Counting-sort and Routing in a Single Hop Radio Network

Maciej Gębala and Marcin Kik
Maciej.Gebala@pwr.wroc.pl, Marcin.Kik@pwr.wroc.pl

Institute of Mathematics and Computer Science,
Wrocław University of Technology
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract. We consider two problems. First, sorting of n integer keys from the $[0, 2^m - 1]$ range, stored in p stations of a single-hop and single channel radio network. Second problem is routing of the packets between the stations of the network. We introduce counting-sort algorithm which has $3mr_i + s_i + d_i + 3$ energetic cost and $nm + n + p$ time cost, where station a_i stores s_i keys (r_i distinct keys) and receives d_i keys. On the basis of this sorting, we construct routing protocols with energetic costs $(3\lceil\log_2 p\rceil + 2)r_i + s_i + d_i + 5$ and $(3\lceil\log_2 p\rceil + 4)r_i + s_i + d_i + 6$, and time costs $n\lceil\log_2 p\rceil + n + 3p$ and $r\lceil\log_2 p\rceil + n + r + 3p$, respectively, where r is sum of all r_i. Our routing is attractive alternative for previous solutions, since it is efficient, deterministic and simple.

1 Introduction

Radio network is a distributed system with no central arbiter, consisting of p radio transceivers called *stations*. Stations are usually small devices running on batteries. Therefore, it is of big importance to design protocols for radio networks with power efficiency in mind, i.e., the station must consume as little power as possible. We assume that each of p stations belonging to the radio network has unique ID – an integer in the $[0, p - 1]$ range. We consider only static radio networks where the number of station is fixed and no members join or leave the network during the protocol operation. In this paper we focus on single-hop radio networks where each single station lies within the transmission range of all other stations. Finally, we only consider model with single channel of communication. A station uses energy only when its transceiver is active, i.e. while sending or receiving any information. When transceiver is inactive the energy consumption is very small and therefore is ignored. We assume that a station uses one unit of energy for sending or receiving single message.

Let p denote the number of stations. For $0 \leq i \leq p - 1$, station a_i initially stores s_i items (with r_i distinct values) and is destination of d_i items from q_i other stations. By $n = \sum_{i=0}^{p-1} s_i$ we denote total number of items. Let $r = \sum_{i=0}^{p-1} r_i$ and $q = \sum_{i=0}^{p-1} q_i$. We assume that during a single round each station can send or receive no more than single message containing either single key or an integer between 0 and n.

We consider two problems. First, sorting n integer keys of range $[0, 2^m - 1]$ stored in p stations of a single-hop and single channel radio network. Second, problem of routing of the packets between the stations of the network.

1.1 Previous works

Some energy efficient sorting algorithms are described in [7,8,2,3]. Singh and Prasanna [7,8] proposed sorting algorithm based on quick-sort and balanced selection with $\Theta(\log n)$ energy cost and $\Theta(\frac{n}{c}\log n)$ time cost, where c is the number of communication channels. Sorting based on merging and an algorithm for merging with energetic cost $O(\log^* n)$ has been proposed in [2]. In these algorithms it is assumed that each station stores single key (i.e. $p = n$). The algorithm described in [3] extends results from [2] for sorting n keys stored in p stations (where each station stores $\frac{n}{p}$ keys) with $8\frac{n}{p}\log_2 p + 2(\log_2 p + 1)\log_2 p$ energetic cost and $(3n + 2p - 2)\log_2 p$ time cost.

Some energy efficient permutation routing protocols are described in [5,1]. The protocol described by Nakano, Olariu and Zomaya routes n packets between p station (each station stores $\frac{n}{p}$ and is destination for $\frac{n}{p}$ packets) with $(4d+7b-1)\frac{n}{p}$ energetic cost and $(2d + 2b + 1)\frac{n}{c} + c$ time cost, where $d = \left\lceil \frac{\log \frac{n}{c}}{\log \frac{n}{p}} \right\rceil$ and $b = \left\lceil \frac{\log c}{\log \frac{n}{p}} \right\rceil$. In such case, if we consider only single channel radio network the cost reaches $(4\left\lceil \frac{\log p}{\log \frac{n}{p}} \right\rceil - 1)\frac{n}{p}$ for energy and $(2\left\lceil \frac{\log p}{\log \frac{n}{p}} \right\rceil + 1)n + 1$ for time. Datta and Zomaya in [1] presented algorithm with $6\frac{n}{p} + 2p + 8$ energetic cost and $2n + p^2 + p + 2$ time cost. Both algorithms are effective when $p \approx \sqrt{n}$.

