Selection, Routing, and Sorting on the Star Graph *

Sanguthevar Rajasekaran
Dept. of Comp. and Info. Sci. and Engg.
University of Florida, Gainesville, FL 32611
e-mail: raj@cise.ufl.edu

David S. L. Wei
School of Computer Science and Engineering
The University of Aizu, Fakushima, 965-80, Japan
e-mail: d-wei@u-aizu.ac.jp

Abstract

We consider the problems of selection, routing and sorting on an n-star graph (with $n!$ nodes), an interconnection network which has been proven to possess many special properties. We identify a tree like subgraph (which we call as a '$(k, 1, k)$ chain network') of the star graph which enables us to design efficient algorithms for the above mentioned problems.

We present an algorithm that performs a sequence of n prefix computations in $O(n^2)$ time. This algorithm is used as a subroutine in our other algorithms. We also show that sorting can be performed on the n-star graph in time $O(n^3)$ and that selection of a set of uniformly distributed n keys can be performed in $O(n^2)$ time with high probability. Finally, we also present a deterministic (non oblivious) routing algorithm that realizes any permutation in $O(n^3)$ steps on the n-star graph.

There exists an algorithm in the literature that can perform a single prefix computation in $O(n \lg n)$ time. The best known previous algorithm for sorting has a run time of $O(n^3 \lg n)$ and is deterministic. To our knowledge, the problem of selection has not been considered before on the star graph.

1 Introduction

Interconnection Networks (denoted as ICNs from hereon) have been generally accepted to be the most practical models of computing. Among those suggested ICNs, a binary n-cube is one of the most popular networks because it possesses some attractive features. The n-cube is a highly fault-tolerant ICN and has low degree and small diameter (which is logarithmic in the network size). The n-star graph has been suggested in [1] as a better alternative ICN to the n-cube. In [1], it has been shown that the star graph has better features than the n-cube with respect to the degree, diameter, etc. The network needs fewer links per node (processing element) and fewer communication steps per message passing request. A number of interesting algorithms have been designed for the star graph (see e.g., [1, 9, 2]). But still a lot more work has to be done.

In this paper, we consider the following problems: 1) Selection, 2) Sorting, and 3) Packet Routing. Sorting is the process of rearranging a given sequence of keys in either ascending or descending order. Packet routing is the problem of sending packets of information from their origins to their destinations. We are interested in *permutation routing* wherein at most one packet originates from any node in the ICN and at

*This research was supported in part by an NSF Research Initiation Award CCR-92-09260 and an NSF Grant CCR-9503007.

most one packet is destined for any node. Efficient sorting algorithms for various ICNs have already been developed [8, 17, 6].

Before our work, the best known sorting algorithm for the n-star graph ran in $O(n^3 \lg n)$ time [7, 2]. A different algorithm with the same run time has been given in [15]. Whereas [7, 2]'s algorithm is based on shearsort, [15]'s algorithm is based on bitonic sort and the underlying constant is also small (i.e., $\frac{1}{2}$). We present an improved sorting algorithm which runs on the n-star graph in $O(n^3)$ time with high probability. Our approach to randomized sorting differs from previous approaches in that we use repeated selection. These algorithms make use of prefix and selection algorithms that we have designed. Our selection algorithm can perform a set of n selections in $O(n^2)$ time with high probability provided the keys to be selected have ranks uniform in the interval $[1, n!]$. The prefix algorithm presented in this paper can compute the prefixes of n different sequences in $O(n^2)$ time. In contrast, Akl and Qiu [2] show that a single prefix computation can be performed in $O(n \lg n)$ time, which is the best possible. Prefix computation is performed using a tree like subgraph (which we call as a '$(k, 1, k)$ chain network'). This network, we believe, is applicable for many other computations as well. Similar networks have been used before [7, 2].

Efficient packet routing algorithms for the star graph have already been obtained in [9]. Although the best known randomized routing algorithm for the star graph runs in $O(n)$ time with very high probability [9], due to the lower bound of [5], the best known deterministic oblivious routing algorithm for the star graph needs a much higher running time. In this paper we develop a non-oblivious deterministic routing algorithm with $O(n^3)$ running time.

The rest of this paper is organized as follows. Section 2 introduces some properties of the star graph. Section 3 contains our prefix algorithm. In section 4 we present our selection algorithm for the star graph while Section 5 describes our randomized sorting algorithm. The deterministic routing algorithm is presented in section 6. Section 7 concludes the paper.

2 Preliminaries

We first define the star graph and then give some definitions and lemmas that will be helpful throughout.

2.1 The Star Graph

Definition 2.1 *Let $s_1 s_2 \ldots s_n$ be a permutation of n symbols, e.g., $1 \ldots n$. For $1 < j \leq n$, we define $SWAP_j(s_1 s_2 \ldots s_n) = s_j s_2 \ldots s_{j-1} s_1 s_{j+1} \ldots s_n$.*

Definition 2.2 *An n-star graph is a graph $S_n = (V, E)$ with $\mid V \mid = n!$ nodes, where $V = \{s_1 s_2 \ldots s_n \mid s_1 s_2 \ldots s_n$ is a permutation of n different symbols$\}$, and $E = \{(u, v) \mid u, v \in V$ and $v = SWAP_j(u)$ for some $j, 1 < j \leq n\}$.*

The 3-star and 4-star graphs are shown in Figure 1. It is not hard to see (from Definition 2.2) that the degree of the n-star graph is $n - 1$. Also, in [1], Akers, Harel, and Krishnamurthy have shown that the diameter of the n-star graph is $\lfloor \frac{3}{2}(n-1) \rfloor$. On the other hand, an n-cube has 2^n nodes, degree n, and diameter n. Thus, in comparison with the n-cube, the degree and diameter of the star graph grow more slowly as functions of the network size. Moreover, the star graph is both vertex (node) symmetric and edge symmetric (just like the n-cube). We assume that the star graph is a MIMD machine in which at each step different nodes could perform different instructions.

Definition 2.3 *A subgraph of an n-star graph S_n is said to be an i-th stage subgraph, denoted $S_{n-i}(s_{n-i+1} s_{n-i} \cdots s_n)$, iff S_{n-i} is itself an $(n-i)$-star graph, $0 < i < n$, and the last i symbols of labels of all the nodes in it are identical.*

The S_{i-1}'s of an S_i partition the S_i into i identical subgraphs. For example, an S_4 consists of 4 S_3's, viz., $S_3(1), S_3(2), S_3(3)$, and $S_3(4)$, and each of the S_3's consists of 3 S_2's, and so on.

