Improved Approximation Algorithms for the Multi-Commodity Flow Problem and Local Competitive Routing in Dynamic Networks

Baruch Awerbuch * Tom Leighton †

Abstract

In this paper, we describe a very simple bounded-queuesize local-control algorithm for routing multi-commodity flows in a dynamically-changing distributed network. The algorithm is based on the edge-balancing approach described in [AL93], but has the added benefits of:

1. a much improved running time, and
2. working even in networks where edge capacities can vary in an unpredictable and unknown fashion.

In fact, the sequential running time of the algorithm is now comparable to (and, in some cases, better than) the time of the best previously known approximation algorithms for the multi-commodity flow problem in a fixed network [LMP+91]. The fact that the new algorithm works well in dynamically changing networks means that problems such as end-to-end communication and load balancing [AMS89, AGR92, AAMR93] can now be solved in a throughput-competitive fashion, even for the more complex case of multiple commodities.

*Johns Hopkins University, Baltimore, MD 21218-2694, and MIT Laboratory for Computer Science, Cambridge MA 02139. Email: baruch@blaze.cs.jhu.edu. Supported by Air Force Contract AFOSR F49620-92-J-0125, NSF contract CCR-9114440, DARPA Contract N00014-92-J-1799, and a special grant from IBM.

†Department of Mathematics and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. This research was supported in part by Air Force Contract AFOSR F49620-92-J-0125 and DARPA Contracts N00014-91-J-1698 and N00014-92-J-1799.

0

1 Introduction

The *multicommodity flow problem* consists of shipping several different commodities from their respective sources to their sinks through a common network so that the total flow going through each edge does not exceed its capacity. Associated with each commodity is a *demand*, which is the amount of that commodity that we wish to ship through the network. Given a multicommodity flow problem, we would like to know if there is a *feasible flow*, i.e. a way of shipping the commodities that satisfies the demands as well as the capacity constraints. More generally, we might also like to know the maximum value z such that at least z percent of each demand can be shipped without exceeding the capacity constraints. The latter problem is known as the *concurrent flow problem*, and is equivalent to the problem of determining the minimum ratio by which the capacities must be increased in order to satisfy 100% of all of the demands.

Multicommodity flow problems arise in a wide variety of contexts and have been extensively studied during the past several decades. For example, many product distribution, traffic planning, and scheduling problems can be expressed and solved as a multicommodity flow problem. In addition, it has recently been discovered [LR88, KARR90] that a wide variety of NP-hard problems (such as graph partitioning, minimum feedback arc set, minimum cut linear arrangement, minimum 2D area layout, via minimization, and optimal matrix arrangement for nested disection) can be approximately solved using multicommodity flow algorithms. Many packet routing and communication problems can also be expressed as multicommodity flow problems but in such applications, it is crucial that the flow algorithms work in an online fashion with local control.

Not surprisingly, the prior literature on flow problems is extensive. Much of the past work centers on the much simpler problem of 1-commodity flow (also known as the *max-flow* problem). A survey of the many 1-commodity algorithms can be found in [GT90]. Most of these algorithms rely on finding augmenting paths to increase the flow from source to sink. An exception is the recent algorithm of Goldberg and Tarjan [GT90] (which is based on an algorithm of Karzanov [Kar74]). The latter algorithm maintains a preflow on the network and pushes local flow excess toward the sink along what is estimated to be a shortest path. The best of these algorithms run in $\tilde{O}(NM)$ steps, where N is the number of nodes in the network and M is the number of edges in the network.

There has been much less progress on the multicommodity flow problem, perhaps because handling K commodities seems to be much more difficult than handling one commodity. All exact algorithms for multicommodity flow are based on linear programming, all have horrendous running times (even though polynomial), and none are used for large networks in practice.

The situation is somewhat better for approximation algorithms, however. In particular, Vaidya [Vai89] developed a $(1+\epsilon)$-approximation algorithm for the min-cost multicommodity flow problem based on linear programming that uses (roughly) $O(K^{\frac{7}{2}}NM^{\frac{5}{2}}\log(DU\epsilon^{-1}))$ steps where K is the number of commodities, N and M are as before, D is the largest demand, and U is the largest edge capacity. Subsequently, Leighton et al [LMP+91] discovered a purely combinatorial $(1+\epsilon)$-approximation algorithm based on 1-commodity min-cost flows that runs in $O(K^2NM\epsilon^{-2}\log K\log^3 N)$ steps. (By using randomization, the running time of the latter algorithm can be improved by a factor of K).

Recently, Awerbuch and Leighton [AL93] proposed a much simpler approach to multicommodity flow that is based on a simple "edge balancing" technique that attempts to send a commodity across an edge $e=(u,v)$ if there is more of the commodity queued at u than there is queued at v. Contention for capacity is resolved by shipping the commodity which has the largest disparity in queue size across e. No attempt is made to find augmenting paths, shortest paths, min-cost paths, or even any path from a node to a sink. Commodities are simply entered at their respective sources, according to their demands, emptied from their sinks when present, and otherwise locally balanced across each edge.

