathcal{A}:(u,S)\in\mathcal{R}} r_u < \epsilon \le \epsilon c_S$. Summing (9) over all the rounds in the algorithm, we get that the outputs satisfy $\sum_u y_u \ge (1-O(\epsilon)) \sum_S c_S x_S$.

It now remains to prove that the final solution $\{y_u\}$ forms an approximately feasible solution to the dual program, i.e., $\sum_{(u,S)\in\mathcal{R}} y_u \le (1+O(\epsilon)) c_S$ for all S. To this end, fix a server S and observe that, in any step, we have

$$\begin{aligned} \sum_{u\in\mathcal{A}:(u,S)\in\mathcal{R}} \Delta y_u &= \alpha \sum_{u\in\mathcal{A}:(u,S)\in\mathcal{R}} \Delta r_u \cdot \frac{\epsilon}{\log \mathcal{B}} \\ &\le \frac{\epsilon}{\log \mathcal{B}} \cdot c_S \cdot \frac{\sum_{u\in\mathcal{A}:(u,S)\in\mathcal{R}} \Delta r_u}{\sum_{(u,S)\in\mathcal{R}} r_u} \\ &\le \frac{\epsilon}{\log \mathcal{B}} \cdot c_S \cdot \frac{\sum_{u\in\mathcal{A}:(u,S)\in\mathcal{R}} \Delta r_u}{\sum_{u\in\mathcal{A}:(u,S)\in\mathcal{R}} r_u} \end{aligned} \tag{10}$$

The first inequality holds since $\alpha \le c_S / \sum_{(u,S)\in\mathcal{R}} r_u$.

From Lemma 2.3, we know that the increase in y_u after u stops being active is at most ϵ/N. Thus the total contribution to $\sum_{(u,S)\in\mathcal{R}} y_u$ from all clients u after they become non-active is at most $\epsilon \le \epsilon c_S$. Thus we can ignore the clients u once they stop being active. Now $\sum_{u\in\mathcal{A}:(u,S)\in\mathcal{R}} r_u$ reduces from $|S|$ to at least $\mathcal{B}^{-1-1/\epsilon}$. Using (10), we now sum $\sum_{(u\in\mathcal{A}:u,S)\in\mathcal{R}} \Delta y_u$ over all rounds. The right-hand-side of the sum is at most

$$\begin{aligned} &(1+O(\epsilon)) \frac{\epsilon}{\log \mathcal{B}} \cdot c_S \cdot \int_{x=\mathcal{B}^{-1-1/\epsilon}}^{x=|S|} \frac{dx}{x} \\ &= (1+O(\epsilon)) \frac{\epsilon}{\log \mathcal{B}} \cdot c_S \cdot \log \frac{|S|}{\mathcal{B}^{-1-1/\epsilon}} \end{aligned}$$

where $x = \sum_{(u,S)\in\mathcal{R}} r_u$. This follows from the fact that for

any $u \in \mathcal{A}$, we have $\Delta r_u \leq \epsilon r_u$. Simplifying the expression, we get

$$\begin{aligned}(1+O(\epsilon))\frac{\epsilon}{\log \mathcal{B}} \cdot c_S \cdot \log(|S|\mathcal{B}^{1+1/\epsilon}) \\ \leq \quad (1+O(\epsilon))\frac{\epsilon}{\log \mathcal{B}} \cdot c_S \cdot \log(\mathcal{B}^{2+1/\epsilon}) \\ \leq \quad (1+O(\epsilon))(1+2\epsilon) \cdot c_S.\end{aligned}$$

Thus we conclude that $\sum_{(u,S)\in\mathcal{R}} y_u$ is at most $(1+O(\epsilon))c_S$, as desired. This completes the proof of Lemma 2.1.

2.2 The Distributed Set-cover algorithm

In this section, we present an algorithm for computing a $(1+\epsilon)$ approximation to the cover problem in polylogarithmic running time. Our algorithm is an instance of the algorithmic framework presented in Section 2.1.

- Let $\mathcal{B} = NC/\epsilon$.
- $\alpha \leftarrow \min_S \frac{c_S}{|S|}$, $\beta \leftarrow \frac{\epsilon^2}{(1+2\epsilon)\log \mathcal{B}}$, $\delta \leftarrow \frac{\epsilon^3}{MC \log \mathcal{B}}$.