Definition 2.4 *The i-th position of the permutation labeling a node u in S_n is denoted by u_{s_i}, $1 \leq i \leq n$.*

Definition 2.5 *The path between a pair of nodes $u, v \in V$ is an ordered sequence of nodes and links (edges) in the graph, such that the first and the last nodes in the sequence are u and v. Adjacent nodes are directly connected by a link in the sequence. The length of the path is the number of links in the path. Adjacent nodes, say u_j and $u_{j+1} = SWAP_i(u_j)$, together with the link connecting them, are denoted by $u_j \overset{SWAP_i}{\longleftrightarrow} u_{j+1}$. For example, in S_4, $4231 \overset{SWAP_4}{\longleftrightarrow} 1234 \overset{SWAP_2}{\longleftrightarrow} 2134 \overset{SWAP_4}{\longleftrightarrow} 4132$ denotes a path of length 3.*

Definition 2.6 *The distance between two nodes u and v is the length of the shortest path between u and v.*

For any network sorting algorithm, we need to specify an ordering (also known as an indexing scheme) of the nodes. The indexing scheme we adopt is reverse lexicographic order and is the same as the one assumed in [7]. Table 1 gives the indexing scheme for S_4.

Definition 2.7 *(Reverse lexicographic order:) Let $\prec$ be the ordering of nodes in the network. Let u be the node labelled as $u_{s_1}u_{s_2}\cdots u_{s_n}$ and v be the node labelled as $v_{s_1}v_{s_2}\cdots v_{s_n}$. Then $u \prec v$ iff there exists an $i, 1 < i \leq n$, such that $u_{s_j} = v_{s_j}$ for all $j > i$, and $u_{s_i} < v_{s_i}$.*

Definition 2.8 *Consider a k-star graph S_k. S_k consists of k copies of S_{k-1}. These copies can be arranged as $S_{k-1}(k), S_{k-1}(k-1), \ldots, S_{k-1}(2), S_{k-1}(1)$ in reverse lexicographic order. We say two or more nodes from distinct S_{k-1}'s are* corresponding *if they have the same index in their respective S_{k-1}'s.*

As an example, in S_4 (see Figure 2), the two nodes 2341 and 1243 are corresponding (since both have index 5 in their S_3's).

Definition 2.9 *A $(k, 1, k)$ chain in S_k is defined to be a sequence of k corresponding nodes $q_k, q_{k-1}, \ldots, q_2, q_1$ such that $q_j \in S_{k-1}(j)$ for $1 \leq j \leq k$.*

Figure 2 identifies all the $(k, 1, k)$ chains (for $1 < k \leq 4$) in an S_4. Notice that there are $(k-1)!$ different $(k, 1, k)$ chains in an S_k. Also, each node in any S_{k-1} belongs to a unique $(k, 1, k)$ chain. If $v_{s_1}v_{s_2}\ldots v_{s_k}$ is any node in S_k, its left neighbor in its $(k, 1, k)$ chain is defined to be the node that is obtained as follows: 'exchange' v_{s_k} with the next smallest symbol. For instance in Figure 2, the left neighbor of 1243 is 1342 and 1342 is obtainable from 1243 by 'exchanging' 3 with 2. Any two symbols can be exchanged with three or less SWAP operations. Right neighbor of $v_{s_1}v_{s_2}\ldots v_{s_k}$ is defined analogously and can be obtained by exchanging v_{s_k} with the next largest symbol.

Thus one could think of a $(k, 1, k)$ chain as a linear array with k nodes. A packet (or item) from one node to its neighbor along the chain can be sent via a physical path of length 3. A $(k, 1, k)$ chain also has the following nice property: Say there is an item at each node of a $(k, 1, k)$ chain $q_k, q_{k-1}, \ldots, q_2, q_1$, and each item has to be moved to its (say) left neighbor. It is easy to see that these items could be moved simultaneously in 3 steps. For an illustration see Table 2.

2.2 Packet Routing and Chernoff Bounds

The following Lemma due to Palis, Rajasekaran and Wei [9] will be applied in our randomized algorithms:

Lemma 2.1 *Permutation routing on S_n can be performed in $O(n)$ time with high probability.*

One of the most frequently used facts in analyzing randomized algorithms is *Chernoff bounds*. These bounds provide close approximations to the probabilities in the tail ends of a binomial distribution. Let X stand for the number of heads in n independent flips of a coin, the probability of a head in a single flip being p. X is also known to have a binomial distribution $B(n,p)$. The following three facts (known as Chernoff bounds) are now folklore:

$$\text{Prob.}[X \geq m] \ \leq \ \left(\frac{np}{m}\right)^m \ e^{m-np},$$

$$\text{Prob.}[X \geq (1+\epsilon)np] \ \leq \ exp(-\epsilon^2 np/2), \text{and}$$

$$\text{Prob.}[X \leq (1-\epsilon)np] \ \leq \ exp(-\epsilon^2 np/3),$$

for any $0 < \epsilon < 1$, and $m > np$.

Like the $O()$ function is used to specify the asymptotic resource bounds of deterministic algorithms, $\widetilde{O}()$ is used to specify resource (like time, space etc.) bounds of randomized algorithms. We say a function $f(.)$ is $\widetilde{O}(g(.))$ if there exist constants c and n_0 such that $f(n) \leq c\alpha g(n)$ with probability $\geq (1 - n^{-\alpha})$ on any input of size $n \geq n_0$, for any $\alpha > 0$.

Throughout let w.h.p. stand for 'with high probability.' By high probability we mean a probability of $\geq (1 - n^{-\alpha})$ for any fixed α, n being the input size.

3 Prefix Computation on the Star Graph

Given a sequence of items $x_0, x_1, \ldots, x_N$ and a binary associative operator $\otimes$, let $p_i = x_0 \otimes x_1 \otimes \cdots \otimes x_i$ for $0 \leq i \leq N$. The process of computing the values $p_0, p_1, \cdots, p_N$ is called a prefix computation. A prefix computation algorithm is an essential tool for the design of numerous other algorithms. In this section we show that on S_n a sequence of n prefix computations can be simultaneously completed in $O(n^2)$ time. In contrast, Akl and Qiu [2] show that a single prefix computation can be completed in $O(n \lg n)$ time and their algorithm is clearly optimal.