In [6] Nakano, Olariu and Zomaya introduced randomized routing protocol where for every $f \geq 1$ the task of routing n items in p stations can be completed with probability exceeding $1 - 1/f$ in time $n + O(q + \ln f)$ with energetic cost below $s_i + d_i + O(q_i + r_i \log p + \log f)$.

1.2 Our results

In this paper we present two following results (we use the notation introduced so far)

Theorem 1. *For the single hop and single channel radio network with p stations there exists sorting algorithm for n integer keys of range $[0, 2^m - 1]$ that works in $mn + n + p$ rounds of time and where each station a_i uses no more than $3mr_i + d_i + s_i + 3$ energy.*

Theorem 2. *For the single hop and single channel radio network with p stations there exist routing algorithms that work*

1. *in $n\lceil \log_2 p \rceil + n + 3p$ rounds of time with each station a_i using no more than $(3\lceil \log_2 p \rceil + 2)r_i + s_i + d_i + 5$ energy;*
2. *in $r\lceil \log_2 p \rceil + n + r + 3p$ rounds of time with each station a_i using no more than $(3\lceil \log_2 p \rceil + 4)r_i + s_i + d_i + 6$ energy.*

The second algorithm is much faster than the first one, if $n \gg r$ (and r is always bounded by $\min\{n, p(p-1)\}$).

For p nearing n (for example when $n \approx p \log_2 p$) our routing is more efficient than algorithms described in [5,1]. Besides it is more universal (arbitrary vs. permutation routing). Also comparing with randomized algorithm described in [6] our protocol is simpler and has comparable energy complexity, without randomization.

2 Preliminaries.

We assume that each key is an integer in the range $[0, 2^m - 1]$. Let $b_i(key)$ denote ith bit in the binary representation of key (i.e. $key = \sum_{i=0}^{m-1} 2^i \cdot b_i(key)$). Let $g_l(key) = \sum_{i=l}^{m-1} 2^{i-l} \cdot b_i(key)$. For each (*level*) l, $m \geq l \geq 0$, we say that key is in the *group* $G(g_l(key), l)$ (i.e. $G(g, l) = \{k : 0 \leq k \leq 2^m - 1 \wedge g_l(k) = g\}$). There are 2^{m-l} disjoint groups on level l, partitioning the set $\{0, \ldots, 2^m - 1\}$ into blocks of size 2^l.

The number of the stations in the network is denoted by p. For $0 \leq i \leq p-1$, a_i denotes the ith station of the network. Each station initially stores s_i keys in its local (sorted) table $key[a_i][0 \ldots s_i - 1]$. Let $n = \sum_{i=0}^{p-1} s_i$ denote the total number of keys. Let $POS = \{(i, j) : 0 \leq i < p \wedge 0 \leq j < s_i\}$ (set of positions of elements). And let r_i denote the number of distinct values of the keys in $key[a_i]$. Each a_i stores these values in $key'[a_i][0 \ldots r_i - 1]$. For $0 \leq j \leq r_i - 1$, $c_{i,j}$ denotes the number of copies of $key'[a_i][j]$ in $key[a_i]$. Thus $s_i = \sum_{j=0}^{r_i-1} c_{i,j}$. Let $r = \sum_{i=0}^{p-1} r_i$.

Let d_i be the number of keys for which a_i is final destination, and let q_i be the number of stations that initially stored such keys ($q_i \leq d_i$).

3 Counting-rank

We use following additional local variables in station a_i:

- $lrm[a_i]$ – copy of last received message (if needed),
- $rig[a_i][j]$ – rank of $key[a_i][j]$ in its "current" group,
- $rng[a_i][j]$ – rank of $key[a_i][j]$ in its "next" group,
- $gs[a_i][j]$ – number of keys in group of $key[a_i][j]$,
- $bg[a_i][j]$ – number of keys in groups preceding group of $key[a_i][j]$.
- $rank[a_i][j]$ – "current" rank of $key[a_i][j]$.
- $first[a_i][j]$, $last[a_i][j]$ – additional Boolean variables used in routing procedures

We say that "(i, j) is in $G(g, l)$" if and only if $key[a_i][j] \in G(g, l)$.

