Hop Count Based Optimization of Bluetooth Scatternets

Csaba Kiss Kalló
Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy
Email: kkcsaba@dit.unitn.it
Carla-Fabiana Chiasserini
Dipartimento di Elettronica, Politecnico di Torino, Italy
Email: chiasserini@polito.it
Sewook Jung
Department of Computer Science, University of California, Los Angeles
Email: sewookj@cs.ucla.edu
Mauro Brunato
Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy
Email: brunato@dit.unitn.it
Mario Gerla
Department of Computer Science, University of California, Los Angeles
Email: gerla@cs.ucla.edu

Abstract

In the past five years Bluetooth scatternets were one of the most promising wireless networking technologies for ad hoc networking. In such networks, mobility together with the fact that wireless network nodes may change their communication peers in time, generate permanently changing traffic flows. Thus, forming an optimal scatternet for a given traffic pattern may be not enough, rather a scatternet that best supports traffic flows as they vary in time is required.

In this paper we study the optimization of scatternets through the reduction of communication path lengths. After demonstrating analytically that there is a strong relationship between the communication path length on one hand and throughput and power consumption on the other hand, we propose a novel heuristic algorithm suite capable of dynamically adapting the network topology to the existing traffic connections between the scatternet nodes. The periodic adaptation of the scatternet topology to the traffic connections enables the routing algorithms to identify shorter paths between communicating network nodes, thus allowing for more efficient communications. We evaluate our approach through simulations, in the presence of dynamic traffic flows and mobility.

I. Introduction

Bluetooth is a short-range wireless network technology that supports ad hoc networking. In the Bluetooth technology a maximum of 8 nodes, out of a total of 256 devices, can actively communicate in a star-shaped cluster, called *piconet*. Within a piconet, the cluster head is called *master* while the other nodes are called *slaves*. Piconets interconnected through so-called *bridge* nodes form a *scatternet*. Bridges are nodes participating in more than one piconet on a time sharing basis. We call *slave&bridges* those nodes that have slave role in all of the piconets they participate in, while nodes having both, slave and master roles in different piconets are *master&bridges*.

The latest Bluetooth Specification (v.1.2 [3]) introduces the concept of scatternet formation, but it does not define it in detail. Several scatternet formation algorithms however have been proposed in the literature. The work in [8] suggests the usage of the low power modes specified by the Bluetooth technology to enable up to 256 nodes to be part of the same piconet. Since only 8 nodes can actively communicate in a piconet, this approach implies that the master would continuously have to put

and call back the nodes into/from low power modes, thus leading to a significant waste of radio resources. In [20], [18], [13], as well as in [8], algorithms for creating tree-shaped scatternets are proposed. Although tree network structures greatly simplify traffic routing, they also have important drawbacks: network partitions may arise as nodes move, source-destination paths may be quite long, the root node may become a bottleneck. These facts suggest that tree-like topologies can operate efficiently under specific scenarios but cannot be considered for general-purpose scatternets.

Mesh-shaped scatternets do not have the limitations of the tree-shaped networks, but they require a more complex routing scheme. However, simulation results show that in general scenarios mesh-shaped scatternets perform better than their tree-shaped counterparts [8]. In the mesh-shaped arena, some early protocol proposals [16], [11] required the nodes to be all in radio range, which simplifies node discovery and piconet formation. Later some solutions [19], [17], [2], [14] were defined to avoid the above shortcomings and operate well under general scenarios, providing high throughput and a balanced number of nodes per piconet. Despite these good characteristics, we can still identify some weaknesses. In particular, in [2] more than 7 slaves can be included in a piconet. The problem is solved in [17] and [14], however the protocol in [17] requires the nodes to know their geographic position.

Despite the wide range of solutions proposed for optimal toplogy formation, the performance of scatternets over time is still challenged by the dynamic behavior of the nodes, since the varying communication needs of the users, node mobility, and channel conditions reshape the network topology in an unpredictable way. As an example, think of scatternets operating in interfering industrial environments with machinery that autonomously or semi-autonomously accomplishes its tasks. Components of such an automated environment include static as well as mobile robots, sensors of various type and human supervisors. All these components need to be networked for exchanging the data necessary for accomplishing their tasks. Raw data used for the tasks, progress reports and control data are all examples for information that need to be exchanged among the components. Also, each node may have multiple communication peers sustaining random data traffic sessions with them, sequentially and/or in parallel. To achieve high performance, a scatternet topology should be continuously maintained so that the current traffic flows can be supported in an optimal way.

One of the factors that has a major impact on scatternet performance is the length of the communications paths. Intuitively, if a packet has to pass through many hops from its source to the destination, it occupies the communication bandwidth on more links and make the nodes consume more energy than in the case of shorter paths. To demonstrate the correctness of this intuition, in this paper first we provide an analytical model for estimating the throughput and power consumption on the communication paths of a scatternet, based on [9]. Then we devise an algorithm suite that enables the reduction of the path length (or hop count) on all traffic connections in the scatternet. These algorithms take advantage of a local search strategy to find a network topology that can support the current traffic connections with a lower number of hops between the communicating nodes. Finally, the original contribution of this paper lies in the evaluation of the impact of the hop reductions on the throughput and power consumption through simulations, in the presence of node mobility and varying traffic connections.

The remainder of this paper is organized as follows. In Section II we present our scatternet model and use it for providing a formal definition of our scatternet optimization problem. In Section III we calculate analytically the usable bandwidth on the links of the scatternet and use the results in Section IV to determine an analytical relationship between the hop count on the one hand, and throughput and power consumption on the other hand. In Section V we present and evaluate our algorithm suite that we use to reduce the hop count on communication paths, while in Section VI we evaluate the impact of these algorithms on the scatternet performance in the presence of mobility and changing traffic flows between the nodes.