First we present our prefix algorithm for a single sequence and later explain how to modify this algorithm for the case of a sequence of prefixes. The star graph under concern is an S_n and there is an element at each node of the graph. The indexing scheme assumed is reverse lexicographic order. There are two phases in the algorithm, namely the forward phase and the reverse phase. There are $n-1$ stages in each phase. In stage i of the forward phase, computation is local to the different S_i's, for $2 \leq i \leq n$.

In fact in any S_i, computation takes place only along a specific $(i,1,i)$ chain, namely the chain in which nodes of largest index from the i different S_{i-1}'s lie. Call any such chain as a *special* $(i,1,i)$ chain. (Each S_i has a unique special $(i,1,i)$ chain.) Referring to Figure 2, in stage 3 of the forward phase, computation takes place only along the chain 2341, 1342, 1243, 1234. Similarly, in stage 2, computation occurs only along the chains 3421, 2431, 2341; 3412,1432,1342; 2413,1423,1243; and 2314,1324,1234. (See also Figure 3.) More details follow.

Algorithm Prefix

(* The forward phase *)

for $i := 2$ *to* n *do*
(* Computation is local to each S_i *)
Perform a prefix computation along the special $(i,1,i)$ chain.

(* The reverse phase *)

for $i := n$ *downto* 2 *do*
(* Computation is local to each S_i *)

Each node q in the special $(i, 1, i)$ chain obtains the sum from its left neighbor and propagates this sum to all the nodes in the special $((i-1), 1, (i-1))$ chain that q belongs to;
The nodes in this $((i-1), 1, (i-1))$ chain, excepting q, simply accumulate the propagated sum to the previously computed sums;

Analysis. In the forward phase, each stage i takes $3(i-1)$ steps. Thus the total run time is $O(n^2)$. In the reverse phase stage i takes time $3i$, accounting for a total of $O(n^2)$ time. Thus the whole algorithm runs in time $\leq 3n^2$. The correctness of the algorithm is quite clear. Thus we get the following

Lemma 3.1 *The prefix computation of a single sequence can be completed on S_n in time $O(n^2)$.*

We could indeed perform a sequence of n prefix computations in $O(n^2)$ time. The idea is to pipeline. The precise definition of our problem is this: There are n items in each one of the $n!$ nodes of S_n. The problem is to: 1) compute the prefix sums of the first items of the nodes; 2) compute the prefix sums of the second items of the nodes; ...; and n) compute the prefix sums of the nth items of the nodes.

We could make use of the same algorithm with a very simple modification. In stage i of the forward phase, compute the prefix sums of the n numbers along the special $(i, 1, i)$ chain using pipeline. Now stage i will terminate in time $3(n+i-2)$ steps. Likewise in stage i of the reverse phase, each node q along the special $(i, 1, i)$ chain obtains the n sums from its left neighbor in $3n$ steps; Followed by this, it propagates these n numbers along its $((i-1), 1, (i-1))$ chain, using pipeline, in $\leq 3(n+i)$ steps. Thus the total run time will be $\leq 9n^2$. We get the following

Lemma 3.2 *A sequence of n prefix computations can be performed on S_n in $O(n^2)$ time.*

COPYING. Consider an S_n. For any $k < n$, say there is a specific S_k of S_n that has $k!$ items (stored one per node), and we want to copy these items to every other S_k. (Similar, but not the same, problems are considered in [2].) This operation will be used in the context of sorting when there are only a small number of keys to be sorted. The idea is to compute the rank of each key making multiple copies of the keys to be sorted.

We could do this copying task as follows: Use all the $((k+1), 1, (k+1))$ chains (in the S_{k+1} that this S_k is in) to copy the contents of the specific S_k into every S_k in its S_{k+1}. The result of this copying is that nodes with the same index in every S_k (of S_{k+1}) will have the **same** item. Now use all the $((k+2), 1, (k+2))$ chains in the S_{k+2} that our S_k is in to make $k+2$ copies of the S_{k+1}. The algorithm proceeds in a similar fashion. Clearly such an algorithm runs in $O(n^2)$ time. Therefore we have the following

Lemma 3.3 *The contents of any S_k in an S_n (for $k < n$) can be copied onto every other S_k in $O(n^2)$ time.*

4 Randomized Selection on the Star Graph

In this section we show that the problem of selection can be solved in $\widetilde{O}(n^2)$ time on a star graph with $n!$ nodes. Given a sequence of N numbers and an integer $1 \leq i \leq N$, the problem of selection is to find the ith smallest element from out of the given N keys. We assume that there is a key at each one of the $N = n!$ nodes to begin with. We prove a stronger result, namely, that we can perform selection of n keys within $\widetilde{O}(n^2)$ time if the ranks of these keys are uniform in the interval $[1, N]$.

4.1 Approach

Randomized selection has a long history [4, 16, 12]. There is a central theme in all these algorithms which we also adopt in our algorithm. The basic steps are: 1) To sample and sort $s = o(N)$ keys from the input;

2) To identify two keys from the sample (call these q_1 and q_2) such that the key to be selected will have a value in the interval $[q_1, q_2]$ w.h.p.; 3) To eliminate all the keys from the input which do not have a value in the interval $[q_1, q_2]$; and 4) Finally to perform an appropriate selection in the set of remaining keys (there will not be many of them w.h.p.).

We adopt the same approach to perform n selections on the star graph. In particular if there is a key at each node of the star graph to begin with, and if $i_j = \frac{jN}{n}$ for $1 \leq j \leq n$, our algorithm will output the i_1th smallest element, the i_2th smallest element, ..., and the i_nth smallest element all in $\widetilde{O}(n^2)$ time.

4.2 The Algorithm

First we show how to perform the selection of a single key and then explain how the same algorithm could be modified to select n different keys. We'll make use of the following facts: We assume a star graph with $N = n!$ nodes.