In the procedure Init, each station a_i learns the total number of keys n and the initial ranks of its keys in the (*single*) group on level m. Note that this initial ranking depends only on the initial positions of the keys and totally ignores

```
procedure Init(⟨a_0, ..., a_{p-1}⟩,m)
begin
    Each station a_i does, for each j: bg[a_i][j] ← 0;
    station a_0 does: begin
        lrm[a_0] ← 0;
        foreach j ∈ {0, ..., s_0 − 1} do rig[a_0][j] ← j;
    end
    for i ← 0 to p − 2 do
        a_i sends ⟨x⟩, where x = lrm[a_i] + s_i;
        a_{i+1} receives ⟨x⟩ and does: begin
            lrm[a_{i+1}] ← x;
            foreach j ∈ {0, ..., s_{i+1} − 1} do rig[a_{i+1}][j] ← j + x;
        end
    In the last time slot: begin
        station a_{p−1} broadcasts ⟨x⟩, where x = lrm[a_{p−1}] + s_{p−1};
        each other station receives ⟨x⟩;
        each station a_i does, for each j: n[a_i] ← gs[a_i][j] ← x;
    end
    each station a_i does, for each j: begin
        if s_i > 0 and rig[a_i][0] = 0 then rng[a_i][0] ← 0;
        rank[a_i][j] ← bg[a_i][j] + rig[a_i][j];
    end
end
```

Algorithm 1: Procedure Init

their values: $rank[a_i][j] < rank[a_{i'}][j']$ if and only if (i, j) is less than (i', j') in lexicographical ordering.

Init is used in the procedure Counting-rank. In the procedure Counting-rank we compute the final ranks of the keys in the sorted sequence of n keys. For equal keys their ordering is the same as in the initial sequence. Hence, our procedure is suitable for *stable* sorting. For each station a_s and for $0 \leq i < n$, we define set of indexes $S(a_s, i)$ as follows: $S(a_s, t) = \{j : bg[a_s][j] < t < bg[a_s][j] + gs[a_s][j]\}$. Note that the value $S(a_s, t)$ depends only on the local variables of a_s and may be computed by internal computations of a_s. Intuitively, it denotes the set positions (s, j) located in a_s that are in the same "current" group as the position with rank t. (We will show in Lemma 2(5), that all positions have consistent view of their current group.)

For $m \geq l \geq 0$, for $(i, j) \in POS$, we say that (i, j) *is classified on level* l if and only if the following conditions are satisfied:

1. $gs[a_i][j]$ is the number of positions (i', j') in $G(g_l(key[a_i][j]), l)$, and
2. $bg[a_i][j]$ is the total number of positions (i', j') in the groups $G(g, l)$ such that $0 \leq g < g_l(key[a_i][j])$, and
3. $rig[a_i][j]$ is the rank (in the lexicographical ordering by (i, j)) of (i, j) in $G(g_l(key[a_i][j]), l)$, and
4. $rank[a_i][j] = bg[a_i][j] + rig[a_i][j]$ (final result, if we ignore the bits $l-1, \ldots, 0$ in the keys).

```
procedure Counting-rank(⟨a_0, ..., a_{p-1}⟩,m)
begin
    Init(⟨a_0, ..., a_{p-1}⟩,m);
    (* REGROUPING PHASE *)
    for l ← m − 1 downto 0 do
        for t ← 0 to n − 1 do
            For the (unique) pair (a_snd, j') such that rank[a_snd][j'] = t, a_snd does:
            begin
                if rig[a_snd][j'] = 0 then lrm[a_snd] ← 0;
                Let x = lrm[a_snd] + (1 − b_l(key[a_snd][j']));
                if |S(a_snd, t)| < gs[a_snd][j'] and
                ( rig[a_snd][j'] = gs[a_snd][j'] − 1 or j' = s_snd − 1 or
                bg[a_snd][j' + 1] ≠ bg[a_snd][j'] ) then
                    a_snd sends message ⟨x⟩
            end
            For the (at most one) pair (a_rcv, j) such that rank[a_rcv][j] = t + 1 and
            bg[a_rcv][j] ≤ t < bg[a_rcv][j] + gs[a_rcv][j], a_rcv listens to ⟨x⟩ (unless
            a_rcv = a_snd) and does:
            (* CASE A: key[a_rcv][j] is successor of key[a_snd][j'] in its group *)
            begin
                if b_l(key[a_rcv][j]) = 0 then rng[a_rcv][j] ← x;
                else rng[a_rcv][j] ← rig[a_rcv][j] − x;
                lrm[a_rcv] ← x;
            end
            Each a_rcv such that ∃_j bg[a_rcv][j] ≤ t = bg[a_rcv][j] + gs[a_rcv][j] − 1,
            listens to ⟨x⟩ (unless a_rcv = a_snd) and does, for each j ∈ S(a_rcv, i):
            (* CASE B: key[a_snd][j'] is the last one in its group *)
            begin
                if b_l(key[a_rcv][j]) = 0 then
                    gs[a_rcv][j] ← x;
                else
                    bg[a_rcv][j] ← bg[a_rcv][j] + x;
                    gs[a_rcv][j] ← gs[a_rcv][j] − x;
                rig[a_rcv][j] ← rng[a_rcv][j];
                rank[a_rcv][j] ← bg[a_rcv][j] + rig[a_rcv][j];
            end
    each a_i, for each j, does: begin
        if rig[a_i][j] = 0 then first[a_i][j] ← true; else first[a_i][j] ← false;
        if rig[a_i][j] = gs[a_i][j] − 1 then last[a_i][j] ← true;
        else last[a_i][j] ← false;
    end
end
```