Figure 3: Initialization of distributed algorithms for clients and servers for the cover problem and its dual

- **Repeat** for $T = O\left(\frac{\log \mathcal{B}}{\epsilon^2}\right)$ phases
  1. **Repeat** for $T_{phase} = O\left(\frac{1}{\beta} \cdot \log \frac{\epsilon}{\delta \log \mathcal{B}}\right)$ steps:
     Update r_u and y_u as given in Figure 2.
  2. $\alpha \leftarrow \alpha(1+\epsilon)$.

Figure 4: Distributed algorithm for client u

- **Repeat** for $T \cdot T_{phase}$ steps:
  1. **If** $\alpha(S) \leq (1+\epsilon)\alpha$
     Then $x_S \leftarrow \max\{x_S(1+\beta), \delta\}$.

Figure 5: Distributed algorithm for server S

The distributed algorithm is given in Figures 3-5. The algorithm is initialized by setting all x_S to zero. However, we allow an additive increase by δ the first time. Note that the total cost of this initial assignment is only an ϵ-fraction of the optimum. We associate a "residual requirement" $r_u > 0$ with each client $u \in \mathcal{U}$ as before given in (5).

We prove that the above algorithm is, in fact, an instance of the framework in Figure 2. As shown in Lemma 2.5, the algorithm always maintains a lower bound α on $\alpha(S) = c_S/\sum_{(u,S)\in\mathcal{R}} r_u$ of any server S. It then iteratively increases the x_S value of *all* $(1+\epsilon)$-least-pricy servers in parallel and updates the residual requirements of the clients. The main result of this section is summarized below.

LEMMA 2.4. *The client-server algorithms in Figures 3-5 compute an $(1+\epsilon)$-approximate solution to the linear program (3) in the semi-cooperative distributed framework in $O(\frac{\log^2 \mathcal{B}}{\epsilon^4} \cdot \log \frac{MC}{\epsilon}))$ parallel steps.*

2.3 Proof of Lemma 2.4

LEMMA 2.5 (α-INVARIANT). *Throughout the algorithm, we have $\alpha \leq \min_S \alpha(S)$ where $\alpha(S) = \frac{c_S}{\sum_{(u,S)\in\mathcal{R}} r_u}$.*

PROOF. We prove this by induction on the number of phases completed. Since initially $r_u = 1$ for all $u \in \mathcal{U}$ and $\alpha = \min_S c_S/|S|$, this invariant is satisfied. Since the values r_u do not increase during the algorithm; and α remains constant during a phase, it remains to be a lower bound. Now we show that this invariant continues to hold when we increase the value of α by a factor of $(1+\epsilon)$ at the end of a phase. To argue this, it is enough to prove that no server satisfies $\frac{c_S}{\sum_{(u,S)\in\mathcal{R}} r_u} < \alpha(1+\epsilon)$ at the end of a phase, just before increasing α by a factor $(1+\epsilon)$.

A phase lasts $T_{phase} = O\left(\frac{1}{\beta} \cdot \log \frac{\epsilon}{\delta \log \mathcal{B}}\right)$ steps. Consider a server S such that

$$c_S \leq \alpha(1+\epsilon) \sum_{(u,S)\in\mathcal{R}} r_u \tag{11}$$

holds in the beginning of a phase. By induction hypothesis, we know that $\alpha \sum_{(u,S)\in\mathcal{R}} r_u \leq c_S$ holds. Now this server increases its x_S value by a multiplicative factor of $(1+\beta)$ in each step as long as (11) continues to hold. Assume, in order to get a contradiction, that this continues to holds at the end of this phase. Since the initial x_S value is at least δ, after T_{phase} steps, the overall increase in x_S is at least $\epsilon/\log \mathcal{B}$. Thus all the r_u values such that $(u,S) \in \mathcal{R}$ decrease by a factor of at least $(1+\epsilon)$ during this phase. This, in turn, contradicts the induction hypothesis. Hence the proof is complete. □

We now prove Lemma 2.4. First of all, it is clear that the algorithm in Figures 3-5 takes the given number of steps. Now to show that it actually computes near-optimal solutions, it is enough to argue that this algorithm fits in the framework given in Figure 2. We already know, from Lemma 2.5, that the value of α in this algorithm is a lower bound on $\min_S \alpha(S)$. Thus we indeed increase the x_S value for only $(1+\epsilon)$-least pricey servers, satisfying the increasing-least-pricey-servers rule.