In [AL93], it is shown that the simple edge-balancing approach to multicommodity flow leads to a $(1+\epsilon)$-approximation algorithm that runs in $O(M^3K^{5/2}L\epsilon^{-3}\log K)$ steps, where L is the length of the longest flow path. In this paper, we describe a closely related algorithm for multicommodity-flow that runs in $O\left(\frac{KL^2M}{\epsilon^3}\ln^3(M/\epsilon)\right)$ steps. The improved bound is competitive with (and in some cases, superior to) the best previous bound of $O(K^2NM\epsilon^{-2}\log K\log^3 N)$ for deterministic algorithms [LMP+91]. Indeed, if L is small, then the algorithm runs in nearly linear time (per commodity). The improved bound is obtained with a more careful amortized analysis of running time that is based on a partition of the flow for each commodity into packets of size equal to the demand for that commodity.

Perhaps more importantly, we also show that a slightly slower version of the algorithm can be used to route flow in an on-line local-control fashion through a distributed network in which the edge capacities are allowed to vary in an arbitrary and an unknown fashion. As long as there exists sufficient capacity to route the flow paths from the sources to the sinks at each step, we show that the on-line local-control algorithm will get the job done with bounded queue sizes and with near-optimal average delay.

Previously, it was not known whether or not it was possible to guarantee such performance even with *infinite* computational power, and *complete global knowledge* of the state of the system, such as link capacities and queue sizes. Indeed, it is not clear how to guarantee that we can route a packet of flow from its source to its destination in some bounded number of steps, if edge capacities near the packet can be zeroed out at any time (thereby blocking the forward progress of the packet and forcing it to choose an alternative path, which then may be become the blocked path). To reiterate, the success of our algorithm only depends on the existence of sufficient capacity for flow paths from the sources to the sinks. The flow paths can vary at each step so that it need not be the case that a particular flow paths exists for the amount of time needed for a packet to traverse it. Moreover, this "dynamic instantaneous" connectivity assumption may be further relaxed to an "amortized connectivity" assumption where the required connectivity between sources and sinks is assumed to exist if we amortize link capacity over some time window (which means that at any specific instant, sources and sinks may be completely disconnected). Nevertheless, we still show that our (nearly-trivial) algorithm is powerful enough to guarantee that all but a bounded number of packets reach their destinations quickly.

As a consequence, our method significantly extends the previous work on end-to-end routing in communication networks with dynamic topology, by exhibiting an $(1+\epsilon)$-throughput-competitive end-to-end protocol for the case of multiple senders and receivers. For the single-receiver case, this problem has been extensively studied [AE83, AG88, AMS89, AG91, AGR92]; yet no upper bounds on throughput-competitiveness in this setting were known.

The remainder of this paper is divided into sections as follows. In Section 2, we describe the general form of the algorithm for directed networks with dynamically changing edge capacities. This algorithm runs in $O(LM\epsilon^{-2}(K+\ln(K\epsilon^{-1}))$ rounds where each round takes $O(KM\ln K)$ steps. Then in Section 3, we show how to reduce the number of rounds to $O(LM\epsilon^{-2}\ln(K\epsilon^{-1})$ for undirected static networks. In Section 4, we show to to decrease number of computation steps per round, by using an amortized "packet-based" analysis, and slightly modifying the algorithm. Finally, in Section 5, we mention applications of this work to distributed computing and extensions of the algorithm for total maximum throughput (as opposed to maximum concurrent flow).

Due to space limitations, we will only sketch the proofs of the most important results in this extended abstract. The full details will appear in the final draft of this paper.

2 Routing Flow through Dynamic Networks

2.1 The Flow Problem

In a typical instance of a multicommodity flow problem, we are given a network with N nodes, M edges, and K commodities. Each edge e has a capacity $c(e)$ and each commodity i has a demand d_i. Without much loss of generality, we can also assume that every commodity i has a single source and a single sink. The goal of the algorithm is to find flow paths from each source to each sink so that it is possible to continuously ship d_i units of commodity i from the i'th source to the i'th sink for $1 \leq i \leq K$ without violating capacity constraints of any edge. We will show how to find such paths in what follows provided that there exists feasible flow paths for the corresponding problem with demands $(1+3\epsilon)d_i$ for commodity i, where $\epsilon > 0$ is a parameter of our choosing.

More importantly, however, we will show how to route the flow (at at rate of $(1+\epsilon)d_i$ units of flow for commodity i during each round) through the network using local decision making when the capacities in the network vary with time, provided that during each step there exist flow paths with enough capacity to satisfy demand $(1+3\epsilon)d_i$ for commodity i. This latter problem is made even more challenging by the fact that a flow path need not last long enough for the flow to traverse it. (Each unit of flow can only traverse one edge during each round.) Hence, we can always keep some amount of flow from ever reaching its destination. Nevertheless, we show that the amount of flow ever present in the system remains bounded over time, no matter how capacities vary, which means that all but a bounded amount of flow reaches its destination in a bounded amount of time after it entered the source. It is perhaps somewhat surprising that this dynamic flow routing problem can be solved at all, much less with a local-control, online algorithm that has no knowledge of future edge capacities (or even the existing edge capacities or the queue sizes in other regions of the network).

2.2 Deriving a Sequential Algorithm for the Traditional Multi-commodity Flow Problem in Static Networks

Any stable algorithm for a continuous flow problem with demands $\{(1+\epsilon)d_i\}$ can be easily used to find a feasible solution to a static flow problem with demands $\{d_i\}$. We simply run the continuous algorithm until the amount of each commodity residing in the queues is at most an $\frac{\epsilon}{(1+\epsilon)}$ fraction of the total amount of that commodity that has been pumped into the network. The number of rounds R needed to reach this point depends on the upper bound on flow residing in queues.