Algorithm 2: Procedure Counting-rank

For $m > l' \geq 0$ and $0 \leq t' < n$, let $slot(l', t')$ denote the time slot of REGROUPING PHASE in which the variables l and t have values l' and t', respectively. Let $slot(-1, 0)$ denote the first time slot *after* the REGROUPING PHASE. Let $next(t, l)$ denote the next slot after $slot(t, l)$. For $m > l \geq 0$, $next(l, t) = slot(l, t + 1)$ if $0 \leq t < n - 1$, and $next(l, n - 1) = slot(l - 1, 0)$.

Lemma 1. *In* Counting-rank*:*

1. *For any $(i, j) \in POS$, if, after some time slot, $rig[a_i][j] = 0$, then in all the following time slots $rng[a_i][j] = rig[a_i][j] = 0$.*
2. *For any $(i, j) \in POS$, before each slot after* Init*, we have $rank[a_i][j] = bg[a_i][j] + rig[a_i][j]$.*

Proof. The code ensures that $rng[a_i][j]$ becomes zero whenever $rig[a_i][j]$ becomes zero. If at the beginning of time slot $rig[a_i][j] = 0$ then $rank[a_j][j] = bg[a_j]$. Thus if $rank[a_i][j] = t + 1$ then $t < bg[a_i][j]$ and a_i will not execute code of CASE A of REGROUPING PHASE (the only fragment that could change $rng[a_i][a_j]$). Consequently $rng[a_i][j]$ and $rig[a_i][j]$ will remain equal to zero. The property 2 can be easily seen from the code. □

Lemma 2. *For $m - 1 \geq l \geq 0$ and $0 \leq t < n$ or $(l, t) = (-1, 0)$, for each $(i, j) \in POS$, at the beginning of $slot(l, t)$:*

1. *either:*
   - *$t < bg[a_i][j] + gs[i][j]$ and (i, j) is classified on level $l + 1$, or*
   - *$t \geq bg[a_i][j] + gs[i][j]$ and (i, j) is classified on level l,*

   and
2. *if $t \geq rank[a_i][j]$ then $rng[a_i][j]$ is the rank (in the lexicographical ordering by (i, j)) of $key[a_i][j]$ in the group $G(g_l(key[a_i][j]), l)$, and*
3. *if $t = rank[a_i][j]$ and $rig[a_i][j] > 0$, then $lrm[a_i][j]$ is the number of pairs (i', j') in $G(g_{l+1}(key[a_i][j], l+1))$ with $rig[a_{i'}][j'] < rig[a_i][j]$ and $b_l(key[a_{i'}][j']) = 0$, and*
4. *$\{rank[a_i][j] : (i, j) \in POS\} = \{0, \ldots, n - 1\}$, and*
5. *for each two pairs (i, j) and (i', j'), such that $0 \leq i, i' < p$, and $0 \leq j < s_i$, $0 \leq j' < s_{i'}$, either:*
   - *$bg[i][j] = bg[i'][j']$ and $gs[i][j] = gs[i'][j']$, or*
   - *$bg[i][j] + gs[i][j] \leq bg[i'][j']$, or*
   - *$bg[i'][j'] + gs[i'][j'] \leq bg[i][j]$.*