We next show that the algorithm runs while there exists a client u such that $\sum_{(u,S)\in\mathcal{R}} x_S < 1$. The initial value of α is $\min_S c_S/|S|$. The algorithm runs for $T = O(\log \mathcal{B}/\epsilon^2)$ phases and in each phase, increases α by a factor of $(1+\epsilon)$. Thus the final value of α is $(\mathcal{B}^{1/\epsilon})^{\theta(1)}$. Since α is a lower bound on $\alpha(S)$ for any S and $1 \leq c_S \leq C$, we get that $\sum_{(u,S)\in\mathcal{R}} r_u \leq (\mathcal{B}^{-1/\epsilon})^{\theta(1)}$. From the definition of r_u, it follows that we can ensure, by setting constants appropriately, that by the end of the algorithm $\sum_{(u,S)\in\mathcal{R}} x_S \geq 1$ for each u.

To complete the proof, we now show that the step-size constraint is also satisfied. Consider a client u such that $\sum_{(u,S)\in\mathcal{R}} x_S \leq 1+\epsilon$. It is easy to see that the update in x_S values imply that the step-size constraint is satisfied for u:

$$\sum_{(u,S)\in\mathcal{R}} \Delta x_S \leq \beta \sum_{(u,S)\in\mathcal{R}} x_S + M\delta \leq \beta(1+\epsilon) + \beta\epsilon \leq \frac{\epsilon^2}{\log \mathcal{B}}.$$

3. DISTRIBUTED ALGORITHM FOR THE PASSIVE COMMODITY MONITORING

The passive commodity monitoring problem is an instance of the cover problem. Here the links e represent the servers with costs c_e. The paths between source-sink pairs are the clients. A link e is considered to cover a path p if it lies on the path: $e \in p$. We denote this by $(p, e) \in \mathcal{R}$ in our notation. Let L be an upper bound on the number of links allowed on any source-sink path that carries a positive flow. By scaling, we assume that $\min_e c_e = 1, \max_e c_e = C, \min_i d_i = 1$, and $\max_i d_i = D$. Let n and m denote the total number of routers and links in the network. Thus $M = m$ is the number of servers and $N = n^L$ is (an upper bound on) the number of clients. The main result of this section is Theorem 1.2.

We now show how to adapt the cover algorithm described in Figures 3-5 for the passive commodity monitoring. Since the paths p between the ith source-sink pair need to be covered to an extent of d_i, we maintain the following residual-requirement invariant throughout the algorithm:

$$r_p = (\mathcal{B}^{1/\epsilon})^{-\left(\sum_{e \in p} x_e\right)/d_i} \tag{12}$$

where $\mathcal{B} = n^L C/\epsilon$. Since the denominator in the exponent above can be as large as D, we incur a term of $\log D$ in the number of steps T_{phase} in a phase. Thus T_{phase} is now set as

$$T_{phase} = O\left(\frac{1}{\beta} \log \frac{mCD}{\epsilon}\right).$$

This is needed to ensure that Lemma 2.5 holds.

Since we are aiming for a polynomial computational overhead per commodity or link agent, we cannot afford to maintain r_p values for each of the exponentially many paths explicitly. These values will be represented implicitly by maintaining the values of x_e for each server. In the algorithm (in Figures 3-5), the only place where the residual requirements get used is while computing the price $\alpha(e) = c_e / \sum_{(p,e) \in \mathcal{R}} r_p$ for a server e. In Section 3.1, we show that the denominator $\sum_{p: e \in p} r_p = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} r_p$ can be computed in polynomial time for any given link e with the local communication available to it in our distributed computation model. In our distributed algorithm, there are no explicit client agents; the server agents communicate with each other and compute the solutions.

The proof of Theorem 1.2 is very similar to that of Lemma 2.4 and is omitted. We discuss the message and space overheads in the following section.

3.1 Computing the aggregate requirement in polynomial time

The general idea is as follows. To compute $\sum_i \sum_{p \in \mathcal{P}_i : e \in p} r_p$, we compute $\sum_{p \in \mathcal{P}_i : e \in p} r_p$ for each commodity i separately. Note that $\sum_{p \in \mathcal{P}_i : e \in p} r_p$ is the total residual requirements of paths of commodity i that pass through e.

Each directed path $p \in \mathcal{P}_i$ from s_i to t_i going through $e = (u, v)$ can be decomposed into three segments, as $p = p_1 \cup \{e\} \cup p_2$ where p_1 is a path from s_i to u while p_2 is a path from v to t_i. Let $r_{e'} = (\mathcal{B}^{1/\epsilon})^{-x_{e'}/d_i}$ for each link e'.