After R rounds, we will have input $(1+\epsilon)d_iR$ units of commodity i into the network. At most ϵd_iR units of commodity i remain in the network, and so d_iR units of commodity i have been pumped through the network. Since the flow has been pumped through in R rounds, an average of d_i units of commodity i is shipped per round. Hence, we can obtain a solution to the static problem by taking the history of the continuous solution and averaging (i.e., dividing by R). The total time needed to find the static solution will then be R times the time needed to implement each round of the continuous algorithm.

2.3 The Dynamic Algorithm

In what follows, we will describe the algorithm for the continuous version of the problem in which $(1+\epsilon)d_i$ units of commodity i are pumped into the network at each step. For simplicity, we will assume that there is a queue for each commodity at the head and tail of each directed edge. We will also assume that there is a single edge leading from each source and a single edge leading to each sink. We will also associate a potential function with each queue. In particular, the potential of a queue of size q for commodity i is defined to be $\phi_i(q) = e^{\alpha_i \cdot q}$ where α_i will be defined to be $\alpha_i = \dfrac{\epsilon}{8Ld_i}$, where L is the length of the longest flow paths for any commodity in an optimal feasible solution. (We will later show how to guess the value for L.) Moreover, we will restrict the source queue for each commodity i to have size Q_i, where the value of Q_i will be specified later. (Up to a log factor, $Q_i = \alpha_i^{-1}$.) Excess flow (beyond Q_i) at the source will be stored in a special *overflow buffer.* The potential of an overflow buffer of size b will be $\sigma_i(b) = \phi_i^{'}(Q_i) \cdot b = \alpha_i \cdot b \cdot e^{\alpha_i \cdot Q_i}$. The algorithm proceeds in rounds, where each round consists of the following four phases.

Phase 1: *Add new flow to the sources.* In particular, add $(1+\epsilon)d_i$ units of flow to the overflow buffer at the the i'th source $(1 \leq i \leq K)$. Then move as much flow as possible from the overflow buffer to the source queue (subject to the maximum height constraint of Q_i for the source queue).

Phase 2: *Push flow across edges* so as to minimize the total potential of the queues in each edge, but respecting the capacity constraint of the edge in the current round.

Phase 3: *Remove flow from the sinks.* Zero out the queue for commodity i at the sink for i $(1 \leq i \leq K)$.

Phase 4: *Rebalance at nodes.* Reallocate each commodity within each node so that the queues for commodity i are all equal within each node $(1 \leq i \leq K)$.

2.4 The Analysis

The key to analyzing the performance of the algorithm is to show that no overflow buffer ever gets too large. Since no queue height for the i'th commodity can ever exceed Q_i (the limit on the source queue height), we will then be able to argue that the total amount of undelivered flow never gets too large. As a consequence, we will also then be able to get good bounds on the number of rounds that will be needed to find flow paths for the off-line, static multicommodity flow problem.

We will show that the overflow buffers stay bounded by showing that the overall potential of the system stays bounded. We will show that the overall potential of the system stays bounded by showing that

1. The potential increase during Phase 1 is not too large, and
2. if the potential is high, then the potential decrease during Phases 2-4 more than compensates for the potential increase during Phase 1.

Our analysis is similar to that in [AL93] except that we use an exponential potential function and a more careful argument that exploits the use of the overflow buffers and queue height constraints (which were not present in the [AL93] algorithm).

Our analysis will make use of the following simple facts about functions that have positive derivatives (such as our potential functions ϕ_i and σ_i):

Fact A: $\forall x$ and $\delta \geq 0, \phi(x+\delta) \leq \phi(x) + \delta\phi^{'}(x+\delta)$

Fact B: $\forall x$ and $\delta \geq 0, \phi(x+\delta) \geq \phi(x) + \delta\phi^{'}(x+\delta) - \delta^2\phi^{''}(x+\delta)$

Fact C: $\forall x$ and $\delta \geq 0, \phi(x+\delta) \leq \phi(x) + \delta\phi^{'}(x) + \delta^2\phi^{''}(x+\delta)$

2.4.1 Potential Increase in Phase 1

Let q_i and b_i denote the heights of the source queue and the overflow buffer for commodity i at the beginning of the round. Then if $q_i + b_i + (1+\epsilon)d_i \leq Q_i$, the increase in the potential for commodity i during Phase 1 will be

$$\begin{aligned}
&\phi_i(q_i + b_i + (1+\epsilon)d_i) - \phi_i(q_i) - \sigma_i(b_i) \\
&\quad \leq \quad (b_i + (1+\epsilon)d_i)\phi_i^{'}(q_i + b_i + (1+\epsilon)d_i) - b_i\phi_i^{'}(Q_i) \\
&\quad \leq \quad (1+\epsilon)d_i\phi_i^{'}(q_i + b_i + (1+\epsilon)d_i)
\end{aligned}$$

by Fact A and the fact that $\phi_i^{'}$ is an increasing function.