Proof. We prove Lemma 2 by induction on time slots of REGROUPING PHASE. (I.e. we show that the conditions of the lemma hold for $slot(m-1, 0)$ and that if they hold for $slot(l, t)$ then they also hold for $next(l, t)$.) For $slot(m - 1, 0)$, the conditions of Lemma 2 are enforced by the Init procedure. Let us assume that the conditions hold for $slot(l, t)$, where $m - 1 \geq l \geq 0$ and $0 \leq t < n$. By condition 4, there is exactly one pair (a_{snd}, j') such that $rank[a_{snd}] = t$. By condition 1, (snd, j) is classified on level $l + 1$, since $t = rank[a_{snd}][j'] = bg[a_{snd}][j'] + rig[a_{snd}][j'] < bg[a_{snd}][j'] + gs[a_{snd}][j']$. Let $G' = G(g_{l+1}(key[a_{snd}][j']), l + 1)$. By conditions 5 and 1, all the pairs (i, j) in G' are classified on level $l + 1$. Let

$G'_0 = \{k \in G' : b_l(k) = 0\}$ and $G'_1 = G' \setminus G'_0$ (the two groups on level l that are halves of G'). The value x computed by a_{snd} is the number of pairs (i,j) in G'_0 with $rig[a_i][j] \leq rig[a_{snd}][j']$, since either $rig[a_{snd}][j'] = 0$ and a_{snd} executed $lrm[a_{snd}] \leftarrow 0$, or it follows from condition 3.

We look at variables at the beginning of $slot(l,t)$ and define three sets:

$$A = \{(i,j) : bg[a_i][j] \leq t < bg[a_i] + gs[a_i] - 1\}$$
$$B = \{(i,j) : t = bg[a_i] + gs[a_i] - 1\}$$
$$C = \{(i,j) : t < bg[a_i] \vee bg[a_i][j] + gs \leq t\}$$

Note that, by conditions 5 and 1: A, B, C is a partition of POS, and $A \cup B$ is the set of pairs that are in G', and either $A = \emptyset$ or $B = \emptyset$.

Consider the case $A \neq \emptyset$ (CASE A). Then there is exactly one pair (rcv, j) in G' such that $rank[a_{rcv}][j] = t + 1$. If $rcv \neq snd$, then $|S(a_{snd}, t)| < gs[a_{snd}][j']$ and either $j' = s_{snd} - 1$ or $bg[a_{snd}][j'+1] \neq bg[a_{snd}][j'] + 1$. (Otherwise $rcv = snd$ since $key[a_{snd}]$ is sorted and the keys in $key[a_{snd}]$ from G' are blocked together and have consecutive ranks.) Hence a_{snd} broadcasts $\langle x \rangle$, if necessary. (rcv, j) is preceded by $rig[a_{rcv}][j]$ pairs (i', j') in G' and x of them are in G'_0. Thus (rcv, j) should be ranked in its group on level l on position x, if $b_l(key[a_rcv][j]) = 0$, and on position $rig[a_{rcv}][j] - x$, otherwise. It follows that $rng[a_{rcv}][j]$ is updated so that condition 2 is satisfied in $next(l,t)$. The execution of $lrm[a_{rcv}] \leftarrow x$ makes the condition 3 satisfied in $next(l,t)$. Condition 1 remains satisfied in $next(l,t)$, since (in CASE A) $next(t,l) = slot(l, t+1)$ and, for each pair (i,j) in G', $t + 1 < bg[a_i][j + gs[a_i][j]$ and (i,j) remains classified on level $l + 1$. For all pairs in C condition 1 does not change. Conditions 4 and 5 remain satisfied, since none of the involved variables is changed.

