Distributed Construction and Maintenance of Bandwidth and Energy Efficient Bluetooth Scatternets

Metin Tekkalmaz, Hasan Sözer, and Ibrahim Korpeoglu, *Member*, *IEEE*

Abstract—Bluetooth networks can be constructed as piconets or scatternets depending on the number of nodes in the network. Although piconet construction is a well-defined process specified in Bluetooth standards, scatternet formation policies and algorithms are not well specified. Among many solution proposals for this problem, only a few of them focus on efficient usage of bandwidth in the resulting scatternets. In this paper, we propose a distributed algorithm for the scatternet formation problem that dynamically constructs and maintains a scatternet based on estimated traffic flow rates between nodes. The algorithm is adaptive to changes and maintains a constructed scatternet for bandwidth-efficiency when nodes come and go or when traffic flow rates change. Based on simulations, the paper also presents the improvements in bandwidth-efficiency and reduction in energy consumption provided by the proposed algorithm.

Index Terms—Bluetooth, scatternet formation, ad hoc networks, network topology, algorithm design, distributed computing.

1 Introduction

Bluetooth [1] is a short range wireless RF technology designed initially for cable replacement at indoor places, but also supports usage scenarios for personal area or local area networking. Its low cost, low power consumption, and ad hoc connectivity features make it a good wireless connectivity choice for mobile devices due to their anytime-anywhere connection requirement and limited battery power.

Bluetooth devices are able to form small networks, which are called piconets, with up to eight nodes. A piconet consists of a master node and one or more slave nodes. The scheduling of packets into the common FHSS radio channel of a piconet is coordinated by the master node via a polling based TDMA scheme. No direct communication between any two slave nodes is allowed within a piconet. Data traffic among slave nodes has to go through the master node.

Bluetooth standards also specify some set of mechanisms to construct larger networks called scatternets. A scatternet is actually nothing but a network consisting of multiple piconets with common nodes between them, called bridges, to forward traffic between those piconets. Since all members of a piconet follow the same pseudorandom hopping sequence over some set of predefined RF channels specified in Bluetooth standards, and since each piconet has a different hopping sequence, a bridge node follows the hopping sequence of different piconets, which it belongs to at different times.

Although piconet construction is a well-defined process described in current Bluetooth standards, scatternet formation algorithms and policies are not specified with enough detail. Therefore, this is an open research area and many methods have been proposed so far for the scatternet formation problem. Once the scatternet is constructed, maintaining it in case of new node arrivals and node departures is another issue, which is not addressed much in the literature. Additionally, despite numerous works on scatternet formation, there is only a few studies ([2], [3], and [4]) that focus on constructing a scatternet with the aim of having the bandwidth capacity of the resulting scatternet utilized as efficiently as possible considering the traffic demands of the nodes constituting the scatternet. These studies all propose static (i.e., do not handle new node arrivals and node departures) and centralized solutions.

In this paper, we provide an algorithm that dynamically constructs a scatternet and concurrently modifies it to make it more bandwidth-efficient. In order to reduce the traffic load on a scatternet and thereby effectively utilize the capacity of the scatternet, our approach is based on reducing the path lengths between highly communicating nodes, as the approach followed in [4]. Different from [4], however, our algorithm works in a distributed manner with no central coordination and without requiring the availability of global topology and traffic demand information in advance. Furthermore, it is dynamic (i.e., new node arrivals and node departures are handled appropriately) and it can adapt to changes in the number of traffic flows and their rates, consequently preserving bandwidth-efficiency. Moreover,

- *M. Tekkalmaz is with ASELSAN A.S. MST-YMM-REH, Mehmet Akif Ersoy mh., 16. cad. No:16, Yenimahalle, Ankara, Turkey. E-mail: metint@cs.bilkent.edu.tr.*
- *H. Sözer is with the Software Engineering Research Group, University of Twente, Faculty of Electrical Engineering, Mathematics, and Informatics, PO Box 217 7500 AE, Enschede, The Netherlands. E-mail: sozerh@ewi.utwente.nl.*
- *I. Korpeoglu is with the Department of Computer Engineering, Bilkent University, TR-06800, Ankara, Turkey. E-mail: korpe@cs.bilkent.edu.tr.*

Manuscript received 15 Feb. 2005; revised 31 May 2005; accepted 2 Sept. 2005; published online 26 July 2006.
Recommended for acceptance by I. Stojmenovic, S. Olariu, and D. Simplot-Ryl.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-0128-0205.

messaging overhead imposed by the algorithm is kept as low as possible.

We evaluated our algorithm by way of simulation experiments to observe how it affects the efficient use of bandwidth in a scatternet. For the evaluation, we used a metric called *weighted average shortest path* (WASP) which is introduced in [4] and that indicates the amount of bandwidth usage in a scatternet for a given traffic demand. Our simulation results show that the use of our algorithm in a scatternet results with up to around 45 percent improvement in the value of the WASP metric, implying a significant improvement in the bandwidth-efficiency in a scatternet. We also investigated how energy consumption in a scatternet is affected by our algorithm. Besides resulting with more available bandwidth and less delay in communication, which are immediate consequences of efficient bandwidth usage, the use of our algorithm also causes reduction in the total energy spent in a scatternet and the energy spent in the individual nodes. Hence, our algorithm also constructs and maintains a scatternet in an energy-efficient way.

The remainder of the paper is organized as follows: In the next section, related previous studies are summarized. In Section 3, a scatternet construction algorithm is described. In Section 4, evaluation criteria for bandwidth efficiency are explained and simulation results, based on these criteria, are presented. Finally, some future work issues are discussed and the paper is concluded in Section 5.

2 Related Work

There have been many solution proposals for the Bluetooth scatternet formation problem [5]. Most of these proposals focus on the efficiency of the scatternet formation algorithm, considering the duration of the construction process and number of messages exchanged during the construction, as it is the case in [6], [7], and [8]. Some of them also apply heuristics in order to make the constructed scatternet efficient in terms of some metrics. One such heuristic that is followed by some studies is to keep the number of piconets as small as possible in the resulting scatternet with the goal of decreasing the amount of interpiconet traffic and interference. There exist other studies, which focus on different aspects of the problem. In [9] and [10], for instance, tree-shaped topologies are formed for the ease of routing. Some studies, like [11], on the other hand, aim to form fault-tolerant topologies by means of constructing alternative paths between nodes.

Although various proposals have been made to form scatternets with various objectives, algorithms that favor the efficient usage of bandwidth are not widely studied. In [12], as one of the earliest studies, traffic patterns among the piconet members is taken into consideration while selecting the master of the piconet. Some studies on constructing efficient scatternet topologies are [2], [3], and [4]. All these three proposals are single-hop, centralized, and static solutions. Baate et al. [2] propose a graph theoretical solution which uses 1-factors to construct a scatternet. Marsan et al. [3] approach to the problem as a min-max optimization. Topal [4], on the other hand, uses a set of heuristics to designate the locations and the roles of the nodes, assuming that the traffic flow information is known a priori.

There are also proposals for the scatternet formation problem that are distributed and dynamic. They are dynamic in the sense that they can handle node arrivals and departures. However, they do not consider traffic dynamics and trigger maintenance procedures when traffic conditions change. In [13], where a distributed and multi-hop algorithm is provided, the focus is on reducing the scatternet formation time, and a scatternet is not maintained based on traffic conditions. Also, the scatternet topology has to be a tree, which is not a requirement in our solution. Chiasserini et al. [14] propose a distributed and dynamic algorithm that resembles our work in considering traffic conditions for re-arranging a scatternet. However, the goal is not increasing the bandwidth efficiency, but reducing the load on the most congested node. It also provides an optimal topology in that respect but achieves this through a centralized algorithm. As another difference, we base our solution on an estimation of traffic load at a finer granularity, whereas [14] assumes that the traffic load on a node is classified as either low or high.