1. initialize $\Pi_1^0(s_i) = 1$, $\Pi_2^0(t_i) = 1$, $\Pi_1^0(u) = 0$ for $u \neq s_i$, and $\Pi_2^0(v) = 0$ for $v \neq t_i$.
2. **for all** $1 \leq l \leq L-1$ **and for all** $u, v \in V$, compute values $\Pi_1^l(u)$, and $\Pi_2^l(v)$ using recurrences
   (a) $\Pi_1^l(u) = \sum_{(w,u) \in E} \Pi_1^{l-1}(w) \cdot r_{w,u}$
   (b) $\Pi_2^l(v) = \sum_{(v,w) \in E} r_{v,w} \cdot \Pi_2^{l-1}(w)$
3. $\forall (u, v) \in E$, compute
$$\sum_{l=1}^{L} \sum_{l_1=0}^{l-1} \Pi_1^{l_1}(u) \cdot r_{u,v} \cdot \Pi_2^{l-1-l_1}(v)$$

Figure 6: A polynomial algorithm based on dynamic programming for computing $\sum_{p \in \mathcal{P}_i : e \in p} r_p$

We observe that

$$r_p = \prod_{e' \in p} r_{e'} = \left(\prod_{e_1 \in p_1} r_{e_1}\right) \cdot r_e \cdot \left(\prod_{e_2 \in p_2} r_e\right) = r_{p_1} \cdot r_e \cdot r_{p_2}.$$

Since L is an upper-bound on the path length, we have

$$\begin{aligned} \sum_{p \in \mathcal{P}_i : e \in p} r_p &= \sum_{p_1 . e . p_2} r_{p_1} \cdot r_e \cdot r_{p_2} \\ &= \sum_{l=1}^{L} \sum_{l_1=0}^{l-1} \Pi_1^{l_1}(u) \cdot r_e \cdot \Pi_2^{l-1-l_1}(v). \end{aligned} \tag{13}$$

Here we let

$$\Pi_1^{l_1}(u) = \sum_{|p_1| = l_1} r_{p_1} \quad \text{and} \quad \Pi_2^{l_2}(v) = \sum_{|p_2| = l_2} r_{p_2}, \tag{14}$$

where the first sum is over paths p_1 of length l_1 from s_i to u and the second sum is over paths p_2 of length l_2 from v to t_i.

The dynamic-programming based algorithm for computing these quantities $\Pi_j^l(w)$ for $l \in [0, L-1], j \in [1, 2], w \in V$ is given in Figure 6. This algorithm is similar to computing the shortest-distance over a *semiring* in [17]. The correctness of the dynamic program in Figure 6 follows directly from (13) and (14).

Remark. Note that this dynamic program also adds r_p values for all *non-simple* s_i-t_i paths of length at most L that contain edge e. However since $\mathcal{P}_i$ contains such paths and we have f_p variables in the primal for such paths as well, the dynamic program computes the correct aggregate requirements.

Distributed implementation of the dynamic program Now, we explain how the dynamic program given in Figure 6 can be implemented in our distributed model where each router exchanges messages only with its neighbors in the network.

Fix a commodity i and an edge $e = (u, v)$. To simplify the presentation, we assume that the routers are associated with the vertices $w \in V$. This is without loss of generality, since we can associate an edge-router with each one of its

end-points. The implementation of the dynamic program is now simple and natural. The routers w first initialize the value of $\Pi_1^0(w)$ and $\Pi_2^0(w)$. In lth iteration ($1 \leq l \leq L-1$), each vertex w computes the value of $\Pi_1^l(w)$ and $\Pi_2^l(w)$ using the recurrence in Figure 6. To accomplish this, it first fetches the values $\Pi_1^{l-1}(v)$ and $\Pi_2^{l-1}(v)$ from all its neighbors v, computes $\Pi_1^l(w)$ and $\Pi_2^l(w)$, and in the next iteration, communicates these values to its neighbors. In the end of L iterations, it finally computes the desired expression using line 3 in Figure 6. It is easy to see that the total number of messages needed for this implementation is $O(m \cdot L^3)$. The space needed per router is $O(L)$ to store values $\Pi_\cdot^l(w)$ for $0 \leq l \leq L-1$. Thus over all commodities, the total message complexity per step is $O(m \cdot k \cdot L)$ and the space needed per router is $O(k \cdot L)$. Since there are $\tilde{O}\left(L^2\right)$ steps overall, the total message complexity is $O(m \cdot k \cdot L^3)$. Since each of the $\tilde{O}\left(L^2\right)$ steps in the algorithm need $O(L)$ distributed rounds in the implementation, the overall convergence time is $\tilde{O}\left(L^3\right)$.