Fact 4.1 *If $1 \leq \ell \leq N$ is any integer, then there exists a sub-star graph of the n-star graph whose size is $\geq \ell$ and $\leq \ell n$.*

Lemma 4.1 *For any fixed $\epsilon < \frac{1}{2}$, a set of N^ϵ keys distributed in a N-node star graph with no more than one keys per node can be sorted in $\widetilde{O}(n^2)$ time.*

Proof. 1) Perform a prefix computation to assign a unique label to each key from the range $[1, N^\epsilon]$. 2) Now route these keys to a sub-star graph of size $N^{\epsilon'}$ where $\epsilon' \geq \epsilon$ and $\epsilon' \leq \frac{1}{2}$. Realize that a sub-star graph of this size exists (cf. Fact 4.1) and a packet whose label is q can be routed to a node indexed q in the sub-star graph. With this prefix computation and routing step we basically concentrate the keys to be sorted in a sub-star graph whose size is no more than $N^{1/2}$. Let the sub-star graph in which the keys are concentrated be an S_r (with $r!$ nodes). Prefix computation takes $O(n^2)$ time (Lemma 3.2) and routing takes $\widetilde{O}(n)$ time (Lemma 2.1).

3) Next we make a copy of these keys in every S_r in S_n. The number of such copies made will be at least $\sqrt{N}$ and these copies can be made in $O(n^2)$ time (cf. Lemma 3.3). If $S_r^1, S_r^2, \ldots, S_r^t$ is the sequence of S_r's in S_n, we make use of the copy in S_r^p to compute the rank of the pth key, i.e., the key whose label is p (as computed in step 1). Rank computation is done in $O(n^2)$ time as follows: Broadcast the pth key to all the nodes in S_r^p (Notice that broadcast is a special case of prefix computation); Each node then compares its own key with the key received producing a 1 or 0; Then a prefix computation is performed to determine the rank. 4) Finally we route the key whose rank is j to the node indexed j in a specific S_r.

Clearly this algorithm runs in $\widetilde{O}(n^2)$ time. □

Note: The above algorithm can be made deterministic to achieve the same run time. The task of concentration in step 2 can be accomplished using the algorithm of [2]. Step 4 also can be done deterministically in $O(n^2)$ time [2].

We also need the following sampling lemma from [13]. Let $S = \{k_1, k_2, \ldots, k_s\}$ be a random sample from a set X of cardinality N. Let 'select(X, i)' stand for the ith smallest element of X for any set X and any integer i. Also let $k_1', k_2', \ldots, k_s'$ be the sorted order of the sample S. If r_i is the rank of k_i' in X and if $|S| = s$, the following lemma [13] provides a high probability confidence interval for r_i.

Lemma 4.2 *For every α, Prob. $\left(|r_i - i\frac{N}{s}| > c\alpha\frac{N}{\sqrt{s}}\sqrt{\lg N}\right) < N^{-\alpha}$ for some constant c.*

A description of the selection algorithm follows. This algorithm and the analysis of it is very similar to the ones in [10]. To begin with each key is *alive*.

Algorithm Select

repeat forever

1) Count the number of *alive* keys using the prefix sums algorithm. Let M be this number. If M is $\leq N^{2/5}$ then *quit* and go to 7);

2) Each *alive* element includes itself in a sample S with probability $\frac{N^{1/3}}{M}$. The total number of keys in the sample will be $\widetilde{\Theta}(N^{1/3})$;

3) Concentrate the sample keys in a sub-star graph of size no more than $N^{1/2}$ and sort them. Let q_1 be select$(S, i\frac{s}{N} - \delta)$ and let q_2 be select$(S, i\frac{s}{N} + \delta)$, where $\delta = d\sqrt{s \lg N}$ for some constant d ($> c\alpha$) to be fixed;

4) Broadcast q_1 and q_2 to the whole star graph;

5) Count the number of *alive* keys $< q_1$ (call this number M_1); Count the number of *alive* keys $> q_2$ (call this number M_2); If i is not in the interval $(M_1, M - M_2]$, go to 2) else let $i := i - M_1$;

6) Any *alive* key whose value does not fall in the interval $[q_1, q_2]$ *dies*;

end repeat

7)

Concentrate the *alive* keys in a sub-star graph and sort them; Output the ith smallest key from this set.

Theorem 4.1 *The above selection algorithm runs in $\widetilde{O}(n^2)$ time.*

Proof. We first show that the *repeat* loop is executed no more than 5 times w.h.p. Followed by this, we show that each of the seven steps in the algorithm runs in $\widetilde{O}(n^2)$ time.

An application of Lemma 4.2 implies that if d is chosen to be large enough ($> c\alpha$), the ith smallest element will lie between q_1 and q_2 w.h.p. Also, the number of keys *alive* after j runs of the *repeat* loop is $\widetilde{O}\left(\frac{N}{(\sqrt{N^{1/3}})^j}(\sqrt{\lg N})^j\right)$. After 4 runs, this number is $\widetilde{O}(N^{1/3}(\sqrt{\lg N})^4) = \widetilde{O}(N^{2/5})$.

Step 1) of the algorithm takes $O(n^2)$ time since it involves just a prefix sums computation. Steps 2) and 6) take $O(1)$ time each. In Step 3), concentration of keys can be done by a prefix computation followed by a packet routing step (cf. the proof of Lemma 4.1). Sorting is done using the algorithm of Lemma 4.1. Thus step 3) takes $\widetilde{O}(n^2)$ time. Steps 4) and 5) can be completed in $O(n^2)$ time using the prefix algorithm. Step 7) is similar to 3). □

4.3 A Set of n Selections

We show now how to modify the above selection algorithm to perform n selections within time $\widetilde{O}(n^2)$. In particular, we are interested in selecting keys whose ranks are $\frac{N}{n}, \frac{2N}{n}, \ldots, \frac{Nn}{n}$. The main idea is to exploit the fact that a sequence of n prefix computations can be completed in $O(n^2)$ time. Let $i_j = \frac{jN}{n}$ for $1 \leq j \leq n$.

We only indicate the modifications to be done. Steps 1) and 2) remain the same. In step 3, we select $2n$ keys (instead of just two). Call these keys $q_{11}, q_{12}, q_{21}, q_{22}, \ldots, q_{n1}, q_{n2}$. q_{j1} and q_{j2} (for any $1 \leq j \leq n$) are such that the i_jth smallest key in the input (i.e., the jth key to be selected) will have a value in the range $[q_{j1}, q_{j2}]$ w.h.p. and q_{j1} and q_{j2} are defined as before. For instance q_{j1}=select$(S, i_j\frac{s}{N} - \delta)$ where $\delta = d\sqrt{s \lg N}$ for some constant $d > c\alpha$. After identifying this sequence of $2n$ keys, in step 4) the sequence is broadcast to the whole star graph so that each processor has a copy. Clearly, this can be done in $\widetilde{O}(n^2)$ time (Lemma 3.2).