Along with the scatternet formation proposals which consider the relation between topology and efficiency, there are studies that try to reveal the relation between them, such as [15], [16], and [17]. Kapoor et al. [16] apply an analytical approach, whereas Miorandi et al. [17] try to establish a mathematical foundation to determine the relations between the topology and network capacity. On the other hand, Miklos et al. [15] apply a statistical approach in order to investigate the effects of topology on performance.

3 The Construction Algorithm

We now describe, in this section, our distributed solution for bandwidth and energy efficient scatternet construction and maintenance. Our solution requires two main procedures, *Link Establishment* and *Maintenance* procedures, to be executed *concurrently* in a scatternet. The *Link Establishment* procedure handles new node arrivals (i.e., new connections) and node departures (i.e., disconnections), hence, incrementally constructing a scatternet. On the other hand, the *Maintenance* procedure works on the base topology constructed by the *Link Establishment* procedure and maintains the bandwidth-efficiency of the scatternet by modifying it.

The *Maintenance* procedure constantly collects information about the traffic flowing in the scatternet and runs a set of operations based on this information when *significant* changes in the traffic pattern and rates are observed. These operations are performed in order to modify the scatternet topology so that the node pairs whose communication demands between each other are relatively higher are placed closer to each other in the scatternet.

The *Maintenance* procedure depends on some assumptions. The first assumption is that all the nodes which are going to be part of the scatternet are in the communication range of each other. Another assumption is that the topology on which the *Maintenance* procedure works has the following properties:

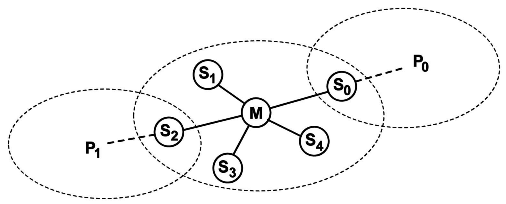


Fig. 1. A sample piconet on which traffic data is gathered.

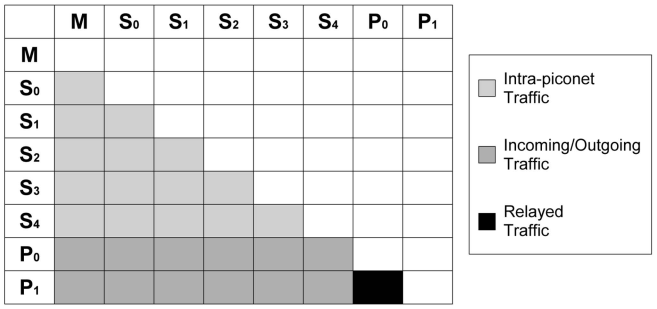
- There are only slave/slave types of bridges (i.e., there is no master/slave type of bridge) and
- a slave/slave type of bridge connects exactly two piconets (i.e., cannot connect to more than two different master nodes).

The *Link Establishment* procedure takes these restrictions into account while establishing new links and the *Maintenance* procedure preserves these properties while modifying the topology.

The restrictions on the properties of the topology can be eliminated with slight modifications on the operations of the *Maintenance* procedure. However, they are preferred to exist for the efficiency of the resulting scatternet topology. As studied in [18], master/slave bridges constitute a burden for the load balancing and simultaneous communication in different piconets. When a master/slave bridge switches from the master role in a piconet to the slave role in another piconet, all traffic in the piconet where the bridge was the master earlier has to be stopped, which prevents all members of that piconet from sending or receiving data. This implies an inefficient use of channel bandwidth for that piconet and long packet delays for the sessions using that piconet. Additionally, the switching time of bridges cannot be underestimated and sharing a bridge's time for more than two piconets would make that bridge a bottleneck. It is also pointed out by other studies, described in [19], that a master/slave bridge ceases intrapiconet communication, while participating in another piconet and bridge degree has a direct impact on switching overhead. Consequently, master/slave bridges are not favored if a special type of topology (i.e., tree) is not an issue and almost all solution proposals for the Bluetooth scatternet formation problem put restrictions on bridge degree [5]. The drawback of such restrictions is the limitation that they impose on the variety of topology configurations. Another approach would be not to put any restriction on bridge degrees and to leave the switching overhead problem to the scheduling algorithm. Hence, the number of different topologies would be increased.

We also assume that the routing protocol running over a scatternet constructed by our algorithm selects paths with minimum number of hops between any two communicating nodes. The proposed algorithm can still work with a routing protocol that does not satisfy this condition, but such a routing protocol contradicts with the basic approach of the algorithm and cost metrics used for evaluation.

In the following section, we explain the method we use to estimate the traffic flow pattern and rates. Then, in Section 3.2, we describe several operations that are used to maintain a scatternet and that make use of the acquired traffic flow information and, in Section 3.3, we describe how



	M	S_0	S_1	S_2	S_3	S_4	P_0	P_1
M								
S_0	░							
S_1	░	░						
S_2	░	░	░					
S_3	░	░	░	░				
S_4	░	░	░	░	░			
P_0	▒	▒	▒	▒	▒	▒		
P_1	▒	▒	▒	▒	▒	▒	■	

░ Intra-piconet Traffic

▒ Incoming/Outgoing Traffic

■ Relayed Traffic

Fig. 2. A traffic table that is maintained at the master node of a piconet.

these several operations are combined together to define the *Maintenance* procedure. Section 3.4 provides the details of *Link Establishment* procedure.

3.1 Estimating the Traffic Pattern

Since the scatternet topology is modified according to the communication demands between nodes, traffic flow pattern and rates must be estimated. We identify three types of traffic in a piconet about which we collect information:

- **Intrapiconet Traffic:** This is the traffic flowing between members of the piconet (i.e., traffic due to master-slave and slave-slave communication).
- **Incoming/Outgoing Traffic:** This is the traffic flowing between members of the piconet and the neighboring piconets.
- **Relayed Traffic:** This is the traffic received from a neighboring piconet and forwarded to another neighboring piconet.

Consider the piconet shown in Fig. 1. It consists of a total of six members: a master node M; two slave/slave bridges, S_0 and S_2, connected to the neighboring piconets P_0 and P_1, respectively; and three slave nodes, S_1, S_3, and S_4. For such a piconet, a traffic table, as illustrated in Fig. 2, is constructed and maintained at the master node M.

Each cell in the table is used to store the rate of traffic observed between the entities that match the corresponding row and column. As indicated with the shaded cells in the figure, half of the table is used, since we are only interested in the rate of total traffic flowing between any two nodes irrespective of its direction. Therefore, we store the total rate of traffic sent and received between any two nodes as a single entity in the table.

Traffic within a piconet has to pass through the master node of the piconet and, therefore, almost all information required to generate the traffic table is available at the master node without any significant extra messaging. Information exchange is only needed for obtaining the rate of traffic flowing between bridge nodes and the corresponding neighboring piconets. In the case shown in Fig. 1, for instance, S_0 and S_2 should send traffic flow information between themselves and piconets P_0 and P_1, respectively, to the master node M.

The rate of traffic flow may change rapidly, which would lead to a constant alteration of the scatternet topology. In order to prevent that, a discrete-time aging method shown in (1) can be used while collecting traffic information. This method smoothly integrates changes in the instantaneous traffic rate, R_{inst}, to the average traffic rate, R_{avg}. In (1), the effect of the instantaneous traffic rate to the average traffic rate can be adjusted by varying the α parameter between 0 and 1:

$$R_{avg}(t) = \alpha \times R_{avg}(t-1) + (1-\alpha) \times R_{inst}(t). \tag{1}$$

How the structure of the traffic table is changed due to topology changes in a piconet is described in Section 3.4.