Consider the case $B \neq \emptyset$ (CASE B). For all pairs (rcv, j) in G', the station a_{rcv} has the same values bg and gs. Hence, all of them execute code for CASE B. If there is some pair (rcv, j) in G', such that $rcv \neq snd$, then $|S(a_{snd}, t)| < gs[a_{snd}][j']$. Since, in CASE B, $t = rank[a_{snd}][j'] = bg[a_{snd}][j'] + gs[a_{snd}][j'] - 1$, it follows that $rank[a_{snd}][j'] - bg[a_{snd}][j'] = rig[a_{snd}][j'] = gs[a_{snd}][j'] - 1$, and a_{snd} broadcasts $\langle x \rangle$, if necessary. Since (snd, j) is the last pair in G', the value x is the number of pairs in G'_0. Hence, each pair $(rcv, j) \in G'_0$ properly updates $gs[a_{rcv}][j]$ to x ($bg[a_{rcv}][j]$ remains unchanged), and each pair $(rcv, j) \in G'_1$ properly decreases $gs[a_{rcv}][j]$ and increases $bg[a_{rcv}][j]$ by x, for classification on level l. Besides (by condition 2) each pair (rcv, j) in G' properly updates the values of $rig[a_{rcv}][j]$ and $rank[a_{rcv}][j]$. Thus in $next(l,t)$, all the pairs in G' are classified on level l and all the pairs in C are classified as before and condition 1 holds in $next(t,l)$. Note that all ranks used by the pairs in G' in classification on level $l + 1$ are "recycled" by them in classification on level l, thus condition 4 holds in $next(l,t)$. Condition 5 holds in $next(l,t)$ since all pairs that are in the same group on level l are classified on the same level (either l or $l + 1$). Let (i,j) be the pair with $rank[a_i][j] = (t+1) \bmod n$. Condition 3 holds in $next(l,t)$ since $rig[a_i][j] = 0$. Condition 2 holds in $next(l,t)$ since (by Lemma 1(1)) also $rng[a_i][j] = 0$, and either $(t+1) = 0$ or positions with $rank < t$ had proper values of rng by induction hypothesis and no variable rng is modified in CASE B. □

By Lemma 2(1), after the REGROUPING PHASE (i.e. before $slot(-1, 0)$), all pairs are classified on level 0, which means the stable ranking of the keys.

4 Sorting

After the ranks of the keys have been computed we may send each *key* with rank r to its destination station $a_{dest(r)}$. The function *dest* should be globally known, however its definition may depend on further applications. For example we may define $dest(r) = \lfloor p \cdot r/n \rfloor$, or $dest(r) = r \bmod p$. Procedure Route-by-ranks performs this task.

```
procedure Route-by-ranks(⟨a_0, ..., a_{p-1}⟩)
begin
    for i ← 0 to n − 1 do
        the (unique) station a_snd containing (unique) j such that i = rank[a_snd][j]
        sends message ⟨x⟩, where x = key[a_snd][j] (if a_dest(i) ≠ a_snd);
        the station a_dest(i) listens (if a_dest(i) ≠ a_snd) and stores x;
end
```

Algorithm 3: Procedure Route-by-ranks

```
procedure Counting-sort(⟨a_0, ..., a_{p-1}⟩,m)
begin
    Counting-rank(⟨a_0, ..., a_{p-1}⟩,m)
    Route-by-ranks(⟨a_0, ..., a_{p-1}⟩)
end
```

Algorithm 4: Procedure Counting-sort

5 Routing

In the case of routing, we assume that each key is a number of the station that is destination of the packet containing this key in the address field of its header. Hence we should route the packets by the keys rather than by the ranks of the keys. However, we use Counting-rank in the preprocessing phase of routing. Besides the ranks $rank[a_i][j]$, we also use the values $rig[a_i][j]$, $gs[a_i][j]$ and $n[a_i]$ computed by Counting-rank. Thus each key is from the set $\{0, \ldots, p-1\}$ and the parameter m (number of bits) is $\lceil \log_2 p \rceil$. After computing the ranks of the keys, the stations perform procedure Compute-intervals. Each station learns time interval in which it should receive its incoming packets in the procedure Finish-routing. The interval for a_i will be stored in variables $i_1[a_i]$ and $i_2[a_i]$. The

packets are then broadcast in the sequence of their ranks. The whole routing is performed by the procedure Route-packets. Note that each packet is transmitted only once (in Finish-routing), while in preprocessing phase the stations transmit only the integers from the range $[0, n]$, which are usually much shorter than the whole packets.

```
procedure Compute-intervals(⟨a_0, ..., a_{p-1}⟩)
begin
    for i ← 0 to p − 1 do
        in time slot 2 · i: begin
            the (at most one) station a_snd containing (unique) j with
            key[a_snd][j] = i and first[a_snd][j] = true sends ⟨x⟩, where
            x = rank[a_snd][j];
            a_i listens and does: if there was a message then i_1[a_i] ← x;
            else i_1[a_i] ← i_2[a_i] ← (−1);
        end
        in time slot 2 · i + 1: begin
            the (at most one) station a_snd containing (unique) j with
            key[a_snd][j] = i and last[a_snd][j] = true sends ⟨x⟩, where
            x = rank[a_snd][j];
            if i_1[a_i] ≠ (−1) then a_i listens and does: i_2[a_i] ← x;
        end
end
```