In step 5, count the number of *alive* keys $< q_{j1}$ (call this number M_{j1}) and the number of *alive* keys $> q_{j2}$ (call this number M_{j2}), for each $1 \leq j \leq n$. Broadcast these numbers to each processor as well. If i_j is not in the interval $(M_{j1}, M - M_{j2}]$ for any j go to 2) else let $i_j := i_j - M_{j1} + \sum_{r=1}^{j-1}(M - M_{r1} - M_{r2})$, for

each j. In this step we need to perform twice a sequence of $2n$ prefix computations and hence we only need $O(n^2)$ time (Lemma 3.2).

In step 6), any *alive* key that does not fall in any of the intervals $[q_{11}, q_{12}]$, $[q_{21}, q_{22}]$, $\ldots, [q_{n1}, q_{n2}]$ *dies*. We emphasize that these n intervals will be disjoint w.h.p. This step takes $O(n)$ time.

In step 7), we output n keys whose ranks are $i_1, i_2, \ldots, i_n$.

Analysis At any time in the algorithm the intervals $[q_{11}, q_{12}], [q_{21}, q_{22}], \ldots, [q_{n1}, q_{n2}]$ will be disjoint w.h.p. for the following reasons: During any run of the *repeat* loop, 1) if N' is the number of *alive* keys, the i_j's (for $1 \leq j \leq n$) will be nearly uniform in the range $[1, N']$ w.h.p., and 2) the number of sample keys in the range $[q_{j1}, q_{j2}]$ (for any $1 \leq j \leq n$) will be $O(\sqrt{s \lg N'})$.

The number of *alive* keys after step 6) of run j is seen to be $\widetilde{O}\left(\frac{N}{(\sqrt{N^{1/3}})^j}(\sqrt{\lg N})^j n^j\right)$. After 4 runs, this number is $\widetilde{O}(N^{1/3} \lg^2 N\ n^4) = \widetilde{O}(N^{2/5})$.

The analysis of the other steps is similar. Thus we get the following

Theorem 4.2 *A set of n keys whose ranks are uniform in the interval $[1, N]$ can be selected on an S_n with $N = n!$ nodes in $\widetilde{O}(n^2)$ time, the queue size being $O(n)$.*

5 Randomized Sorting

Randomized algorithms for sorting have been proposed on various models: [16] (PRAM), [17] (CCC), etc. All the abovementioned algorithms have a central idea similar to that of Quicksort. A summary of their approach follows. 1) Given N keys to be sorted, sample $o(N)$ keys and sort the sample using any nonoptimal algorithm; 2) Partition the input using the sample keys as splitters; and 3) Finally sort each part recursively.

Our algorithm takes a different approach. We make use of the selection algorithm as a subroutine. In fact we exploit Theorem 4.2 to partition the given input into n exactly equal parts and sort each part recursively. The indexing scheme used is the reverse lexicographic order.

There are n phases in the algorithm. In the first phase each key will end up in the correct S_{n-1} it belongs to. In the second phase, sorting is local to each S_{n-1}. At the end of second phase each key will be in its correct S_{n-2}. In general, at the end of the ℓth phase, each key will be in its right $S_{n-\ell}$ (for $1 \leq \ell \leq n-1$).

Algorithm Sort

for $i := n$ *downto* 2 *do*
(* Computation is local to each S_i. Let $M_i = i!$ and the nodes in any S_i be named $1, 2, \ldots, M_i$. *)

1) Select i keys whose ranks are uniform in the range $[1, i!]$ using the algorithm of the previous section. At the end of this selection, each node will have a copy of these i keys (call them $k_1, k_2, \ldots, k_i$ in sorted order).

2) Each processor p ($1 \leq p \leq i!$) identifies the S_{i-1} its key k belongs to, by sequentially scanning through the i selected keys. In particular it sets $N_j^p := 1$ if $k_{j-1} < k \leq k_j$; for every other j ($1 \leq j \leq i$) it sets $N_j^p := 0$. (Assume that $k_0 = -\infty$.)

3) Compute the prefix sums of the following i sequences: 1) $N_1^1, N_1^2, \ldots, N_1^{M_i}$; 2) $N_2^1, N_2^2, \ldots, N_2^{M_i}$; ...; i) $N_i^1, N_i^2, \ldots, N_i^{M_i}$.

4) If processor p has set N_j^p to 1 in step 2), it means that the key k of processor p belongs to the jth S_{i-1}. The pth prefix sum of the jth sequence will then assign a unique node for this key k in the jth S_{i-1}. Route each one of the $i!$ keys to a unique node in the S_{i-1} it belongs to.

Analysis. We first compute the time needed for the completion of a single phase (say the ith phase). Later we compute the high probability run time of the whole algorithm. The proof technique for obtaining high probability bound is adopted from [14].

Step 1 can be completed in $O(i^2)$ time w.h.p. Here by high probability we mean a probability of $\geq 1 - \frac{1}{(i!)^c}$ for any constant c. Step 2 can clearly be completed in $O(i)$ steps. Step 3 involves the computation of a sequence of i prefix sums and hence can be performed in $O(i^2)$ time (according to Lemma 3.2). The routing task in step 4) takes $\widetilde{O}(i)$ time (cf. Lemma 2.1).

Thus we can make the following statement: If T_i is the run time of the ith phase, then,

$$\text{Prob.}[T_i \geq c\alpha i^2] \ \leq \ \frac{1}{(i!)^\alpha}$$

for some constant c and any α. But $i!$ is $\Omega((i/e)^i)$ for large i's. Therefore rewriting the above we get

$$\text{Prob.}[T_i \geq c\alpha i^2] \ \leq \ 2^{-\alpha i \lg i}$$

for some constant c and any α. Let $t_i = c'\alpha i^2$ for some constant c'. Then,

$$\text{Prob.}[T_i \geq c\alpha i^2 + t_i] \ \leq \ 2^{-\alpha i \lg i}.$$

Also,

$$\text{Prob.}[T_i \geq c\alpha i^2 + t_i] \ \leq \ 2^{-\sqrt{t_i}}.$$

Let $Q = \sum_{i=1}^{n} i^2$. (Of course Q is $O(n^3)$). If T is the run time of the whole algorithm, we are interested in computing the probability that $T > Q + t$ for any t. This probability is less than the probability of events where $\sum_{i=1}^{n} t_i = t + j$ for $0 \leq j \leq Q$. We compute the probability that $\sum_{i=1}^{n} t_i = t$ and multiply the result by Q to get an upper bound.