3.2 Operations

The *Maintenance* procedure is composed of a set of operations, which are explained in the following sections.

3.2.1 Master/Bridge Reassignment

This operation reassigns master and bridge roles to the members of a piconet so that the cost function introduced with (2), (3), and (4) is minimized. In these equations, n is the number of slaves in the piconet (including the bridges) where we perform this operation, and m is the number of neighboring piconets. **T** represents the traffic table, while subscripts identify row and column indices.

$$C_p = C_{intra-p} + C_{inter-p}, \tag{2}$$

$$C_{intra-p} = \sum_{i=0}^{n-1} \mathbf{T}_{S_i M} + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 2 \times \mathbf{T}_{S_i S_j}, \tag{3}$$

$$C_{inter-p} = \sum_{i=0}^{m-1} \mathbf{T}_{P_i M} + \sum_{i=0}^{m-1} \sum_{\substack{j=0 \wedge \\ S_j \neq bridge\,To_{P_i}}}^{n-1} 2 \times \mathbf{T}_{P_i S_j}. \tag{4}$$

The cost function (C_p) has two components: cost of intrapiconet traffic ($C_{intra-p}$) and cost of interpiconet traffic ($C_{inter-p}$). In order to calculate the cost, traffic flow rates between communicating entities multiplied by the length of corresponding communication paths are summed up. Since all traffic flow passes through the master node, traffic flow rates of slave/slave and slave/piconet communication are multiplied by two. There is no cost of communication incurred on the piconet of interest for traffic flowing between a neighboring piconet and the corresponding bridge node ($bridgeTo_P$), hence, such traffic amounts are excluded in the calculation of $C_{inter-p}$. Similarly, the cost of relayed traffic is not included in the cost function, since it is not affected by the selection of different master and bridge nodes.

Algorithm 1 describes the execution steps of the *Master/Bridge Reassignment* operation. The nested loops generate all possible master and bridge assignments and, for each such assignment, the cost function, C_p, is calculated. If the current cost is the minimum among the ones calculated by now, the current master/bridge configuration is saved. At Line 26, we have the master/bridge configuration that minimizes the bandwidth usage within the piconet and roles are reassigned to the nodes according to it. Since there are n choices for the master node, and m out of $n-1$ nodes will be chosen as bridges and these bridges can be assigned to neighboring piconets in $m!$ ways, C_p is calculated $n \times C(n-1, m) \times m!$ times.

If the master node changes because of the operation, the traffic table is transferred to the new master node where the *Maintenance* procedure will be executed next.

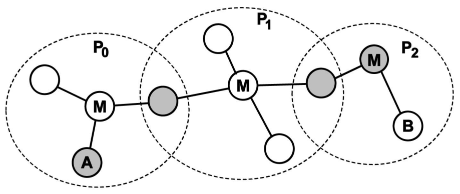


Fig. 3. Global effect of Master/Bridge Reassignment.

Algorithm 1 Master/Slave Reassignment

```
1. $min\text{-}C_p \leftarrow \infty$
2. $min\text{-}master \leftarrow$ NULL
3. $min\text{-}bridge_0 \leftarrow$ NULL
4. $min\text{-}bridge_1 \leftarrow$ NULL
5. ...
6. $min\text{-}bridge_m \leftarrow$ NULL
7. for all Nodes $M$ in the piconet do
8.  for all Permutations of neighboring piconets:
    $P_a P_b \ldots P_x$ do
9.   for all $m$ out of $n-1$ combinations of nodes, excl. $M$:
     $S_k S_l \ldots S_y$ do
10.    $bridge_a \leftarrow k$
11.    $bridge_b \leftarrow l$
12.    ...
13.    $bridge_x \leftarrow y$
14.    Calculate $C_p$
15.    if $C_p < min\text{-}C_p$ then
16.     $min\text{-}C_p \leftarrow C_p$
17.     $min\text{-}master \leftarrow M$
18.     $min\text{-}bridge_0 \leftarrow P_a : S_k$
19.     $min\text{-}bridge_1 \leftarrow P_b : S_l$
20.     ...
21.     $min\text{-}bridge_m \leftarrow P_x : S_y$
22.    end if
23.   end for
24.  end for
25. end for
26. Rearrange master and bridge roles in the piconet
     according to the values of $min\text{-}master$, $min\text{-}bridge_0$,
     $min\text{-}bridge_1$, ..., and $min\text{-}bridge_m$
```

Although this operation rearranges roles locally inside a piconet, it also has a global effect as illustrated in Fig. 3. Suppose that nodes labeled as A and B in the figure communicate heavily with each other. The master node of P_0 will assign node A as the bridge node for P_1, since cost function will be minimized in this way because of the high traffic flow rate between node A and piconet P_1. Afterward, the master node of P_1 will notice a high traffic flow rate between the new bridge node A and piconet P_2. As a result of the *Master/Bridge Reassignment*, node A will now be given role as the bridge node between piconets P_1 and P_2. After node A becomes a member of piconet P_2 and the master node of P_2 executes the *Master/Bridge Reassignment*, A will be the new master node in piconet P_2 and the length of the communication path between nodes A and B will be reduced to one.

This is just a sample scenario and it might have been the case that B approaches A, or they could have met at piconet P_1, depending on the order of execution of the *Master/Bridge*

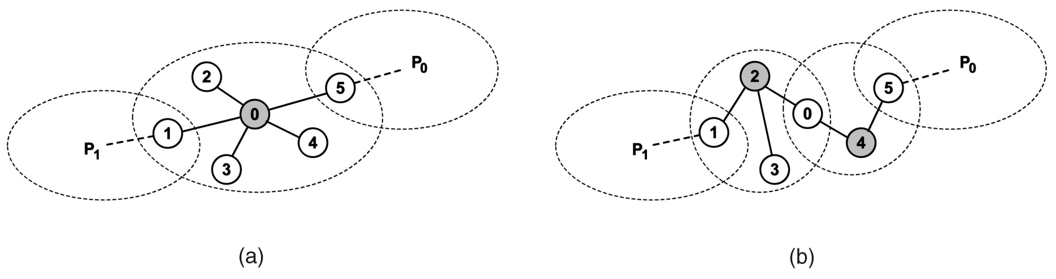


Fig. 4. Piconet Division operation.

Reassignment operation by the master nodes. At the end, however, A and B will be closer to each other in the scatternet as a result of successive executions of the *Master/Bridge Reassignment* operation in different piconets.

The *Master/Bridge Reassignment* operation improves the bandwidth usage in the scatternet due to the following reasons in which P is the piconet where the operation is executed and N is the set of neighboring piconets of P.

- *For each* piconet p *s.t.* $(p \neq P) \wedge (p \notin N)$, p is not affected by the execution of this operation, since none of its members are changed and the amount of traffic that is sent, received, or forwarded is not affected by the operation.
- *For each* piconet p *s.t.* $p \in N$, p is not affected by the execution of this operation, since the amount of traffic that is sent, received, or forwarded is unchanged. From the viewpoint of p, the only difference could be the replacement of the bridge between p and P with another member of P. However, traffic that is sent to or received from new bridge and the previous one will be the same.
- Bandwidth usage is improved in P since the roles are rearranged to reduce the cost of communication.