If $q_i + b_i + (1+\epsilon)d_i \geq Q_i$, then the increase in potential for commodity i during Phase 1 is

$$\begin{aligned}
&\phi_i(Q_i) + \sigma_i(q_i + b_i + (1+\epsilon)d_i - Q_i) - \phi_i(q_i) - \sigma_i(b_i) \\
&\quad \leq \quad (Q_i - q_i)\phi_i^{'}(Q_i) + (q_i + (1+\epsilon)d_i - Q_i)\phi_i^{'}(Q_i) \\
&\quad = \quad (1+\epsilon)d_i\phi_i^{'}(Q_i)
\end{aligned}$$

by Fact A. In either case, the increase in potential for commodity i during Phase 1 is at most

$$(1+\epsilon)d_i\phi_i^{'}(s_i) \tag{1}$$

where s_i is the height of the i'th source queue after Phase 1.

2.4.2 Potential Decrease in Phases 2 and 3

If we push g units of commodity i across an edge e in Phase 2, then the resulting decrease in the potential will be

$$\phi_i(q_{\text{tail}}) - \phi_i(q_{\text{tail}} - g) + \phi_i(q_{\text{head}}) - \phi_i(q_{\text{head}} + g)$$

where q_{tail} is the initial height of the queue at the tail of e and q_{head} is the initial height of the queue at the head of e. Using Facts B and C, this quantity is easily bounded below by

$$g\phi_i'(q_{\text{tail}}) - g^2\phi_i''(q_{\text{tail}}) - g\phi_i'(q_{\text{head}}) - g^2\phi_i''(q_{\text{head}} + g). \tag{2}$$

Note that this quantity can be negative. Also note that when we sum Equation 2 across a sequence (or path) of queues for which the queue at the head of one edge is initially equal in height to the queue at the tail of the next edge, then we get a cancellation of first derivative terms. (i.e., the negative $g\phi_i'(q_{\text{head}})$ term from one edge will cancel the positive $g\phi_i'(q_{\text{tail}})$ term from the next edge in the path.) This cancellation will prove to be very useful in our analysis.

For example, let $\mathcal{P}$ denote a path of length L from the source for commodity i to the sink for commodity i, and let q_i^* denote the maximum height of a queue for commodity i along this path. (Note that $q_i^* \geq s_i$.) Now consider what happens if wepush g units of flow across each edge from the node with queues of height q_i^* to the sink during Phase 2, and then we zero out the queue at the sink during phase 3. Because we have balanced all the queues for each commodity within each node during the previous Phase 4, we can use Equation 2 to conclude that the resulting drop in potential will be at least

$$g\phi_i'(q_i^*) - g\phi_i'(0) - 2Lg^2\phi_i''(q_i^* + g). \tag{3}$$

Next consider a feasible flow $\mathcal{F}$ for which we ship $f_i = (1+2\epsilon)d_i$ units of commodity i for each i. Partition the flow for each commodity i into elementary flow paths $\mathcal{P}_{i,j}$ so that the flow along path $\mathcal{P}_{i,j}$ is $g_{i,j}$. Let $q_{i,j}^*$ be the maximum queuesize for commodity i along $\mathcal{P}_{i,j}$ and let $\mathcal{P}_{i,j}^*$ be that portion of $\mathcal{P}_{i,j}$ from the node with height $q_{i,j}^*$ to the sink for commodity i. Also define $q_i^* = \max q_{i,j}^*$.

Now consider what happens if we push $g_i(e)$ units of commodity i across each edge e during phase 2 where

$$g_i(e) = \sum^{j|e\in\mathcal{P}_{i,j}^*} g_{i,j}.$$

By Equation 2, the resulting drop in potential during phases 2 and 3 will then be at least

$$\begin{aligned}
&\sum_{i=1}^{K}\sum^{e} g_i(e)\left[\phi_i'(q_{\text{tail}}(e)) - \phi_i'(q_{\text{head}}(e))\right] \\
&\quad - g_i(e)^2\left[\phi_i''(q_{\text{tail}}(e)) + \phi_i''(q_{\text{head}}(e) + g_i(e))\right] \\
\geq &\sum_{i=1}^{K}\sum^{e} g_i(e)\left[\phi_i'(q_{\text{tail}}(e)) - \phi_i'(q_{\text{head}}(e))\right] \\
&\quad - g_i(e)f_i(e)\left[\phi_i''(q_{\text{tail}}(e)) + \phi_i''(q_{\text{head}}(e) + g_i(e))\right] \\
= &\sum_{i=1}^{K}\sum^{j}\sum^{e\in\mathcal{P}_{i,j}^*} g_{i,j}\left[\phi_i'(q_{\text{tail}}(e)) - \phi_i'(q_{\text{head}}(e))\right] \\
&\quad - g_{i,j}2f_i\phi_i''(q_{i,j}^* + f_i)
\end{aligned}$$

$$= \sum_{i=1}^{K}\sum^{j} g_{i,j}\left[\phi_i'(q_{i,j}^*) - \phi_i'(0) - 2Lf_i\phi_i''(q_{i,j}^* + f_i)\right] \tag{4}$$