Consider a computation tree the root of which is phase 1 of the algorithm. There are n children for the root (one corresponding to phase 2 of each one of the S_{n-1}'s). The tree is defined for the rest of the levels in a similar way. We can associate a time bound for each path in this tree. The run time of our algorithm is nothing but the maximum of all the path times. Consider one such worst case path. Probability that along this path $\sum_{i=1}^{n} t_i$ is $= t$ is $\leq$

$$\Pi_{\sum_{t_i = t}} 2^{-\sqrt{t_i}} \ \leq \ 2^{-\sqrt{t}}.$$

The number of ways of distributing t over the n phases is $t^{O(n)}$. Therefore,

$$\text{Prob.}[T > Q + t] \ < \ Q 2^{-\sqrt{t} + O(n \lg t)}.$$

Taking $t = c'Q$ we get

$$\text{Prob.}[T > Q + c'Q] \ < \ n^3 2^{-\Omega(n^{1.5}) + O(n \lg n)}$$

which is less than $\left(\frac{1}{n!}\right)^\alpha$, for any fixed α and $c' > 0$.

Thus we have the following

Theorem 5.1 *Sorting of $N = n!$ keys can be performed on an S_n in $\widetilde{O}(n^3)$ time, the queue size being $O(n)$.*

6 A Deterministic Routing Algorithm for the Star Graph

The routing problem is defined as follows: A network has a set of packets of information in which a packet is a $\langle source, destination \rangle$ pair. To start with, the packets are placed in their sources. These packets must be sent in parallel to their correct destinations such that at most one packet passes through any link of the network at any time and all packets arrive at their destinations as quickly as possible. Usually, the

performance of a routing algorithm is determined by its *run time* and *queue size*. The run time of a routing algorithm is the time needed for the last packet to reach its destination, and the queue size is the maximum number of packets that will accumulate at any node in the network during the entire course of routing. A paradigmatic case of general routing is **permutation routing** in which initially there is exactly one packet at each node, and exactly one packet is destined for any node. An optimal randomized on-line permutation routing algorithm for the star graph has been obtained in [9]. It runs in time $O(n)$ w.h.p., but requires a queue of size $O(n)$ for each link. Although an oblivious deterministic permutation routing algorithm is also obtained in the same paper, it takes $O(\sqrt{n!})$ steps, and needs a queue of size $O(\sqrt{n!})$ for each node due to the lower bound of [5]. We will present a deterministic routing algorithm which realizes a permutation routing in time $O(n^3)$, and requires only a queue of size n for each node, and without a queue needed for each link.

We first introduce a *packing* procedure which will be invoked by our routing algorithm. A packing problem is a restriction of routing problem, which routes $M \leq N$ packets (one per node), where N is the size of the network, from their sources to a set of M contiguous nodes, say from node s to node $s+M-1$, where $s \geq 1$ and $s+M-1 \leq N$, so that the relative order of these M packets is still preserved. The following Lemma pertains to packing:

Lemma 6.1 Given an n-star graph of $N = n!$ nodes and a set of $M \leq N$ packets, one per node, these M packets can be packed in $O(n^2)$ steps.

Proof : Packing can be done using a 'concentration' followed by a 'cyclic shift' operation both of which can be done in $O(n^2)$ time [2][Section 2.5]. □

Definition 6.1 *A general version of packing called* multiple packing *will be invoked in our packet routing algorithm. Consider an n-star graph S_n where there are a total of $n!$ packets such that there are no more than n packets in any node. If node i has k_i packets, let these packets be arbitrarily named $1, 2, \ldots, k_i$. We could now define n sequences of packets as follows: All the packets that have been named 1 will form the first sequence; All the packets that have been named 2 will form the second sequence, and so on. Let the number of packets in the ith sequence be M_i, for $i = 1, 2, \ldots, n$. The problem of multiple packing is to route the packets in the first sequence to the first M_1 nodes of S_n (one per node), to route the second sequence packets to the next M_2 nodes, and so on.*

The following Lemma applies to multiple packing:

Lemma 6.2 If each node in the n-star graph can receive a packet from each incoming link and send a packet along each outgoing link in one unit of time, then multiple packing can be performed in $O(n^2)$ steps.

Proof : We use the multiple prefix sums algorithm (Lemma 3.2) to determine the destination of each packet in every sequence. Followed by this, we simply pipeline the packings. After each packing is triggered for n steps, we trigger the next packing. Since each individual packing takes $< n^2$ steps (Lemma 6.1), totally n sequences of packing will take $< 2n^2 - n$ steps (because of the overlap due to the pipeline) which is still $O(n^2)$. □

We need the following definition to describe our permutation routing algorithm.

Definition 6.2 *A stage is said to be i-th stage stable, denoted S^i_{stable}, iff for every i-th stage subgraph S_{n-i}, the destination of each packet in the subgraph is in the subgraph itself, and each node of the subgraph has exactly one packet.*

Our algorithm is designed as a sequence of stage transitions $S^0_{stable}, \cdots, S^{n-1}_{stable}$ in which initially we are in S^0_{stable}. In each subsequent stage we route packets such that the stage transits from S^i_{stable} to S^{i+1}_{stable}. This could be done by routing each packet along the $(n-i, 1, n-i)$ chain to which it belongs. However, some

nodes may accumulate several packets because many packets in the same chain may be destined for the same subgraph, and thus end up at the same node. For example, in Figure 2, if the destinations of nodes 1234, 1243, 1342, and 2341 are all in subgraph $S_3(1)$, then during the transition from S^0_{stable} to S^1_{stable}, all these four nodes will be accumulated at node 2341. So as not to keep accumulating too many packets at some nodes in subsequent stages (which might mean longer delays for some packets) we do the following: Before we start the next transition, we balance the network such that each node contains exactly one packet. This could be done by *token distribution*.

According to our algorithm, in stage i, after routing each packet along its $(n-i+1, 1, n-i+1)$ chain to its right subgraph, every node of each subgraph S_{n-i} has between 0 and $n-i+1$ nodes, and each S_{n-i} has exactly $(n-i)!$ nodes. To distribute the packets so that each node of the subgraph has exactly one packet we perform multiple packing, i.e., we simply invoke packing procedure (in Lemma 6.1) $\leq n-i$ times. In each packing, a node which contains more than one packets will contribute a packet to be packed. Also, if previous packing ends at position s, and there are M nodes which contribute packets in current packing, then these packets will be packed to positions from $s+1$ to $s+M$. If the maximum number of packets in the individual nodes of a subgraph is k, then after $k-1$ packings, each node of the subgraph will have exactly one packet.