3.2.2 Piconet Division

This operation splits a piconet into two in order to increase available bandwidth per piconet whenever necessary. Similar to *Master/Bridge Reassignment*, in the *Piconet Division*, for all possible role assignments to the nodes that will constiute two piconets (out of one), the value of a cost function is calculated and the assignment minimizing the cost is selected. The cost function used in *Piconet Division* operation is similar to (3) and (4) in the sense that it sums up the end-to-end traffic flow rates between node pairs multiplied by the corresponding communication path lengths, for all such pairs. However, the resulting topology of *Piconet Division* may have either one, two, three, or four-hop paths, since we now have two piconets for which we are trying to find the best role assignment, whereas hop distances are either one or two in the *Master/Bridge Reassignment* operation. In Fig. 4, a piconet is split into two piconets.

The *Piconet Division* operation requires that the piconet has at least two slaves which are not bridges currently. Such a constraint is necessary because, as a result of the operation, a node will be required to be the master in the offspring piconet (number of piconets increased from one to two) and another node will be required to be the bridge to connect the two piconets. Other than these, there must be enough slave nodes to communicate with neighboring piconets.

The *Piconet Division* operation improves the bandwidth usage in the scatternet by increasing the total amount of bandwidth that is available in the scatternet, since each piconet adds extra bandwidth to the scatternet. Splitting a piconet is not preferred unless the bandwidth in the piconet is insufficient for intrapiconet, interpiconet, and forwarded traffic, because it increases the communication path lengths unlike other operations described in the paper. The *Piconet Division* operation is executed whenever the bandwidth is not sufficient in a piconet and the operation favors the topology where the newly added bandwidth is used most efficiently.

3.2.3 Slave Transfer

This operation transfers a slave node of a piconet to a neighboring piconet if it is beneficial to do so and if the neighboring piconet can accept it. In order to determine whether a transfer of a slave is beneficial or not, gain and cost of the transfer have to be computed first. Equations (5) and (6) show how the gain, G_{ST}, and cost, C_{ST}, of a slave transfer is calculated. In the equations, n denotes the number of slave nodes in the piconet and m denotes the number of neighboring piconets.

$$G_{ST} = \mathbf{T}_{S_s P_p} \tag{5}$$

$$C_{ST} = \mathbf{T}_{S_s M} + \sum_{\substack{i=0\wedge\\ i\neq p}}^{m-1} \mathbf{T}_{S_s P_i} + \sum_{\substack{j=0\wedge\\ j\neq s\wedge\\ S_j\neq bridgeTo_{P_p}}}^{n-1} \mathbf{T}_{S_s S_j}. \tag{6}$$

The gain and cost of transfer is calculated for each nonbridge slave node, S_s, in the piconet and for each neighboring piconet, P_p, that has less than eight members. The gain, G_{ST}, is the traffic flow rate between S_s and P_p. The cost of transferring a slave, C_{ST}, is the summation of traffic flow rates between the slave to be transferred and

- the master node of the piconet the slave belongs to before transfer,
- neighboring piconets other than the candidate target piconet, and
- other slave nodes except the bridge node connected to the candidate piconet, which we denote as $bridgeTo_{P_p}$.

In fact, G_{ST} and C_{ST} should be multiplied by two, because the transfer of a slave node increments or decrements

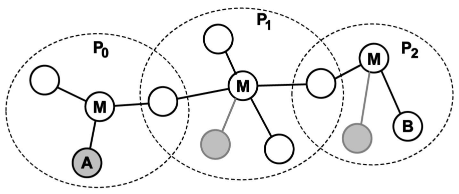


Fig. 5. Global effect of the Slave Transfer operation.

routing paths by two hops. However, they are omitted, since this is the case for both cost and gain.

G_{ST} and C_{ST} values are calculated for all S_s $(0 \leq s \leq n-1)$ and P_p $(0 \leq p \leq m)$ pairs beforehand. Transfer of a slave S_s to a piconet P_p for which $G_{ST} - C_{ST}$ is positive is marked as a beneficial transfer. The greater this value is, the more the transfer is beneficial. After the determination of beneficial transfers, each transfer is performed one by one starting from the most beneficial one down to the least beneficial one, unless for an (S_s-P_p) pair, slave S_s has already been transferred to another piconet, since it was more beneficial, or P_p has no more room for new slaves.

Although this operation affects two piconets, successive execution of the operation also has a global effect as depicted in Fig. 5. Suppose nodes labeled as A and B communicate heavily with each other. The master node of piconet P_0 will notice that A communicates with piconet P_1 more than it communicates with the members of piconet P_0 and, therefore, the master node will eventually transfer node A to piconet P_1. After that, piconet P_1 will eventually notice a high traffic flow rate between node A and piconet P_2, and now it will transfer the node A to piconet P_2. In this way, nodes A and B will come together in the same piconet.

The *Slave Transfer* operation improves the bandwidth usage in the scatternet due to following statements in which P is the piconet where the operation is executed, hence the one that the node is transferred from, and D is the set of neighboring piconets to which at least one slave node is transferred.

- *For each* piconet p *s.t.* $(p \neq P) \wedge (p \notin D)$, p is not affected by the execution of this operation, since its member set remains unchanged and the amount of traffic that is sent, received, and forwarded is not affected by the operation.
- *For each* piconet pair (P, p) *s.t.* $p \in D$, the bandwidth usage is improved, since a node transfer is scheduled only if the traffic sent and received by the transferred node generates less load on the topology after the transfer compared to the topology before the transfer.

3.2.4 Piconet Merge

This operation creates one piconet out of two neighboring piconets. Algorithm 2 outlines the execution steps of the operation. The master node checks all the neighboring piconets if merging with one of them is feasible or not. Merge is feasible if the total number of nodes in the resulting piconet would not exceed 8, and if the total traffic flow rate in the resulting piconet would not exceed the 1 Mbps raw capacity of a Bluetooth piconet (or a practical upper bound). The 8-node limit check is performed first, since it is simpler. If the 8-node limit is satisfied, the traffic tables of the piconets that are candidates for merge are combined into a single table, to check whether the traffic flow rate is less than 1-Mbps in the resulting piconet. If this is also true, all nodes of the neighboring piconet are assigned as slave nodes to the master node, executing the operation. Thereafter, *Master/Bridge Reassignment* is executed according to the new traffic table, which is constructed out of two old tables, in order to assign roles in the newly constructed piconet.

Algorithm 2 Piconet Merge

```
1. for all Neighboring piconets $p$ do
2.  if (Total # of nodes in current piconet
          + Total # of nodes in $p$) $< 8$ then
3.   Merge traffic tables
4.   if Total traffic $< 1$ Mbps then
5.    Assign all nodes of $p$ as a slave to current piconet
6.    Execute Master/Bridge Reassignment operation
7.    Terminate
8.   end if
9.  end if
10. end for
```

The *Piconet Merge* operation can be thought as the reverse operation of *Piconet Division*. The topology shown in Fig. 4b, for example, turns into the topology shown in Fig. 4a as a result of the execution of this operation.

Merging the traffic tables of two candiate piconets, which is an important step of the procedure, is not a straightforward task because the traffic table of the resulting piconet cannot be generated out of the traffic tables of these piconets. As an example, in Fig. 4b, the traffic flow rate between nodes *1* and *5* cannot be obtained from traffic tables stored at nodes *2* and *4*. Such information could be obtained by keeping and maintaining extra information and by exchanging extra messages. However, we apply a heuristic for completing the unknown parts of the merged traffic table: Envision two neighboring piconets P_0 and P_1 that are going to merge. From the traffic table of P_1, we know the amount of traffic flow between P_1 and P_0. However, we do not know how much of it is destined to (or originated from) P_0, and how much of it is relayed at P_0. But, from the traffic table of P_0, we also know what ratio of traffic flow between P_0 and P_1 is generated or consumed by the members of P_0. Combining this information, we can estimate the amount of traffic flow between P_1 and the members of P_0 assuming the traffic is equally shared by the piconet members.