We next set α_i so that $2Lf_i\phi_i''(q_{i,j}^* + f_i)$ will be at most a small fraction of $\phi_i'(q_{i,j}^*)$ for any i and $q_{i,j}^*$. In particular, we set

$$\alpha_i = \frac{\epsilon}{8Ld_i} \tag{5}$$

Since then

$$\begin{aligned}
2Lf_i\phi_i''(q_{i,j}^* + f_i) &= 2L(1+2\epsilon)d_i\alpha_i^2 e^{\alpha_i(1+2\epsilon)d_i} e^{\alpha_i q_{i,j}^*} \\
&= 2L(1+2\epsilon)d_i\alpha_i^2 e^{\alpha_i(1+2\epsilon)d_i}\phi_i'(q_{i,j}^*) \\
&\leq \frac{\epsilon}{2}\phi_i'(q_{i,j}^*)
\end{aligned} \tag{6}$$

as desired. Then the drop in potential from Equation 4 will be at least

$$\begin{aligned}
&\sum_{i=1}^{K}\sum^{j} g_{i,j}\left[(1-\frac{\epsilon}{2})\phi_i'(q_{i,j}^*) - \phi_i'(0)\right] \\
&\quad\geq \sum_{i=1}^{K}\sum^{j} g_{i,j}\left[(1-\frac{\epsilon}{2})\phi_i'(s_i) - \alpha_i\right] \\
&\quad= \sum_{i=1}^{K} f_i\left[(1-\frac{\epsilon}{2})\phi_i'(s_i) - \alpha_i\right] \\
&\quad= \sum_{i=1}^{K}(1+2\epsilon)d_i\left[(1-\frac{\epsilon}{2})\phi_i'(s_i) - \alpha_i\right]
\end{aligned}$$

$$= (1 + \frac{3\epsilon}{2} - \epsilon^2)\sum_{i=1}^{K} d_i\phi_i'(s_i) - \frac{\epsilon K(1+2\epsilon)}{8L}. \tag{7}$$

Of course, we don't actually know the feasible solution $\mathcal{F}$, but as long as we push flow across each edge so as to maximize the potential drop locally for each edge during Phase 2, we sill be guaranteed to decrease the potential by at least as much as if we push $g_i(e)$ units of flow i for each i. Thus the potential drop during Phase 2 and 3 is at least as large as the amount given in Equation 7

2.4.3 Overall Change in Potential

Since the potential does not increase during Phase 4, we can combine Equations 1 and 7 to find that the overall potential decrease during the round is at least

$$\sum_{i=1}^{K}\left(\frac{\epsilon}{2}-\epsilon^2\right)d_i\phi_i'(s_i)-\frac{\epsilon K(1+2\epsilon)}{8L}. \qquad (8)$$

If $s_i = Q_i$ for some i, then the decrease in potential is at least

$$\begin{aligned}&\left(\tfrac{\epsilon}{2}-\epsilon^2\right)d_i\phi_i'(Q_i)-\tfrac{K(1+2\epsilon)\epsilon}{8L}\\ &=\left(\tfrac{\epsilon}{2}-\epsilon^2\right)d_i\alpha_i e^{\alpha_i Q_i}-\tfrac{K(1+2\epsilon)\epsilon}{8L}\\ &=\tfrac{\epsilon}{2}(1-2\epsilon)\tfrac{\epsilon}{8L}e^{\alpha_i Q_i}-\tfrac{K(1+2\epsilon)\epsilon}{8L}.\end{aligned}$$

This decrease is nonnegative if

$$Q_i = \frac{1}{\alpha_i}\ln\left(\frac{2K(1+2\epsilon)}{\epsilon(1-2\epsilon)}\right)=\Theta\left(\frac{Ld_i\ln(K/\epsilon)}{\epsilon}\right)$$

If $s_i < Q_i$ for all i, then there is no potential in the overflow buffers after Phase 1, which means that the overall potential in the system at the end of the round is at most

$$\begin{aligned}2M\sum_{i=1}^{K}\phi_i(Q_i) &= 2M\sum_{i=1}^{K}e^{\alpha_i Q_i}\\ &= 2MK\left(\frac{2K(1+2\epsilon)}{\epsilon(1-2\epsilon)}\right)\\ &= \frac{4MK^2(1+2\epsilon)}{\epsilon(1-2\epsilon)}\end{aligned}$$

By induction on the number of rounds, this means that the potential in the system at the end of every round is at most $2MKe^{\alpha_i Q_i}=\Theta\left(\frac{MK^2}{\epsilon}\right)$.

As a consequence, we also know that the maximum overflow buffer for commodity i never exceeds size

$$\frac{2MK}{\alpha_i}=\frac{16Ld_iMK}{\epsilon}.$$

Hence, the total amount of commodity i in the system at the end of any round is at most

$$\begin{aligned}&\frac{2MK}{\alpha_i}+\frac{2M\ln\left(\frac{2K^2(1+2\epsilon)}{\epsilon(1-2\epsilon)}\right)}{\alpha_i}\\ &=\frac{32Ld_iM\left(K+\ln\left(\frac{2K^2(1+2\epsilon)}{\epsilon(1-2\epsilon)}\right)\right)}{\epsilon}\\ &=O\left(\frac{d_iLM(K+\ln(K/\epsilon))}{\epsilon}\right)\end{aligned}$$

Since $(1+\epsilon)d_i$ units of commodity i are entered into the network during each round, this means that at most

$$O\left(\frac{LM(K+\ln(K/\epsilon))}{\epsilon}\right)$$

rounds worth of input flow for any commodity are present in the network at any time. Thus the amount of flow residing in the network will be at most an ϵ-fraction of the total amount of flow that has been pumped into the network (for every commodity) once we have run for a total of

$$R=O\left(\frac{LM(K+\ln(K/\epsilon))}{\epsilon^2}\right)$$

rounds. Hence, if we are interested in finding a flow in a static network, we can find the flow in R rounds, where each round involves $O(MK\log K)$ work.

