# Improved Bounds for the Token Distribution Problem

Kieran T. Herley*

TR 89-1051
October 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*The work of this author was supported in part by the Joint Services Electronics Program under contract F49620-87-C-0044.

# Improved Bounds for the Token Distribution Problem

Kieran T. Herley *
Department of Computer Science
Cornell University
Ithaca, NY14853

**Abstract**

The problem of packet routing on bounded degree networks is considered. An algorithm is presented that can route $n$ packets in $O(\log n + K)$ time on a particular $n$-node expander-based network provided that no more than $K$ packets share the same source or destination.

**Keywords:** parallel algorithms, routing, expander graphs

## 1 Introduction

The $(n, K_1, K_2)$-*routing problem*, involves routing $n$ packets in a network of processors subject to the constraint that no node is the source of more than $K_1$ packets, or the destination of more than $K_2$ packets. Peleg and Upfal [PU89] formulated this problem and developed a network and a routing algorithm with an optimal number of communication steps. In this note, an improved algorithm is presented that is optimal on a model in which both communication and local computation steps are counted.

Routing is a fundamental operation for computation on bounded degree networks. For example, the optimal simulations of the shared memory PRAM of [HB88] rely on a form of $(n, K_1, K_2)$-routing.

A Bounded Degree Network(BDN) is a synchronous collection of $n$ processors; each with its own private local memory, and each connected to a constant number of others by means of bidirectional channels. At each time step a processor may either perform a local operation, or send (receive) a single value to (from) one of its neighbours.

*The work of this author was supported in part by the Joint Services Electronics Program under contract F49620-87-C-0044.

In the BDN model the algorithm of [PU89] has a running time of $O(\log n + K_1 + K_2 + \log(K_1 + K_2)\log n)$. This note presents an interconnection and an $(n, K_1, K_2)$-routing algorithm with a running time of $O(\log n + K_1 + K_2)$, which is optimal according to a lower bound of [PU89].

When generalized to handle sets of size $N$ larger than $n$, our main result may be stated as follows.

**Theorem 1** *There is an $n$-node BDN and a deterministic algorithm that routes any instance of the $(N, K_1, K_2)$-routing problem in $O(\lceil N/n \rceil \log n + K_1 + K_2)$ time.*

In essence, the algorithm of [PU89] reduces an instance of the generalized routing problem to a partial permutation routing problem by means of *balancing* (referred to as *token distribution* in [PU89]). Balancing involves redistributing a set of packets initially scattered unevenly among the nodes of a BDN so that each node has the same number. Formally, a *distributed set* $\hat{S} \triangleq (S, f)$ is a set $S$ of packets distributed among the nodes of the BDN with packet $s \in S$ at node $f(s)$. The *profile* $\pi(\hat{S})$ of $\hat{S}$ is an $n$-tuple $(k_1, k_2, \ldots, k_n)$, where $k_i$ represents the number of packets of $S$ mapped to node $i$ by $f$. The *degree* $\Delta(\hat{S})$ of $\hat{S}$ is the maximum number of packets assigned by $f$ to any one node. The set $\hat{S}$ is *balanced* if it has degree $\lceil |S|/n \rceil$, and *approximately balanced* if it has degree $O(\lceil |S|/n \rceil)$. A distributed set $\hat{S} = (S, f)$ of packets, labelled with integer values, is said to be *balancesorted* if it is balanced and the packets with the $\lceil |S|/n \rceil$ smallest keys lie in node 1, the next $\lceil |S|/n \rceil$ next smallest lie in node 2 and so on.

**Lemma 1** *There exists an $n$-node BDN and a deterministic algorithm that balancesorts any distributed set $\hat{S} = (S, f)$ in $O(\lceil |S|/n \rceil \log n + \Delta(\hat{S}))$ time.*

The balancing of [PU89] is achieved in a sequence of *diffusions*, each of which reduces the degree of the distribution by some fixed factor $\gamma < 1$. Associated with each diffusion is a directed subnetwork of the BDN called a *flowdag*. During a diffusion, processors transmit packets only along the edges of the corresponding flowdag. In some cases the overhead in constructing these flowdags dominates the useful work required to physically move the packets. Our main idea is to decompose

the initial problem of balancing $n$ packets on an $n$-node BDN into some number $s$ essentially identical problems of balancing $n/s$ packets each on a subnetwork of size $n/s$. It is easier to construct the flowdags for the smaller problem within the stipulated time bounds.

In the next section the algorithm of [PU89] is sketched. In the subsequent section the modifications necessary to achieve the stated time bounds are described.

## 2 The Method of Peleg and Upfal

This section paraphrases the results of [PU89].

Let $G = (V, E)$ represent the structure of a bounded degree network. A directed acyclic subgraph $D = (V, E_D)$ of the graph $G$ is a *flowdag* for a subset $U \subseteq V$ if, for some $l$ depending only on $G$, (i) each $v \in V$ has indegree at most $l$ in $D$, and (ii) each $u \in U$ has outdegree at least $l + 1$ in $D$.