The *Piconet Merge* operation improves the bandwidth usage in the scatternet due to following statements in which P_1 and P_2 are the piconets that are merged, whereas N_1 and N_2 are the sets of neighboring piconets of P_1 and P_2, respectively. P is the piconet formed as a result of the merge operation and N is the set of its neighbors satisfying $N = N_1 \cup N_2$.

- *For each* piconet p *s.t.* $(p \neq P) \wedge (p \notin N)$, p is not affected by the execution of this operation, since none of their members are changed and the amount of traffic that is sent, received, and forwarded is not affected by the operation.

- *For each* piconet p *s.t.* $p \in N$, p is not affected since the amount of traffic that is sent, received, and forwarded is still the same, although one node at each neighboring piconet is possibly replaced by the execution of this operation.
- The bandwidth usage is improved in the network constituted by the members of P, since the communication cost is lower after the operation.

3.3 Maintenance Procedure

As indicated in the previous sections, the *Maintenance* procedure is a combination of a set of operations which we have described already. In this section, we present a fusion algorithm that combines these operations together and establishes an execution relation among them.

The fusion algorithm, shown in Algorithm 3, is executed at a master node of a piconet after a warmup period when enough information is ready in the traffic table if it will be the first execution, whenever a *significant* change is observed in the traffic table maintained at that master, or when the total traffic that has to flow in the corresponding piconet is close to the 1 Mbps capacity of the piconet. When total traffic demand in a piconet is expected to exceed 1 Mbps, the piconet should be split into two piconets in order to increase the capacity. When this is not the case, however, as long as the bandwidth constraints are satisfied, merging piconets as much as possible is beneficial for the sake of shortening paths and decreasing the number of bridges, which have to switch between piconets.

Algorithm 3 The Fusion Algorithm

```
1. Execute Master/Bridge Reassignment operation
2. if Current node is still master then
3.   Execute Slave Transfer operation
4.   if Total piconet traffic > 1 Mbps then
5.     Execute Piconet Division operation
6.   else
7.     Execute Piconet Merge operation
8.   end if
9. end if
```

Significant change is supposed to occur when the value of *totalDiff* calculated as in (7) exceeds a threshold value. Adjustment of the threshold would affect the sensitivity of the *Maintenance* procedure to the changes in the traffic flow rate. In the equation, basically, the sum of differences between the traffic flow rates for each pair of communicating entities (i.e., master node, slave nodes, and neighboring piconets) represented in the current traffic table and the traffic table that was used in the last execution of the fusion algorithm is calculated.

$$totalDiff = \sum_{i=0}^{t-1} \sum_{j=i+1}^{t-1} \left| T_{ij} - T_{last_{ij}} \right|. \qquad (7)$$

As mentioned in Section 3.1, the messaging required to collect information about the traffic characteristics is rather low, since the information is collected at the master node, where almost all the traffic related to the piconet passes through. Similarly, the operations constituting the *Maintenance* procedure have low messaging demands, since the messaging required during the execution of an operation occupies only the nearby links with a low bandwidth requirement. For example, during the *Slave Transfer* operation the master node that initiates the transfer communicates only with the node to be transferred and the master node that accepts the transferred node. As another example, during the *Master/Bridge Reassignment* operation, the master node initiating the reassignment communicates only with its slave nodes to inform them about their new roles and with the master nodes of the neighboring piconets to inform them about the new bridge nodes. Likewise, during the *Piconet Merge* and *Piconet Division* operations, the communication between the master node initiating the operation and other nodes that have an active role in the operation is kept within the same set of nodes, leaving the rest of the nodes unaffected as far as the bandwidth required during the operation is concerned.

The fusion algorithm is expected to yield improvement in the bandwidth usage of the scatternet, since the operations which constitute the fusion algorithm are expected to yield improvement in the bandwidth usage whose reasons are explained in the sections in which the operations are described. In general, all operations shorten the paths between talkative nodes except the *Piconet Division* operation, which improves the feasibility of satisfying the requested traffic demands using the constructed scatternet. Gathering talkative nodes together as much as possible would not make any sense if the communication demands exceeds the practical limits. In such cases, the *Piconet Division* operation splits the piconet so that the capacity of the scatternet is increased. This lengthens the communication paths, conflicting with the aim of other operations, but it is a necessity to reduce the amount of unsatisfied demands.

A master node should ensure that it is the only one which is going to execute the fusion algorithm among the master nodes of neighboring piconets to prevent possible conflicting role and topology changes. It should be noted that, conflicting, here, means different decisions, all of which improve the performance of the scatternet as far as bandwidth usage is considered. Hence, coordination is required among the master nodes of neighboring piconets to avoid concurrent execution of the fusion algorithm and to provide sequential execution and iterative improvement. However, such a coordination mechanism is not described here, since it is considered out of scope of this paper.

3.4 Link Establishment Procedure

The links in Bluetooth are established after inquiry and page steps. Inquiry and inquiry-scan modes of inquiry step play a key role for device discovery and determination of master/slave roles. The *Link Establishment* procedure in our proposal specifies whether a node having a certain role (i.e., master, bridge, slave, or free-node) can switch to inquiry or inquiry-scan modes. This specification controls the discoverability of nodes and, in this way, the established links are forced to meet the restrictions described in Section 3. Table 1 shows which modes of inquiry step are allowed in which roles of nodes in a scatternet.

TABLE 1
Modes Assigned to Roles at Inquiry Step

Role	Inquiry	Inquiry-Scan
Free-Node	Yes	Yes
Master	Yes	No
Slave	No	Yes
Bridge	No	No

Because of the mode assignments shown in Table 1, link establishments are restricted to happen only between pairs of nodes having the following role combinations: master/slave, where slave becomes bridge; master/free-node, where free-node becomes slave; slave/free-node, where free-node becomes master; and free-node/free-node, where one of them becomes master and the other becomes slave. Bridge nodes cannot establish new links. As a result, there is no master/slave bridges and a bridge node has exactly two masters as required by the *Maintenance* procedure.

To handle disconnections, the *Link Establishment* procedure uses the existing mechanisms of Bluetooth for this purpose as it is, since if a scatternet satisfies the topological requirements of *Maintenance* procedure, it will do so after any node disconnection. Hence, handling node disconnections does not require a special treatment unlike the case for handling new connections.

Unlike the *Maintenance* procedure, the *Link Establishment* procedure runs on any node. Nodes, except bridges and masters that already have seven slaves, are scheduled to enter one or both of the inquiry and inquiry-scan modes, which is determined by the rules listed above, at certain intervals. These intervals are determined by the master node for itself and for its slaves. Once a new node is discovered in accordance with the restrictions on the topology of scatternet, the *Link Establishment* procedure further decides to establish the link or not to. Link establishment can be restricted in order to limit the average node degrees. One such restriction would be to prevent establishing links between a slave and the master of a one-hop distance piconet. This information can be passed to slave nodes each time they are instructed to enter inquiry-scan mode by the master, which already has one-hop distance master information in its traffic table. Other controls can be imposed whether to continue with page/page-scan mode after the discovery, using probabilistic approaches, or acquiring 2 or 3-hop information by extra messaging if desired.