4. DISTRIBUTED ALGORITHM FOR THE MAXIMUM MULTICOMMODITY FLOWS

The maximum throughput multicommodity flows problem is the dual of the passive commodity monitoring problem. Thus an approximate solution for the flow problem can be computed using the framework given in Figure 2. The clients here correspond to the flow paths $p \in \mathcal{P}_i$ for each commodity i and the dual variables f_p correspond to the flows on these paths.

The flow variables are initialized to zero. In each step, the flow on path p is updated as (see step 3c in Figure 2):

$$f_p \leftarrow f_p + \alpha \cdot \Delta r_p \cdot \frac{\epsilon}{\log \mathcal{B}}.$$

Since we are aiming for polynomial computational overhead, we cannot afford to maintain the f_p variables explicitly. We instead maintain the value $f_i(e) := \sum_{p \in \mathcal{P}_i : e \in p} f_p$ of the total flow of commodity i through edge e.

Initially we have $f_i(e) = 0$ for all e and i. Note that in a single step, the increase in $f_i(e)$ is given by

$$\Delta f_i(e) = \sum_{p \in \mathcal{P}_i : e \in p} \Delta f_p = \alpha \cdot \frac{\epsilon}{\log \mathcal{B}} \cdot \sum_{p \in \mathcal{P}_i : e \in p} \Delta r_p.$$

Here Δf_p denotes the increase in the flow on path p and Δr_p is the decrease in the residual requirement of the path p. Note however that the dynamic program in Figure 6 can be used to compute $\sum_{p \in \mathcal{P}_i : e \in p} r_p$ at any step. Thus, knowing the values of this quantity before and after the step, their difference gives $\sum_{p \in \mathcal{P}_i : e \in p} \Delta r_p$ at this step. Therefore we can maintain the value of $f_i(e)$ for each commodity i during the algorithm without doing any extra work than that needed by the passive commodity monitoring algorithm. The fact that this flow forms a near feasible and near optimal solution to the maximum throughput multicommodity flow problem follows from Lemma 2.1.

5. REFERENCES

[1] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *FOCS*, 2004.

[2] B. Awerbuch. Complexity of network synchronization. *J. of the ACM*, 32(4):804–823, Oct. 1985.

[3] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems. In *STOC*, 1987.

[4] B. Awerbuch and R. G. Gallager. Distributed BFS algorithms. In *FOCS*, 1985.

[5] B. Awerbuch and R. Khandekar. Greedy distributed optimization of multi-commodity flows. In *PODC*, 2007.

[6] B. Awerbuch, R. Khandekar, and S. Rao. Distributed algorithms for multicommodity flow problems via approximate steepest descent framework. In *SODA*, 2007.

[7] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *FOCS*, 1993.

[8] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multicommodity flow problem and local competitive routing in dynamic networks. In *STOC*, 1994.

[9] B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *FOCS*, 1990.

[10] B. Awerbuch and D. Peleg. Sparse partitions. In *FOCS*, 1990.

[11] S. Fischer, H. Racke, and B. Vocking. Fast convergence to wardrop equilibria by adaptive sampling methods. In *STOC*, 2006.

[12] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13:505–520, 2000.

[13] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. *Lecture Notes in Computer Science*, 2170:556+, 2001.

[14] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS*, 1998.

[15] M. Kaart et al. The importance of internet measurements for internet policy (extended abstract).

[16] M. Luby and N. Nissan. A parallel approximation algorithm for positive linear programming. In *STOC*, 1993.

[17] M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350, 2002.

[18] M. Murray and K. Claffy. Measuring the immeasurable: Global internet measurement infrastructure. In *PAM – A workshop on Passive and Active Measurements*, 2001.

[19] T. Roughgarden. The price of anarchy is independent of the network topology. In *STOC*, 2002.

[20] Y. Shavitt and E. Shir. Dimes: let the internet measure itself. *SIGCOMM Comput. Commun. Rev.*, 35(5):71–74, 2005.

[21] C. R. Simpson, Jr., and G. F. Riley. Neti@home: A distributed approach to collecting end-to-end network performance measurements.

[22] N. E. Young. Sequential and parallel algorithms for mixed packing and covering. In *FOCS*, 2001.