Algorithm 5: Procedure Compute-intervals

```
procedure Finish-routing(⟨a_0, ..., a_{p-1}⟩)
begin
    for i ← 0 to n do
        in time slot i: begin
            the (unique) station a_snd containing (unique) j with rank[a_snd][j] = i
            sends packet addressed by key[a_snd][j];
            the (unique) a_rcv with i_1[a_rcv] ≤ i ≤ i_2[a_rcv] receives this packet;
            (* should be: rcv = key[a_snd][j] *)
        end
end
```

Algorithm 6: Procedure Finish-routing

6 Complexities of the procedures.

Recall, that for each station a_i, s_i denotes the number of keys initially stored by a_i, r_i is the number of distinct values of these keys, d_i is the number of keys

```
procedure Route-packets(⟨a_0, ..., a_{p-1}⟩)
begin
    (* In each station a_i there are s_i outgoing packets sorted by their destination
    addresses, which are stored in the table key[a_i] *)
    Counting-rank(⟨a_0, ..., a_{p-1}⟩,⌈log_2 p⌉);
    Compute-intervals(⟨a_0, ..., a_{p-1}⟩);
    Finish-routing(⟨a_0, ..., a_{p-1}⟩);
end
```

Algorithm 7: Procedure Route-packets

for which a_i is destination, and q_i is the number of stations that initially stored such keys. Also p is the number of stations, $n = \sum_{i=0}^{p-1} s_i$ and $r = \sum_{i=1}^{p-1} r_i$.

Lemma 3. *For* Init *the energetic cost of listening, for each a_i, is at most* 2 *and the energetic cost of sending is at most* 1*. Time of* Init *is p.*

Lemma 4. *For* Counting-rank*, for each a_i, the energetic cost of listening is at most $2m \cdot r_i + 2$ and the energetic cost of sending is at most $m \cdot r_i + 1$. Time of* Counting-rank *is $m \cdot n + p$.*

Proof. Ranks of the keys from the same group g are continuous in a single station a_i, and all keys with the same value v are always in the same group. For each such group g, a_i has to listen only to the predecessor of its key with the lowest rank in g and to the last element in g, if it is in another station. Similar arguments can be used to estimate the cost of sending. Time complexity is easily seen from the code of the procedure. □

Lemma 5. *For* Route-by-ranks*, for each a_i, the energetic cost of listening is at most d_i and the cost of sending is s_i. The time of* Route-by-ranks *is n.*

Lemma 6. *For* Counting-sort*, for each a_i, the energetic cost of listening is $2m \cdot r_i + d_i + 2$ and the cost of sending is $m \cdot r_i + s_i + 1$. (The total energetic cost is: $3m \cdot r_i + d_i + s_i + 3$.) The time is $m \cdot n + n + p$.*

Lemma 7. *For* Compute-intervals*, for each a_i, the energetic cost of listening is* 2 *and energetic cost of sending is at most $2r_i$. The time is $2p$.*

Lemma 8. *For* Finish-routing*, for each a_i, the energetic cost of listening is d_i, and the energetic cost of sending is s_i. The time is n.*

Lemma 9. *For* Route-packets*, for each a_i, the energetic cost of listening is $2m \cdot r_i + d_i + 4$, and the cost of sending is $m \cdot r_i + s_i + 2r_i + 1$, where $m = \lceil \log_2 p \rceil$ and $r_i \leq p - 1$ (no station sends packets to itself). (Total energetic cost is: $3m \cdot r_i + s_i + d_i + 2r_i + 5$). The time is $m \cdot n + n + 3p$.*

```
procedure Expand-ranks(⟨a_0, ..., a_{p-1}⟩)
begin
    Each a_i, for each 0 ≤ j < s_i, does: first[a_i][j] ← last[a_i][j] ← false;
    Each a_i, for each 0 ≤ j' < r_i, does: begin
        first[a_i][min P(a_i, key'[a_i][j'])] ← first'[a_i][j'];
        last[a_i][max P(a_i, key'[a_i][j'])] ← last'[a_i][j'];
    end
    Each a_i does: lrm[a_i] ← 0;
    for t ← 0 to r − 2 do
        the (unique) a_snd with table rank'[a_snd] containing t, does: begin
            Let j be such that rank'[a_snd][j] = t;
            a_snd assigns sequential ranks lrm[a_snd], ..., lrm[a_snd] + c_{snd,j} − 1 to
            the positions P(a_snd, key'[a_snd][j]) of the table rank[a_snd], where
            c_{snd,j} = |P(a_snd, key'[a_snd][j])|;
            a_snd sends message ⟨x⟩, where x = lrm[a_snd] + c_{snd,j};
        end
        the (unique) a_rcv with table rank'[a_rcv] containing t + 1, receives ⟨x⟩ and
        does: lrm[a_rcv] ← x;
    the (unique) a_snd with table rank'[a_snd] containing r − 1, does: begin
        Let j be such that rank'[a_snd][j] = r − 1;
        a_snd assigns sequential ranks lrm[a_snd], ..., lrm[a_snd] + c_{snd,j} − 1 to the
        positions P(a_snd, key'[a_snd][j]) of the table rank[a_snd], where
        c_{snd,j} = |P(a_snd, key'[a_snd][j])|;
        a_snd sends message ⟨x⟩, where x = lrm[a_snd] + c_{snd,j}.
    end
    each a_i ≠ a_snd listens to ⟨x⟩;
    each a_i does: n[a_i] ← x;
end
```