Whenever the *Link Establishment* procedure handles a node connection or detects a node disconnection, it invokes the *Maintenance* procedure to change the structure of the traffic table, described in Section 3.1, appropriately. In such cases, rows and columns should be added to or deleted from the traffic table as described below. Here, M denotes a master node, S denotes a slave node, B denotes a bridge node, and F denotes a free-node:

- If a new connection is established between
  - F and M, a row and a column should be added for F, as a member of the piconet, to the traffic table at M;
  - F and S, whose master is M, a row and a column should be added for F, as a neighboring piconet, to the traffic table at M;
  - M_1 and S, whose master is M_2, rows and columns should be added for S, as a member of the piconet, and for M_2, as a neighboring piconet, to the traffic table at M_1. In addition, a row and a column should be added for M_1, as a neighboring piconet, to the traffic table at M_2.
- If a disconnection is detected between
  - M and S, a row and column for S should be deleted from the traffic table at M;
  - M_1 and B, whose other master is M_2, rows and columns for B and M_2 should be deleted from traffic table at M_1. In addition, a row and column for M_1 should be deleted from the traffic table at M_2.

The *Link Establishment* procedure enables dynamic handling of new link establishments and disconnections. Hence, a scatternet topology, which can be further modified by *Maintenance* procedure for bandwidth-efficiency, is constructed on the fly. During the execution of the *Maintenance* procedure, the set of nodes constituting the piconet should not be changed, requiring coordination between the *Link Establishment* and *Maintenance* procedures. This can be achieved if the masters do not schedule slave nodes for device discovery before starting a *Maintenance* procedure.

4 Simulation Results

In this section, we present the evaluation for our proposed algorithm. For evaluation, we use a performance metric called weighted average shortest path (WASP) proposed in [4]. WASP relates the end-to-end *traffic demand* to the *traffic load* imposed on a network with a given topology. The amount of end-to-end traffic demand between two nodes is expressed as the sum of two one-way traffic that has to flow between these two nodes. The load imposed on the network due to the traffic demand between two nodes is computed by multiplying the amount of traffic demand by the number of hops between the two nodes. Equation (8) shows how to define the WASP of the flow of a node-pair x (WASP$_x$), where d_x is the traffic demand of node pair x, L_x is the number of hops between these two nodes, and n is the number of all node-pairs demanding some amount of traffic to flow in between:

$$WASP_x = (d_x \times L_x) / \sum_{i=1}^{n} d_i. \tag{8}$$

The WASP of a network is defined as the sum of WASPs of all end-to-end flows demanded in the network:

$$WASP = \sum_{i=1}^{n} (d_i \times L_i) / \sum_{i=1}^{n} d_i. \tag{9}$$

The WASP value in a network will be equal to one, when communicating nodes in the network are separated from each other by direct links (single-hop communication). In

TABLE 2
Initial and Final WASP Values for TC-1, TC-2, and TC-3

Node Count	TC-1 Initial	TC-1 Final	TC-2 Initial	TC-2 Final	TC-3 Initial	TC-3 Final
10	2.913	1.931	2.927	1.797	2.774	1.563
20	4.202	2.999	4.126	2.866	4.161	2.264
30	5.060	4.087	4.932	3.924	4.891	2.971
40	5.670	4.917	5.686	4.672	5.656	3.813
50	6.331	5.577	6.189	5.255	6.146	4.748
60	6.688	6.041	6.626	5.752	6.633	5.372
70	7.063	6.365	6.903	6.084	7.292	6.091
80	7.467	6.827	7.332	6.602	7.402	6.396
90	7.755	7.149	7.801	6.987	7.783	6.820
100	7.995	7.417	8.029	7.329	8.245	7.348

this case, the traffic load imposed on the network by a certain traffic demand pattern will be minimum. When demand is fixed, the WASP value gets closer to one as the traffic load on the network decreases, meaning that pairs with higher traffic demands are getting closer.

In order to test the performance of the proposed algorithm, a custom simulation environment, which is implemented in C++, is developed. The simulator computes the bandwidth efficiency based on the WASP metric and energy usage of the topologies. Since the simulation environment is flow based, rather that packet based, it focuses on the bandwidth and energy efficiency of a topology and does not examine the amount of messaging required for or the delay due to the execution of the algorithm. The simulator takes the traffic characteristics as input and computes WASP values and energy usages for the initial and final topologies, for different scatternet sizes. Simulations are run for different traffic characteristics and for scatternet sizes varying between 10 and 100 nodes with a step size of 10 nodes. For each scatternet size, 100 different initial scatternet topologies are generated and the proposed algorithm is run on arbitrary master nodes until no significant improvement on total bandwidth usage is obtained. Numerically speaking, if the improvement is less than 1 percent compared to the previous topology, the scatternet is assumed to be stabilized. For each scatternet size, the WASP values for each of 100 initial and final topologies are gathered and their average is calculated. For more information on the simulation environment itself, [20] can be referenced.

Three different traffic characteristics are used in simulations. In the first type of traffic characteristic (TC-1), given a pair of nodes out of all possible pairs of nodes, they

- do not communicate with a probability of 0.3;
- communicate at
  - 5 Kbps with a probability of 0.2,
  - 10 Kbps with a probability of 0.2,
  - 15 Kbps with a probability of 0.2, and
  - 20 Kbps with a probability of 0.1.

In the second type of traffic characteristic (TC-2) given a pair of nodes, they

- do not communicate with a probability of 0.4;
- communicate at
  - 5 Kbps with a probability of 0.3 and
  - 30 Kbps with a probability of 0.3.

In the third type of traffic characteristic (TC-3), nodes are grouped into three as Group-A, Group-B, and Group-C, which constitute 30 percent, 30 percent, and 40 percent of all nodes in a scatternet, respectively. Given a pair of nodes:

- both from Group-A, they communicate at 20 Kbps with a probability of 0.7;
- both from Group-B, they communicate at 15 Kbps with a probability of 0.8;
- both from Group-C, they communicate at 15 Kbps with a probability of 0.6;
- one from Group-A and one from Group-B, they communicate at 4 Kbps with a probability of 0.2;
- one from Group-A and one from Group-C, they communicate at 2 Kbps with a probability of 0.3; and
- one from Group-B and one from Group-C, they communicate at 3 Kbps with a probability of 0.3.

In TC-1 and TC-2, although there are demands with different communication rates, the demands are evenly distributed among node pairs. Different from TC-1 and TC-2, in TC-3, however, there are groups where we have higher rate of traffic demands between the nodes belonging to the same group and lower rate demands between the nodes belonging to different groups. The initial and final WASP values for these three different traffic characteristics on varying scatternet sizes are listed in Table 2. As it can be clearly seen, whatever the traffic characteristics are, the initial WASP values are very similar. However, as the scatternet-wide communication relations among nodes get weaker, the final WASP values decrease.

In Fig. 6, improvements in WASP for TC-1, TC-2, and TC-3 are shown for varying scatternet sizes. For TC-3, the improvements are higher than TC-1 and TC-2, and

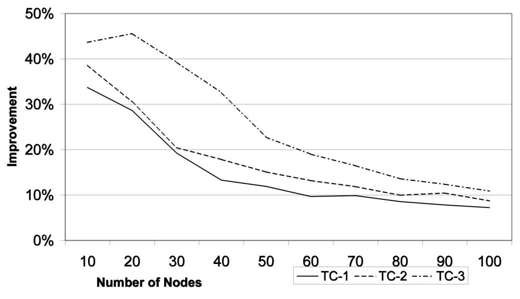


Fig. 6. Improvement in WASP.

improvements up to 46 percent are reached, since heavily communicating nodes can be grouped together in the scatternet for TC-3. On the other hand, in TC-1 and TC-2, as a node is moved toward another node it highly communicates with, it gets further from the other nodes it communicates with at the same rate, because every node has a similar probability of communication with another node at certain rate. TC-2 has better improvement than TC-1, since the ratio of noncommunicating nodes is higher in TC-2 and it has a more diverse traffic distribution than TC-1. Fig. 6 also shows that, as the scatternets get larger, the improvements decrease. This is due to the increasing communication rate per piconet. As the number of communicating nodes increases, the total traffic demand and, consequently, the load on the scatternet increases, meaning that a piconet should relay more traffic. According to the proposed algorithm, a piconet is divided if the traffic flow rate within the piconet exceeds 1 Mbps. Hence, as the scatternet gets larger, the piconets tend to divide, which increases the path lengths between communicating nodes and consequently the value of WASP metric. Although improvement in the WASP decreases, since the average available bandwidth per piconet on the way from source to destination increases, the data packets are not dropped due to high congestion, which, in fact, means improvement in general performance.