3 Improved Bounds for Static Networks

In the special case when the edge capacities do not vary with time and the edges are undirected, we can improve upon the analysis in Section 2 by not placing any restriction on the queue height at the sources (thereby eliminating the need for overflow buffers) and by taking credit for pushing flow from a very large queue to the appropriate sink. We must be careful to first restrict the flow for each commodity i so that it only uses edges with capacity at least $\frac{\epsilon d_i}{M}$. (This can reduce the resulting flow by at most an ϵ amount.) Then, whenever there is a very large queue in the network, we know that it is feasible to push $\frac{\epsilon d_i}{M}$ of the commodity from this queue to its sink. If the max queue for commodity i has size Q_i^*, this results in a potential drop of at least

$$\frac{\epsilon d_i}{M}\phi_i'(Q_i^*)-\frac{\epsilon d_i}{M}\phi_i'(0)-2N\left(\frac{\epsilon d_i}{M}\right)^2\phi_i''\left(Q_i^*+\frac{\epsilon d_i}{M}\right), \qquad (9)$$

(where we have replaced L by N and g by $\frac{\epsilon d_i}{M}$ from Equation 3). By using the value of α_i in Equation 5 and the analysis from Section 2, this bound can be shown to be at least

$$\begin{aligned}&\frac{\epsilon d_i}{M}\left(1-\frac{\epsilon}{2}\right)\phi_i'(Q_i^*)-\frac{\epsilon d_i}{M}\phi_i'(0)\\ &=\frac{\epsilon^2}{8ML}\left(\left(1-\frac{\epsilon}{2}\right)e^{\alpha_i Q_i^*}-1\right) \qquad (10)\end{aligned}$$

We now consider two cases, depending on the height of the source queues. First, if

$$\sum_{i=1}^{K}\left(\frac{\epsilon}{2}-\epsilon^2\right)d_i\phi_i'(s_i)\geq\frac{K(1+2\epsilon)\epsilon}{8L},$$

then by Equation 8, we know that the potential does not increase during the round. If, on the other hand,

$$\sum_{i=1}^{K}\left(\frac{\epsilon}{2}-\epsilon^2\right)d_i\phi_i'(s_i)\leq\frac{K(1+2\epsilon)\epsilon}{8L},$$

then the increase in potential during Phase 1 is at most

$$\frac{1+\epsilon}{(\frac{\epsilon}{2}-\epsilon^2)}\frac{K(1+2\epsilon)\epsilon}{8L}=\frac{(1+\epsilon)(1+2\epsilon)K}{4(1-2\epsilon)L}$$

by Equation 1. Hence, we will not have an increase in potential during the round if

$$\frac{\epsilon^2}{8ML}\left(\left(1-\frac{\epsilon}{2}\right)e^{\alpha_i Q_i^*}-1\right)\geq\frac{(1+\epsilon)(1+2\epsilon)K}{4(1-2\epsilon)L},$$

which will hold if

$$Q_i^*=\frac{1}{\alpha_i}\ln\left(\frac{cKM}{\epsilon^2}\right)$$

where c is a constant close to 2. Hence, the potential will drop whenever a queue reaches height Q_i^*.

This means that the maximum potential in the system is at most

$$\sum_{i=1}^{K}2Me^{\alpha_i Q_i^*} \quad = \quad 2KM\left(\frac{cKM}{\epsilon^2}\right)=\frac{2cK^2M^2}{\epsilon^2}$$

Thus, the maximum amount of commodity i in the system at any time is $2Mq$ where q is such that

$$2Me^{\alpha_i q}\leq\frac{2cK^2M^2}{\epsilon^2}$$

which is satisfied when

$$q\leq\frac{1}{\alpha_i}\ln\left(\frac{cK^2M}{\epsilon^2}\right)\leq\frac{8Ld_i}{\epsilon}\ln\left(\frac{cK^2M}{\epsilon^2}\right).$$

Hence, the maximum amount of commodity i in the system at any time is

$$\frac{16LMd_i}{\epsilon}\ln\left(\frac{cK^2M}{\epsilon^2}\right).$$

This means that after

$$R=O\left(\frac{LM}{\epsilon^2}\ln\left(\frac{M}{\epsilon}\right)\right)$$

rounds, the flow residing in the system will be at most an ϵ-fraction of the flow that has been pumped through the network for that commodity. This represents a factor of K improvement over the comparable number of rounds for the directed dynamic network algorithm described in Section 2.

4 Improving the Performance by Partitioning Flow into Packets

In Sections 2 and 3, we proved upper bounds on the number of rounds R needed before the flow remaining in the network for any commodity will be an ϵ-fraction of the flow that has been pumped through the network for that commodity. At first glance, it appears that $\Theta(MK\log K)$ work is required to implement each round. This is because we need to push flow for up to K commodities for each edge in a way that maximizes the potential decrease across the edge. As a consequence, the algorithm requires a total of $\Theta(RMK\log K)$ work before the leftover flow is a small fraction of the total flow.