**Lemma 2 ([PU89])** *There is BDN $(V, E_{bal})$ such that for every $U \subseteq V$ of size at most $\beta|V|$ ($\beta$ a constant, independent of $|V|$), there is a flowdag $D_U$. Moreover there is a deterministic algorithm that allows each node in $U$ to determine its neighbours in $D_U$ in $O(\log |U|)$ time.*

Let $\hat{S} = (S, f)$ be a distributed set with profile $\pi(\hat{S}) = \mathbf{k} = (k_1, k_2, \ldots, k_n)$. Define $U(\hat{S}) \triangleq \{v_i \in V | k_i \geq (1/2)\Delta(\hat{S})\}$ and let $D(\hat{S})$ represent the flowdag $D_{U(\hat{S})}$. Let $Flow(\hat{S}, t)$ denote the operation of moving $t$ packets from $x$ to $y$ for each directed edge $(x, y)$ in $D(\hat{S})$.

**Lemma 3 ([PU89])** *There exist constants $\gamma < 1$ and $\rho < 1$ such that for all distributed sets $\hat{S}$ of size at most $n$, the degree of the distributed set after* $\mathrm{Flow}(\hat{S}, \rho\Delta(\hat{S}))$ *is at most $\gamma\Delta(\hat{S})$.*

Let an application of $Flow(\hat{S}, \rho\Delta(\hat{S}))$ be referred to as a *diffusion* step. The above lemma suggests the following straightforward balancing algorithm consisting of a sequence of *stages*. The $i^{th}$ stage is applied to the distributed set $\hat{S^{(i)}}$ resulting from the first $i - 1$ stages. During each stage a flowdag $D_i = D(\hat{S^{(i)}})$ is built, and then

a diffusion $Flow(\hat{S^{(i)}}, \rho\Delta(\hat{S^{(i)}}))$ is performed. The $i^{th}$ diffusion runs in $O(\gamma^i\Delta(\hat{S}))$, but the corresponding flowdag requires $O(\log n)$ time to build, so the algorithm spends $O(\sum \gamma^i\Delta(\hat{S})) = O(\Delta(\hat{S}))$ time actually moving packets during flowphases, but spends $O(\log \Delta(\hat{S}) \log n)$ time building flowdags. When balancing distributed sets of small degree the overhead due to flowdag construction dominates and so this algorithm does not meet the time bounds stipulated in Lemma 1.

The next section shows how this overhead can be reduced so that it is always subsumed by the work required for the movement of packets. A key element is the following lemma due to Peleg and Upfal that shows that it is possible to construct a sequence of subsets $U'_1, U'_2, \ldots$ efficiently that approximates the sequence $U(\hat{S^{(2)}}, U(\hat{S^{(2)}}), \ldots$ in the sense that the corresponding set of flowdags $D'_1, D'_2, \ldots$ can be used in place of $D_1, D_2, \ldots$ during balancing.

**Lemma 4 ([PU89])** *for each $\hat{S} = (S, f)$ there is a sequence of subsets of $V$ $U'_1, U'_2, \ldots, U'_H$, where (i) $U'_1 = U(\hat{S^{(1)}})$, (ii) $U_j \subseteq U'_j$ for each $1 \leq i \leq H$, (iii) $\gamma^H\Delta(\widehat{S}) = O(1)$, (iv) $|U'_j| \leq \beta|V|$ for $1 \leq i \leq H$, and (v) each $U'_{j+1}$ can be computed from $U'_j$ in $O(\log \log n)$ time.*

## 3 An Optimal Balancing Algorithm

The BDN underlying the algorithm presented here has the structure $(V, E_{exp} \bigcup E_{AKSL} \bigcup E_{tree})$. The nodes of $V$ are arranged in the pattern of a $N_1 \times N_2$ array where $N_1 \triangleq \log \log n$ and $N_2 \triangleq n/N_1$. All the rows of $V$ has the same structure, that of a square root expander[LPS86] with $N_2$ nodes. Similarly all the columns have the structure of a square-root expander on $N_1$ nodes. All of the edges in the row and column expanders together constitute the set $E_{exp}$. The edges of $E_{AKSL}$ give the nodes the structure of the $n$-node sorting network of [AKS83,Lei85] that can sort $n$ integers in $O(\log n)$ time; those of $E_{tree}$ are the edges of a complete binary tree with $n$ nodes.

First a balancesorting algorithm is outlined, the solution of the more general $(n, K_1, K_2)$-routing problem will follow from the results of [PU89].

Let $\hat{S} = (S, f)$ be a be a distributed set of size at most $n$. The overall structure of the algorithm is shown below.

**BalanceSort**

I Approximately balance the packets in each column.

II Approximately balance the packets in each row.

III Sort the packets.