The simulation environment is also used to investigate the differences between the initial and final scatternet topologies. For example, the scatternet, which has a chain topology, shown in Fig. 7, is converted into the topology shown in Fig. 8a in four executions of the algorithm. In the figure, the dark nodes are masters, gray nodes are bridges, and white nodes are slaves. For this case, the traffic demand for each node pair is assumed to be 3 Kbps. The WASP value of the initial topology is 3.3 and the WASP value of the final topology is 2. As it is clear from the figure, after the algorithm is applied, we have fewer of piconets and, therefore, they are utilized better. Starting from the same chain topology, but this time with traffic demands among the nodes increased to 8 Kbps per node pair, the final topology depicted in Fig. 8b is reached. This final topology

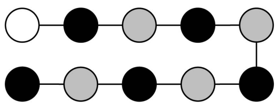


Fig. 7. Initial scatternet topology.

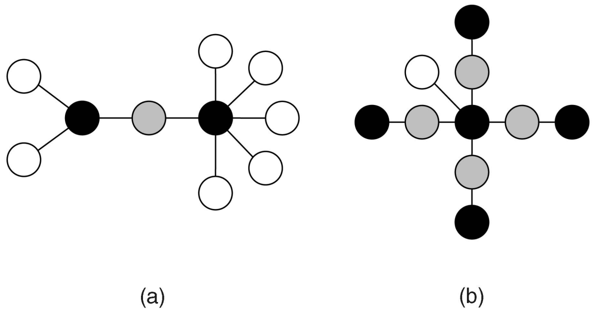


Fig. 8. Final scatternet topologies at communication rates of (a) 3 Kbps and (b) 8 Kbps.

is different from the previous final topology, due to the fact that the capacity of a piconet is limited and, therefore, its value affects the topology of the resulting scatternet. The final WASP value is 2.18 this time.

Our algorithm shortens the path lengths between nodes. Hence, it reduces the amount of data needed to be forwarded. This, in turn, reduces the energy consumption. In order to measure the effectiveness of our algorithm in this respect as well, we observed various metrics related to energy consumption with our simulation experiments. To compute the energy required for transporting traffic between two adjacent nodes, we used the radio model proposed in [21]. One parameter of this model is the distance (d) between two adjacent nodes that are communicating. In the model, this distance affects the energy required to transmit. We assumed that the Bluetooth nodes do not perform power control, hence, the energy required to transmit is independent of the distance and we take it fixed. The fixed energy consumption value per unit transfer is computed with respect to the maximum communication range of Bluetooth, which is 10 meters for most Bluetooth devices. We fixed the values of the remaining parameters of the model as they are in [21], since these values are consistent with the values specified in Bluetooth standards [1].

As in the case with the WASP measurements, we run simulations for different scatternet sizes. For each scatternet size, we repeated the experiments 100 times, each time with a different random topology. Energy consumptions of the nodes are measured for each of these topologies and their average is calculated. In Fig. 9, energy consumption distribution is plotted, which shows how many nodes

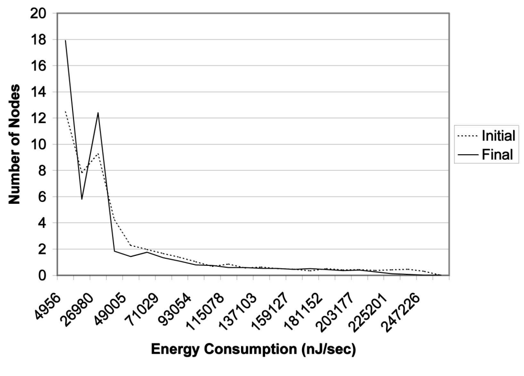
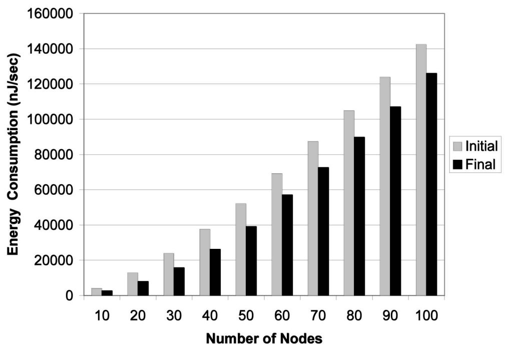
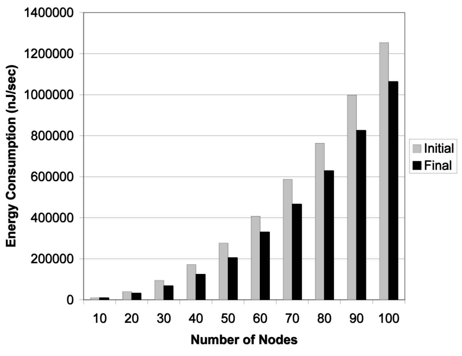


Fig. 9. Energy consumption distribution.



(a)



(b)

Fig. 10. Average and maximum energy consumption.

consume how much power (nJ/sec) before and after the execution of our algorithm. This is the simulation result we obtained for a scatternet with 50 nodes, but we also obtained very similar results for other scatternet sizes between 10 and 100. In Fig. 10a, the average energy consumption value per node is plotted for different scatternet sizes. We observe that up to 38 percent reduction in the average energy consumption per node is achieved after the execution of our algorithm. This is also the case for total energy consumption in a scatternet. Energy consumption of the node that consumes the maximum energy in a scatternet is plotted in Fig. 10b for different scatternet sizes. As it can be seen from the figure, our algorithm achieves up to 28 percent reduction in the maximum energy consumption at a node. In Fig. 11, we also plot the standard deviation of the energy consumption per node in a scatternet. We see a decrease in the standard deviation after execution of our algorithm. This show that our algorithm not only reduces the overall energy consumption in a scatternet, but also it leads to a more even distribution of power consumption among the nodes. Reduction in standard deviation is up to 24 percent.

5 Conclusion and Future Work

In this paper, we propose a Bluetooth scatternet formation and maintenance algorithm. The algorithm is different from previous proposals due to its basic motivation, which is to minimize the bandwidth usage in the resulting scatternet topology. The proposed algorithm works in a distributed fashion and it is adaptive, that is, it preserves bandwidth efficiency as traffic demands change. It is also dynamic so that it handles arrivals of new nodes and departures of existing ones. As the bandwidth usage is reduced in the scatternet, available bandwidth for new communication demands increases. Furthermore, since the average load on the links is reduced, the average end-to-end latency is also reduced. Another benefit is that the average power consumption per node is expected to decrease, since the amount of traffic which is not directly related with a node but still needs to be forwarded by that node is reduced. This expectation is also verified by simulation results, which showed that our algorithm not only reduces overall and average power consumption, but also leads to a more even distribution of power consumption. We evaluated the proposed algorithm with respect to different aspects and the results are observed to be promising.