In what follows, we will briefly describe how to modify the algorithm so that it is possible to substantially decrease the total work performed to reach the same ratio of flow remaining to flow pushed. The basic idea is to portion the flow for each commodity i into packets of size $(1+\epsilon)d_i$. In addition, we will approximate the queuesize 8 at the tail of each by an integer number of packets p so that $q-2(1+\epsilon)d_i\leq p(1+\epsilon)d_i\leq q$, and we will approximate the queuesize at the head of each edge by an integer number of packets so that $q\leq p(1+\epsilon)d_i\leq q+2(1+\epsilon)d_i$ for use in deciding which packets to route across each edge.

During each round, we will add one packet at the source for each commodity during Phase 1. During Phase 2, we will route packets across edges according to the following protocol. We first find the commodity i for which

$$\phi_i'(p_{i,tail}(1+\epsilon)d_i)-\phi_i'(p_{i,head}(1+\epsilon)d_i)$$

is maximized and then we route a packet for this commodity across the edge (assuming that $p_{i,tail}\geq p_{i,head}+2$). If there is still excess capacity remaining, we update the queuesizes and select another packet by the same measure.

In the case when the capacity of the edge is smaller than the size of the packet, then we route as much of the packet as capacity allows during this round and we continue routing the same packet during the next several rounds until it is delivered, or until it becomes advantageous to interrupt the routing of the packet because some other commodity has become more desirable in the interim.

To keep computing costs to a minimum, we only update the queuesizes at either end of an edge when the final piece of each packet has been delivered or when the delivery of the packet has been interrupted. In the case that the delivery of a packet is interrupted we update the true queuesizes at either end of the edge but we do not update and approximate queuesize unless the true queuesize has changed by more than one packet's worth since the last time we updated the approximate queuesize. When the approximate queuesize is updated, it is made as close as possible to the true queuesize up to the constraints mentioned earlier.

Note that a packet can only be interrupted when the approximate queuesize at either end of the edge changes and that this can only happen when the true queuesize has changed by at least one packet's worth. hence we can change the cost of the interrupt to the packet's worth of flow that was moved into or out of the true queue at either end of the edge. (A similar argument is used to handle the balancing of packets at each node during Phase 4.)

We can then argue that the total work is bounded by the total number of times P that a full packet moves forward in the network multiplied by $O(\log K + \log N) = O(\log M)$ (since we have K queues at each edge and we will need to maintain priorities among them, and since we have up to N queues at each node). The value of P can be upper bounded by the total number of packets that enter the system in R rounds (this is easily seen to be KR) multiplied by the maximum path length traveled by any packet.

Since the "height" of a packet (in a queue) must decrease whenever a packet traverses an edge by at least one (when height is measured in terms of packets), it can be shown that no packet moves more than $\frac{Q_i}{d_i}$ times, where Q_i is the maximum queuesize ever encountered for commodity i. Since the maximum potential in the system (for undirected static networks) is $O(K^2M^2\epsilon^{-2})$, this means that the maximum queuesize for commodity i is at most Q_i where

$$e^{\alpha_i Q_i} \leq O(K^2M^2\epsilon^{-2}).$$

Hence,

$$Q_i = O\left(\frac{1}{\alpha_i}\ln(M/\epsilon)\right) = O\left(\frac{Ld_i}{\epsilon}\ln(M/\epsilon)\right).$$

Hence, no packet ever moves more than

$$O\left(\frac{L}{\epsilon}\ln(M/\epsilon)\right)$$

times and

$$P \leq O\left(\frac{KRL}{\epsilon}\ln(M/\epsilon)\right) .$$

Plugging in the value of R from Section 3 this means that we can find a feasible flow for an undirected static network in a total of

$$O\left(\frac{KL^2M}{\epsilon^3}\ln^3(M/\epsilon)\right)$$

steps. When L is small, this represents a substantial improvement over the deterministic

$$O\left(K^2NM\epsilon^{-2}\log K\log^3 N\right)$$

step algorithm described in [LMP+91].

Accounting for the rounding error. The preceding analysis assumed that we can approximate each queuesize for commodity i with a nearby integer multiple of $d_i(1+\epsilon)$ without affecting the bound on maximum potential. This is not quite true. In particular, whenever we introduce an error of $\Theta(d_i)$ in the queuesize, we need to compensate by subtracting an error term of

$$g\phi_i'(q+\Theta(d_i)) - g\phi_i'(q) \leq \Theta(gd_i)\cdot\phi_i''(q+\Theta(d_i))$$

into Equation 2. This, in turn, introduces an error term of

$$\Theta(Lgd_i)\cdot\phi_i''(q_i^*+g)$$

in Equation 3 and an error term of

$$\Theta(Sgd_i)\cdot\phi_i''(Q_i^*)$$

in Equation 9, where S is the length of the path from the large queue to its sink. The first error term is easily handled by decreasing α_i by a small constant factor, which does not seriously alter the analysis.

The second error term is more problematic, however, since S might be as large as N. In order for Equation 10 to still hold, we will thus have to replace L by N in the definition of α_i, which will mean that L is replaced by N throughout the remaining analysis and that final running time will be $\tilde{O}\left(KN^2M\right)$, instead of $\tilde{O}\left(KL^2M\right)$.