Remark 1 Observe that for each node in the network, although there may be several packets accumulated at the node during routing, it's not necessary to put these packets in the queue along the links they come in. Because excepting for one of the packets, all other packets will be distributed to other nodes in the same subgraph, and we simply store these packets in the local memory of the node before sending them out.

Theorem 6.1 A permutation routing on the n-star graph can be realized in time $O(n^3)$ without queues needed for each link.

Proof: For a permutation routing, initially the n-star graph is in S^0_{stable}. We perform $n-1$ stage transitions so that eventually the network is in S^{n-1}_{stable}. During the transition from S^i_{stable} to S^{i+1}_{stable}, we first route each packet in a S_{n-i} along its $(n-i+1, 1, n-i+1)$ chain to its right subgraph S_{n-i} (this will take at most $n-i$ steps), and then perform packing for $n-i$ times such that the network is in S^{i+1}_{stable}. Each transition takes $(n-i)$ steps for routing and $O((n-i)^2)$ steps for token distribution (Lemma 6.2). Totally we have $n-1$ transitions, and hence the permutation algorithm takes total of $< \sum_{i=1}^{n-1}(n-i) + (n-i)^2 = O(n^3)$ steps. Also, according to Remark 1, the algorithm requires no queues for each link. □

7 Conclusions

In this paper we have addressed the problems of selection, sorting and routing on the star graph. Randomized algorithms have been given in this paper for sorting and selection. The time bound of our randomized sorting is better than that of the previously best known sorting algorithm. We also have presented a deterministic routing algorithm which runs in $O(n^3)$ time on S_n. Both selection and sorting have the obvious lower bound of $\Omega(n \lg n)$ on the star graph. Discovering algorithms with matching time bounds is still open.

Acknowledgements

The authors thank the referees for their careful reading and very helpful comments. We are grateful to an anonymous referee for pointing out that the algorithm of Lemma 4.1 can be made deterministic.

References

[1] S. Akers, D. Harel and B. Krishnamurthy, The Star Graph: An Attractive Alternative to the n-Cube, Proc. *International Conference of Parallel Processing*, 1987, pp. 393-400.

[2] S.G. Akl and K. Qiu, Data Communication and Computational Geometry on the Star and Pancake Interconnection Networks, TR 91-301, Dept. of Computing and Information Science, Queen's University at Kingston, Ontario, Canada, May 1991.

[3] K. Batcher, Sorting Networks and Their Applications, Proc. *AFIPS Spring Joint Comput. Conf.*, 1968, pp. 307-314.

[4] R.W. Floyd and R.L. Rivest, Expected Time Bounds for Selection, *Communications of the ACM*, vol. 18, no.3, 1975, pp. 165-172.

[5] C. Kaklamanis, D. Krizanc and Th. Tsantilas, Tight Bounds for Oblivious Routing in the Hypercube, Proc. ACM Symposium on Parallel Algorithms and Architectures, 1990, pp. 31-36.

[6] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Trees, Arrays, Hypercubes*, Morgan-Kaufmann Publishers, San Mateo, CA, 1992.

[7] A. Menn and A.K. Somani, An Efficient Sorting Algorithm for the Star Graph Interconnection Network, Proc. *International Conference on Parallel Processing*, 1990, vol. 3, pp. 1-8.

[8] D. Nassimi and S. Sahni, Bitonic Sort on a Mesh Connected Parallel Computer, *IEEE Trans. on Computers*, C-28:2-7, 1979.

[9] M. Palis, S. Rajasekaran and D.S.L. Wei, Packet Routing and PRAM Emulation on Star Graphs and Leveled Networks, *Journal of Parallel and Distributed Computing* 20, 1994, pp. 145-157.

[10] S. Rajasekaran, Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing, Sorting and Selection, Proc. *First Annual European Symposium on Algorithms*, 1993. See also *IEEE Transactions on Computers*, 45(5), 1996, pp.529-539.

[11] S. Rajasekaran, $k - k$ Routing, $k - k$ Sorting, and Cut Through Routing on the Mesh, *Journal of Algorithms* 19, 1995, pp. 361-382.

[12] S. Rajasekaran, Randomized Parallel Selection, Proc. *Tenth International Conference on Foundations of Software Technology and Theoretical Computer Science*, 1990. Springer-Verlag Lecture Notes in Computer Science 472, pp. 215-224.

[13] S. Rajasekaran and J.H. Reif, Derivation of Randomized Sorting and Selection Algorithms, in *Parallel Algorithm Derivation And Program Transformation*, edited by Paige, Reif, and Wachter, Kluwer Academic Publishers, 1993, pp.187-205.

[14] S. Rajasekaran, and S. Sen, Random Sampling Techniques and Parallel Algorithms Design, in *Synthesis of Parallel Algorithms*, editor: Reif, J.H., Morgan-Kaufmann Publishers, San Mateo, California, 1993.

[15] S. Rajasekaran and D.S.L. Wei, Selection, Routing, and Sorting on the Star Graph, Proc. *International Parallel Processing Symposium*, 1993.

[16] R. Reischuk, Probabilistic Parallel Algorithms for Sorting and Selection, *SIAM Journal of Computing*, 14(2), 1985, pp. 396-411.

[17] J.H. Reif and L.G. Valiant, A Logarithmic Time Sort for Linear Size Networks, *Journal of the ACM*, volume 34, January, 1987, pp. 60-76.

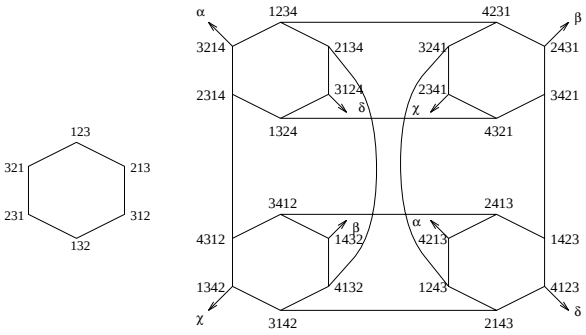

Figure 1: 3-star graph and 4-star graph.

permutation	index
4321	0
3421	1
4231	2
2431	3
3241	4
2341	5
4312	6
3412	7
4132	8
1432	9
3142	10
1342	11
4213	12
2413	13
4123	14
1423	15
2143	16
1243	17
3214	18
2314	19
3124	20
1324	21
2134	22
1234	23

Table 1: An indexing scheme for S_4.