Although the proposed scatternet formation and maintenance algorithm is designed to meet the requirements of a practical solution for wireless ad hoc networks, such as distributed approach with as low messaging overhead as possible and adaptivity to changing conditions, it has one major restriction which is the assumption that all nodes are in the Bluetooth range of each other. This assumption may not be valid for some application scenarios. Therefore, for future work, we consider enhancing the proposed algorithm to handle this kind of situation as well. Another open issue about the algorithm is that it lacks the proof of guaranteeing to reach a stable topology if the nodes that constitute the scatternet and their communication demands remain unchanged. Due to the dynamic nature of ad hoc networks, both in the sense of node movements and traffic variations, this is not a very significant problem for practical purposes, but it is an important issue and, therefore, should be examined as part of our future work.

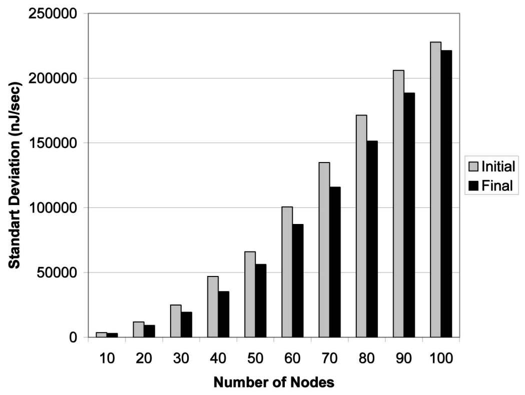


Fig. 11. Standard deviation of energy consumption.

Acknowledgments

This work is supported partially by The Scientific and Technological Research Council of Turkey (TÜBITAK) with projects 103E014 and 104E028, and by the Europion Union FP-6 Program with project NEWCOM IST NoE 507325. The authors also thank Dilek Karabudak for kindly reviewing this paper.

REFERENCES

[1] Bluetooth, Bluetooth Special Interest Group, 2006, http://www.bluetooth.com.

[2] S. Baatz, C. Bieschke, M. Frank, C. Kuhl, P. Martini, and C. Scholz, "Building Efficient Bluetooth Scatternet Topologies from 1-Factors," *Proc. IASTED Int'l Conf. Wireless and Optical Comm.*, 2002.

[3] M.A. Marsan, C.F. Chiasserini, A. Nucci, G. Carello, and L.D. Giovanni, "Optimizing the Topology of Bluetooth Wireless Personal Area Networks," *Proc. IEEE INFOCOM,* 2002.

[4] T. Topal, "Constructing Efficient Bluetooth Scatternets," Master's thesis, Dept. of Computer Eng., Bilkent Univ., http://www.thesis.bilkent.edu.tr/0002478.pdf, 2004.

[5] R.M. Whitaker, L. Hodge, and I. Chlamtac, "Bluetooth Scatternet Formation: A Survey," *Ad Hoc Networks,* to appear.

[6] C. Law, A.K. Mehta, and K.Y. Siu, "Performance of a New Bluetooth Scatternet Formation Protocol," *Proc. ACM Symp. Mobile Ad Hoc Networking and Computing (MobiHoc),* 2001.

[7] C. Law and K.Y. Siu, "A Bluetooth Scatternet Formation Algorithm," *Proc. Symp. Ad Hoc Wireless Networks,* 2001.

[8] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire, "Distributed Topology Construction of Bluetooth Personal Area Networks," *Proc. INFOCOM,* 2001.

[9] G. Tan, A. Miu, J. Guttag, H. Balakrishnan, T. Berners-Lee, L. Masinter, and M. McCahill, "Forming Scatternets from Bluetooth Personal Area Networks," Technical Report Mit-lcs-tr-826, MIT Laboratory for Computer Science, 2001.

[10] G. Zaruba, S. Basagni, and I. Chlamtac, "Bluetrees-Scatternet Formation to Enable Bluetooth-Based Personal Area Networks," *Proc. IEEE Int'l Conf. Comm.,* 2001.

[11] C. Petrioli, S. Basagni, and I. Chlamtac, "Configuring Bluestars: Multihop Scatternet Formation for Bluetooth Networks," *IEEE Trans. Computers,* special issue on wireless Internet, vol. 52, no. 6, pp. 779-790, 2003.

[12] D. Miorandi and A. Zanella, "On the Optimal Topology of Bluetooth Piconets: Roles Swapping Algorithms," *Proc. Mediterranean Conf. Ad Hoc Networks, Med-Hoc-Net,* 2002.

[13] F. Cuomo, G. di Bacco, and T. Melodia, "Shaper: A Self-Healing Algorithm Producing Multi-Hop Bluetooth Scatternets," *Proc. IEEE Globecom,* Dec. 2003.

[14] C.-F. Chiasserini, M.A. Marsan, E. Baralis, and P. Garza, "Towards Feasible Distributed Topology Formation Algorithms for Bluetooth-Based WPANs," *Proc. 36th Hawaii Int'l Conf. System Science (HICSS-36),* Jan. 2003.

[15] G. Miklos, A. Racz, Z. Turanyi, A. Valko, and P. Johansson, "Performance Aspects of Bluetooth Scatternet Formation," *Proc. First ACM Int'l Symp. Mobile Ad Hoc Networking and Computing,* 2000.

[16] R. Kapoor, M. Sanadidi, and M. Gerla, "An Analysis of Bluetooth Scatternet Topologies," *Proc. Int'l Conf. Comm. (ICC),* 2003.

[17] D. Miorandi, A. Trainito, and A. Zanella, "On Efficient Topologies for Bluetooth Scatternets," *Lecture Notes in Computer Science,* vol. 2775/2003, pp. 726-740, 2003.

[18] M. Kalia, S. Garg, and R. Shorey, "Scatternet Structure and Inter-Piconet Communication in the Bluetooth System," *Proc. IEEE Nat'l Conf. Comm.,* 2000.

[19] K.E. Persson, D. Manivannan, and M. Singhal, "Bluetooth Scatternets: Criteria, Models and Classification," *Ad Hoc Networks,* to appear.

[20] M. Tekkalmaz, "Distributed Construction and Maintenance of Bandwidth-Efficient Bluetooth Scatternets," Master's thesis, Dept. of Computer Eng., Bilkent Univ., http://www.thesis.bilkent.edu.tr/0002644.pdf, 2004.

[21] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," *Proc. Hawaii Int'l Conf. System Sciences,* 2000.

Metin Tekkalmaz received the BS and MS degrees in computer engineering from Bilkent University, Ankara, Turkey, in 2002 and 2004, respectively. Currently, he is a software engineer at ASELSAN Inc. and a PhD student at Bilkent University. His research interests include wireless ad hoc and sensor networks.

Hasan Sözer received the BS and MS degrees in computer engineering from Bilkent University, Ankara, Turkey, in 2002 and 2004, respectively. From August 2002 until January 2004, he worked as a software engineer at ASELSAN Inc. in Turkey. He is currently a PhD student at the University of Twente in The Netherlands. His research interests include software engineering and wireless ad hoc networks.

Ibrahim Korpeoglu received the BS degree in computer engineering from Bilkent University, Ankara, Turkey, and the MS and PhD degrees in computer science from the University of Maryland, College Park. He is currently an assistant professor in the Department of Computer Engineering at Bilkent University. Prior to that, he worked in several companies in the US, including Ericsson, the IBM T.J. Watson Research Center, Bell Laboratories, and Telcordia Technologies. His research interests include wireless ad hoc and sensor networks, mobile computing, peer-to-peer networks, and distributed systems. He is a member of the IEEE and the ACM.