Let $k_{i,j}$ represent the number of packets initially held by node $(i, j)$, and let $m_j$ denote the maximum number of packets held by any node in the $j^{th}$ column following Step I. After column balancing, $m_j$ can be at most $(A/N_1)\sum_{i=1}^{N_1} k_{i,j}$ for some constant $A$. Immediately prior to Step II no row can contain more than $\sum_{j=1}^{N_2} m_j = O(N_2)$ packets. Thus Step II involves balancing $O(N_2)$ packets in each row. The distribution $\mathbf{k}^{(i)} = (k_1^{(i)}, k_2^{(i)}, \ldots, k_{N_2}^{(i)})$ of packets row $i$ is *dominated* by $\mathbf{m} \triangleq (m_1, m_2, \ldots, m_{N_2})$, in the sense that $k_j^{(i)} \leq m_j$ for $1 \leq j \leq N_2$. In a sense Step I has decomposed the initial balancing problem into $N_1$ essentially identical subproblems.

Following the approximate balancing of each row during Step II each node holds at most $O(1)$ packets. A constant number of sorting steps suffices to complete the balancesorting in Step III.

The processors of the BDN can compute the value of $\Delta(\hat{S})$ in $O(\Delta(\hat{S}) + \log n)$ time. The straightforward balancing method mentioned in the previous section is used to perform Step I in $O(\log(\Delta(\hat{S}))\log N_1 + \Delta(\hat{S}))$ time. This quantity is bounded by $O(\Delta(\hat{S}) + \log n)$ since $N_1 = \log\log n$. Step III can be accomplished in $O(\log n)$ time, applying the sorting techniques of [AKS83,Lei85].

If $\Delta(\hat{S}) = \Omega(\log n \log\log n)$ then the same method as Step I is employed for Step II, otherwise the method outlined below is adopted. For the remainder of this section it will be assumed that $\Delta(\hat{S}) = o(\log n \log\log n)$.

The distribution $\mathbf{m}$ dominates the distribution of packets in each row after Step I. The key point is that same set of flowdags will suffice for all rows during Step II, and the overhead of constructing the flowdags may be shared among the rows.

Step II

1 Compute $m_j$ for each column $j$. Let $U'_1 = U(\mathbf{m})$.

2 Compute the sequence of sets $U'_1, U'_2, \ldots, U_H$ in the first row.

3 Distribute the $U'_j$ evenly among the rows.

4 For each $U'_j$ assigned to row $i$ compute the corresponding flowdag $D'_j$ in row $i$.

5 Let the processors within each column $j$ pool adjacency information so that each processor in the column knows the neighbours of the $j^{th}$ vertex in each of flowdags $D'_1, \ldots, D'_H$.

6 Execute the flowphases in sequence. (Do this concurrently in each row.)

Each column has a bounded degree tree of $O(\log N_1)$ depth as a subgraph, since each column expander has diameter $O(\log N_1)$, and so Substep 1 can easily be executed in $O(\Delta(\hat{S}) + \log N_1)$ time in each column using these column trees. Substep 2 involves the computation of sequence of $H$ sets, each of which can be computed from the previous one in the sequence in $O(\log\log N_1)$ time by Lemma 4. Since $\Delta(\hat{S}) = o(\log n \log\log)$ by assumption, $H$ must be at most $O(\log\log n) = O(N_1)$. In other words at most $O(1)$ distributions are assigned to each row during Substep 3. Thus Substeps 3 and 5 can be accomplished in $O(\log N_1)$ time using the column trees. The construction of the flowdags in Substep 4 can be completed in $O(\log n)$ time by Lemma 2. By Lemma 3, Substep 6 can be completed in $O(\Delta(\hat{S}))$ time. Summing the contributions of all these substeps, and simplifying, the total running time for Step II is seen to be $O(\log n + \Delta(\hat{S}))$.

This establishes Lemma 2 in the case where $|S| \leq n$. Sets of larger size can be handled by grouping packets into parcels of size $|S|/n$, approximately balancing the parcels (treating each parcel as a unit) with the method just described. Since there are $O(n)$ such parcels, and each flowstep requires $O(\lceil |S|/n \rceil)$ time, balancing can be accomplished in $O(\lceil |S|/n \rceil \log n + \Delta(\hat{S}))$ time. The individual packets can be sorted using the techniques of [AKS83,Lei85,BS78].

**Acknowledgement:** I would like to thank Gianfranco Bilardi for several helpful discussions in relation to this work.

## References


[AKS83] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(\log n)$ sorting network. In *Proceedings of the* $15^{th}$ *Annual Symposium on the Theory of Computing,* Boston, Massachusetts, pages 1–9, Apr 1983.

[BS78] G. Baudet and D. Stevenson. Optimal sorting algorithms for parallel computers. *IEEE Transactions on Computers*, c-27(1):84–87, Jan 1978.

[HB88] K. T. Herley and G. Bilardi. Deterministic simulations of PRAMs on bounded-degree networks. In *Proceedings of the* $26^{th}$ *Annual Allerton Conference on Communication, Control and Computation, Monticello, Illinois.*, Sept 1988.

[Lei85] F. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, c-34(4):344–354, April 1985.

[LPS86] A. Lubotzky, R. Phillips, and P. Sarnak. Explicit expanders and the Ramanujan conjectures. In *Proceedings of the* $18^{th}$ *Annual Symposium on the Theory of Computing,* Berkeley, California, pages 240–246, May 1986.

[PU89] D. Peleg and E. Upfal. The token distribution problem. *SIAM Journal on Computing*, 18(2):229–243, April 1989.