Algorithm 8: Procedure Expand-ranks

7 Accelerating Counting-rank.

Time complexity of Counting-rank contains component $m \cdot n$. Note that, in the case of routing, $r \leq p(p-1)$ (each station sends packets to at most $p-1$ other stations) and $m = \lceil \log_2 p \rceil$. We may expect that n is much larger than r. In this section we show how to replace this component with $(m+1) \cdot r$ while the energetic cost for each a_i is increased by at most $2r_i + 1$. By Compressed-counting-rank we denote the procedure Counting-rank with the code modified as follows: Each station a_i pretends that it contains only one key of given value (i.e. it uses key', $rank'$, $last'$, $first'$, r_i and r instead of key, $rank$, $last$, $first$, s_i and n, respectively.) The code of the sub-procedure Init is modified the same way. The computed results are stored in $rank'$, $first'$ and $last'$. At the end of Compressed-counting-rank we add procedure Expand-ranks (Algorithm 8) which computes ranks and proper values $first$ and $last$, for all keys, and proper value of n in each station. Let $P(a_i, k) = \{j \mid key[a_i][j] = k\}$. The time of Expand-ranks is r and its energetic cost for each a_i is $2r_i + 1$. (Each a_i may need to listen and send at most once for each its key value and listen to the last message.) Let us call the resulting algorithm Accelerated-Routing.

Lemma 10. *For* Accelerated-Routing*, for each a_i, the energetic cost of listening and sending is $3\lceil \log_2 p \rceil \cdot r_i + s_i + d_i + 4r_i + 6$, where $r_i \leq p-1$. The time is $\lceil \log_2 p \rceil \cdot r + n + r + 3p$.*

References

1. Amitava Datta and Albert Y. Zomaya. An Energy-Efficient Permutation Routing Protocol for Single-Hop Radio Networks. IEEE Trans. Parallel Distrib. Syst., 15:331-338, 2004.
2. M. Kik. Merging and Merge-sort in a Single Hop Radio Network. SOFSEM 2006, LNCS 3831, pp. 341-349, 2006.
3. M.Kik. Sorting Long Sequences in a Single Hop Radio Network. MFCS 2006, LNCS 4162, pp. 573-583, 2006.
4. K. Nakano. An Optimal Randomized Ranking Algorithm on the k-channel Broadcast Communication Model. ICPP 2002, pp. 493-500, 2002.
5. K. Nakano, S. Olariu, A. Y. Zomaya. Energy-Efficient Permutation Routing in Radio Networks. IEEE Transactions on Parallel and Distributed Systems, 12:544-557, 2001.
6. K. Nakano, S. Olariu, A. Y. Zomaya. Energy-Efficient Routing in the Broadcast Communication Model. IEEE Trans. Parallel Distrib. Syst., 13:1201-1210, 2002.
7. M. Singh and V. K. Prasanna. Optimal Energy Balanced Algorithm for Selection in Single Hop Sensor Network. SNPA ICC, May 2003.
8. M. Singh and V. K. Prasanna. Energy-Optimal and Energy-Balanced Sorting in a Single-Hop Sensor Network. PERCOM, March 2003.
9. Compendium of Large-Scale Optimization Problems. (DELIS, Subproject 3). `http://ru1.cti.gr/delis-sp3/`