We can overcome this problem by artificially restricting packet movement so that no packet ever moves to a node that is more than distance L from its sink (where distance is measured using only edges with capacity at least $\epsilon d_i/M$). This can be accomplished by first running a preprocessing phase in which nodes are marked according to whether or not they are distance at most L (using large-enough capacity edges) from the respective sinks. This preprocessing requires $O(KLM)$ steps. Of course, we do not know the value of L ahead of time, but we can guess L by successively increasing powers of 2, until a good value is found. (The collective cost of guessing is at most factor of 2 in running time.)

Once we have restricted packets in this way, we know that $S \leq L$, and we can compensate for the error terms by making α_i to be smaller by a constant factor, which does not significantly affect the analysis.

We do need to be careful to check that the restriction that a packet cannot move to a node at distance $L+1$ from the respective sink does not otherwise alter the bound on the potential drop for Phases 2 and 3. This is easily checked, however, by observing that in the feasible flow, no flow is ever pushed to a node at distance $L+1$ from its sink. Hence, we can constrain our edge-balancing algorithm in the same way, without affecting the lower bound on potential drop due to edge-balancing.

This concludes the analysis of the $\tilde{O}\left(KL^2M\right)$-step algorithm for the multi-commodity flow problem. ∎

5 Final remarks

Implementing end-to-end routing in dynamic distributed networks. It is pretty obvious how to implement the algorithm in distributed networks. This requires sending a message over an edge for every flow packet that traverses an edge.

The specifications of the end-to-end communication problem require that every packet makes it to the destination. Even though some packets may be forever stuck in the network, standard error-correcting codes [MS78] or Rabin's information dispersal [Rab89] can be used to overcome this difficulty, as first suggested in [AGR92], without deterioration in amortized bit rate.

Applications to Min-cost flows. A modification of the gravitational approach of [AL93] and the current paper is considered in [KPP93], where the balancing algorithm is applied to the *derivatives* of queue heights rather than to the queue heights themselves. Incorporating edge costs into the potential function differences across the edges leads to local-control algorithms for the min-cost version of the multi-commodity flow problem.

Running time in practice. We feel that the "packet-based" algorithm in Section 4 will perform very well in practice. Indeed, the bottleneck of the previous implementations as well as in [AL93] was the number of steps required to precisely optimize potential drop on every edge. In the packet-based approach, exact optimization is replaced by computationally-easier approximation. Further, rounding queue size means that we do not need to waste time with pushes of small amounts of flow over edges.

Maximizing throughput. Extensions of the algorithm work for total maximum throughput problem (as opposed to maximum concurrent flow). This leads to somewhat less efficient algorithms. We are currently working on closing the gap between the two versions of the problem.

Acknowledgments

We thank Andrew Goldberg, Serge Plotkin, and Eva Tardos for stimulating discussions.

References

[AAMR93] William Aiello, Baruch Awerbuch, Bruce Maggs, and Satish Rao. Approximate load balancing on dynamic and synchronous networks. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 632–641, May 1993.

[AE83] Baruch Awerbuch and Shimon Even. A formal approach to a communication-network protocol; broadcast as a case study. Technical Report TR-459, Electrical Engineering Department, Technion-I.I.T., Haifa, December 1983.

[AG88] Yehuda Afek and Eli Gafni. End-to-end communication in unreliable networks. In *Proc. 7th ACM Symp. on Principles of Distrib. Computing*, pages 131–148. ACM SIGACT and SIGOPS, ACM, 1988.

[AG91] Yehuda Afek and Eli Gafni. Bootstrap network resynchronization. In *Proc. 10th ACM Symp. on Principles of Distrib.Computing*, pages 295–307, 1991.

[AGR92] Yehuda Afek, Eli Gafni, and Adi Rosen. Slide - a technique for communication in unreliable networks. In *Proc. 11th ACM Symp. on Principles of Distrib. Computing*, pages 35–46, August 1992.

[AL93] Baruch Awerbuch and Tom Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. 34rd IEEE Symp. on Found. of Comp. Science*, pages 459–46. IEEE, November 1993.

[AMS89] Baruch Awerbuch, Yishay Mansour, and Nir Shavit. End-to-end communication with polynomial overhead. In *Proc. 30th IEEE Symp. on Found. of Comp. Science*, pages 358–363, 1989.

[GT90] A.V. Goldberg and R. E. Tarjan. Solving minimum-cost flow problems by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[Kar74] A. V. Karzanov. Determining the maximal flow in a network by the method of preflows. *Soviet Math. Dokl.*, 15:434–437, 1974.

[KARR90] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *Proc. 31st IEEE Symp. on Found. of Comp. Science*, pages 726–727, 1990.

[KPP93] Anil Kamath, Omri Palmon, and Serge Plotkin. Simple and fast distributedmulticommondity flow algorithm. Unpublished manuscript, December 1993.

[LMP+91] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problem. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 101–111, May 1991.

[LR88] F. T. Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommod ity flow problems with applications to approximation algorithms. In *29th Annual Symposium on Foundations of Computer Science, IEEE*, pages 422–431, 1988.

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North Holland Publishing Company, Amsterdam, 1978.

[Rab89] M.O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *ACM*, 36(2):335–348, 1989.

[Vai89] P.M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proc. 30th IEEE Symp. on Found. of Comp. Science*, pages 332–337, 1989.