Figure 2: All the $(k, 1, k)$ chains in an S_4, for $1 < k \leq 4$.

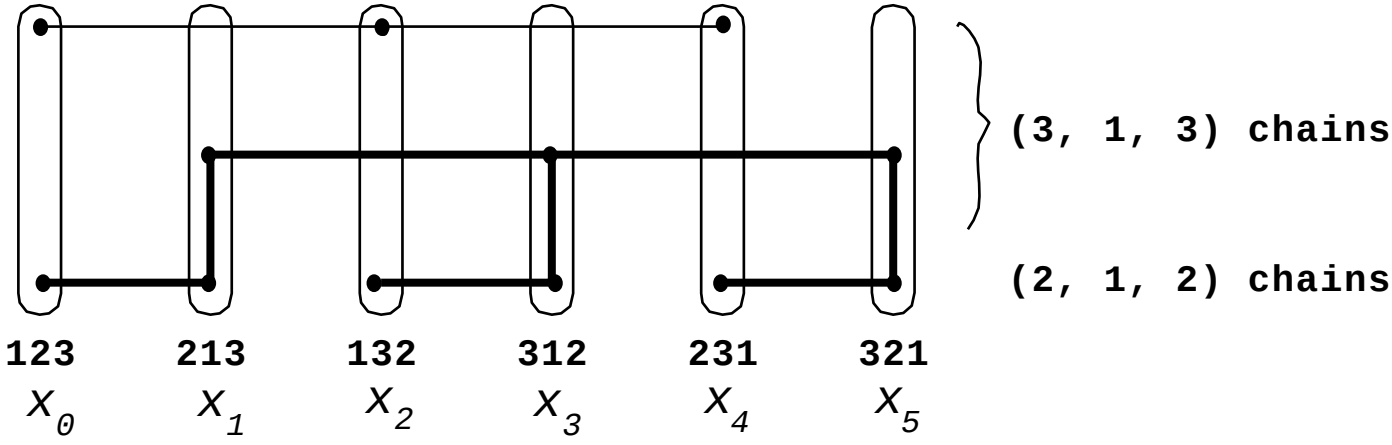

Figure 3: A tree-like $(k, 1, k)$ chain network for the prefix computations of the star graph.

4321	$\overset{SWAP_k}{\longleftrightarrow}$	1324	$\overset{SWAP_3}{\longleftrightarrow}$	2314	$\overset{SWAP_k}{\longleftrightarrow}$	4312
3421	$\overset{SWAP_k}{\longleftrightarrow}$	1423	$\overset{SWAP_3}{\longleftrightarrow}$	2413	$\overset{SWAP_k}{\longleftrightarrow}$	3412
4231	$\overset{SWAP_k}{\longleftrightarrow}$	1234	$\overset{SWAP_2}{\longleftrightarrow}$	2134	$\overset{SWAP_k}{\longleftrightarrow}$	4132
2431	$\overset{SWAP_k}{\longleftrightarrow}$	1432	$\overset{SWAP_4}{\longleftrightarrow}$	2431	$\overset{SWAP_k}{\longleftrightarrow}$	1432
3241	$\overset{SWAP_k}{\longleftrightarrow}$	1243	$\overset{SWAP_2}{\longleftrightarrow}$	2143	$\overset{SWAP_k}{\longleftrightarrow}$	3142
2341	$\overset{SWAP_k}{\longleftrightarrow}$	1342	$\overset{SWAP_4}{\longleftrightarrow}$	2341	$\overset{SWAP_k}{\longleftrightarrow}$	1342
4312	$\overset{SWAP_k}{\longleftrightarrow}$	2314	$\overset{SWAP_2}{\longleftrightarrow}$	3214	$\overset{SWAP_k}{\longleftrightarrow}$	4213
3412	$\overset{SWAP_k}{\longleftrightarrow}$	2413	$\overset{SWAP_4}{\longleftrightarrow}$	3412	$\overset{SWAP_k}{\longleftrightarrow}$	2413
4132	$\overset{SWAP_k}{\longleftrightarrow}$	2134	$\overset{SWAP_3}{\longleftrightarrow}$	3124	$\overset{SWAP_k}{\longleftrightarrow}$	4123
1432	$\overset{SWAP_k}{\longleftrightarrow}$	2431	$\overset{SWAP_3}{\longleftrightarrow}$	3421	$\overset{SWAP_k}{\longleftrightarrow}$	1423
3142	$\overset{SWAP_k}{\longleftrightarrow}$	2143	$\overset{SWAP_4}{\longleftrightarrow}$	3142	$\overset{SWAP_k}{\longleftrightarrow}$	2143
1342	$\overset{SWAP_k}{\longleftrightarrow}$	2341	$\overset{SWAP_2}{\longleftrightarrow}$	3241	$\overset{SWAP_k}{\longleftrightarrow}$	1243
4213	$\overset{SWAP_k}{\longleftrightarrow}$	3214	$\overset{SWAP_4}{\longleftrightarrow}$	4213	$\overset{SWAP_k}{\longleftrightarrow}$	3214
2413	$\overset{SWAP_k}{\longleftrightarrow}$	3412	$\overset{SWAP_2}{\longleftrightarrow}$	4312	$\overset{SWAP_k}{\longleftrightarrow}$	2314
4123	$\overset{SWAP_k}{\longleftrightarrow}$	3124	$\overset{SWAP_4}{\longleftrightarrow}$	4123	$\overset{SWAP_k}{\longleftrightarrow}$	3124
1423	$\overset{SWAP_k}{\longleftrightarrow}$	3421	$\overset{SWAP_2}{\longleftrightarrow}$	4321	$\overset{SWAP_k}{\longleftrightarrow}$	1324
2143	$\overset{SWAP_k}{\longleftrightarrow}$	3142	$\overset{SWAP_3}{\longleftrightarrow}$	4132	$\overset{SWAP_k}{\longleftrightarrow}$	2134
1243	$\overset{SWAP_k}{\longleftrightarrow}$	3241	$\overset{SWAP_3}{\longleftrightarrow}$	4231	$\overset{SWAP_k}{\longleftrightarrow}$	1234

Table 2: The communication between each pair of adjacent nodes in $(4, 1, 4)$ chains of S_4.