II. Scatternet Modeling

To provide a formal definition for our optimization objectives, we devise the following scatternet model.

Let $\mathcal{N}$ be the *set of nodes* in the scatternet, $\mathcal{M}$ the *set of masters*, and $\mathcal{S}$ the *set of all slaves*. Notice that only pure masters are not elements of $\mathcal{S}$ and $\mathcal{S} \bigcap \mathcal{M} \neq \emptyset$ if there are master&bridge nodes in the scatternet. We denote with $\mathcal{C}$ the *set of traffic connections* in the scatternet.

$\mathcal{R} = \{r_{ij}^{sd}\}$, the *routing matrix*, stores the path between each source-destination pair $(s,d) \in \mathcal{C}$; we have

$$r_{ij}^{sd} = \begin{cases} 1 & \text{if connection } (s,d) \text{ is routed on arc } (i,j), \\ 0 & \text{otherwise.} \end{cases}$$

$\mathcal{T} = \{t_{sd}\}$ is the *traffic matrix* with $t_{sd} \in [0,1]$ indicating the intensity of the data flow on the connection (s,d). $t_{sd} = 0$ means that there is no traffic flow between the nodes s and d.

$\mathcal{H} = \{h_{sd}\}$, the *hop matrix*, contains the minimum number of hops between any connection $(s,d) \in \mathcal{C}$.

$\mathcal{P} = \{p_{ij}\}$ is the *radio proximity matrix* with

$$p_{ij} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ are in-range,} \\ 0 & \text{otherwise.} \end{cases}$$

The *link matrix* $\mathcal{L} = \{l_{ij}\}$ is defined as

$$l_{ij} = \begin{cases} 1 & \text{if } i \text{ is master of } j,\ \forall i,j \in \mathcal{N}; i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

The link matrix indicates the master-slave connections in the scatternet. Link matrix properties are explained below.

1) A *master* has on its row one entry equal to 1 for each of its slaves.

2) A *pure slave* has one entry equal to 1 on its column corresponding to its master.

3) A *slave&bridge* has on its column exactly one entry equal to 1 for each of its masters.

4) A *master&bridge* node has one entry equal to 1 for each of its slaves on its row and for each of its masters on its column.

5) A *free node* – node not belonging to any piconet – has all 0s on both its row and column.

We define function F as

$$F = \sum_{(s,d) \in \mathcal{C}} t_{sd} h_{sd}. \tag{1}$$

F is the sum of the number of hops weighted with the traffic intensity between all source-destination pairs in the scatternet.

Our objective is to solve the following optimization problem,

$$\mathcal{P}: \qquad \min_{H} F \tag{2}$$

subject to the following constraints [1].

- A piconet must contain one master and up to 7 slaves,

$$\sum_{j=1}^{N} l_{kj} \leq 7, \forall\, k \in \mathcal{M} \tag{3}$$

- There can exist a master-slave relationship between two nodes if and only if they are in radio proximity of each other,

$$l_{ij} \leq p_{ij}, \forall\, i,j \in \mathcal{N} \tag{4}$$

- If i is master of j, then j cannot be master of i,

$$l_{ij} + l_{ji} \leq 1, \forall\, i, j \in \mathcal{N}; i \neq j \tag{5}$$

- Traffic between source s and destination d can be routed through edge (i, j) only if i and j communicate, i.e., either i is assigned to j, or j is assigned to i,

$$r_{ij}^{sd} \leq l_{ij} + l_{ji} \quad \forall\, (s, d) \in \mathcal{C}, \forall\, i, j \in \mathcal{N} \tag{6}$$

- All traffic between two nodes is routed through a minimum length paths, with no loops. The selected path may not necessarily be the same in both directions, if more than one minimum length paths exist,

$$h_{sd} = h_{ds} \quad \forall\, (s, d) \in \mathcal{C} \tag{7}$$

$$\sum_{i,j \in \mathcal{N}} r_{ij}^{sd} = h_{sd} \quad \forall\, (s, d) \in \mathcal{C} \tag{8}$$

- Standard constraints used for routing should also be considered.

III. Communication Capacity

To determine an analytic relationship between hop count, throughput and power consumption in a scatternet we need to take into account two fundamental issues: the *Bluetooth packet types* and *link scheduling*.

Bluetooth data communication takes place through Asynchronous Connectionless Links (ACL) using time slots of 625 μs. Data packets may use 1, 3 or 5 slots and they may be Forward Error Coded (FEC). FEC packets are called DM1, DM3 and DM5, enabling the correction of single-bit errors in each codeword of 15 bits, while the non-error coded ones are named DH1, DH3 and DH5 (with the digits indicating the number of slots used). The useful maximum payload of these packets is 136, 968 and 1816 bits for DM packets and 216, 1464 and 2712 bits for DH packets, respectively. Packets with larger payloads can achieve higher throughput in error-free environments (i.e., with high link quality); however, if a bit gets corrupted, the whole packet has to be retransmitted. DM packets have smaller payloads than DH packets, but their content is error checked, in contrast with DH packets.

Link scheduling refers to the allocation of time slots to a bridge node. A bridge node can be present in one piconet at a time, thus it has to switch continuously between its piconets for being reachable by each of its masters and to relay traffic efficiently. Since we aim at analyzing the scatternet throughput, link scheduling is of fundamental importance. Next we present the analytical model of the link scheduling algorithm that we use in our work.

In our approach each piconet is assigned an overall traffic capacity of 1 (hence, the traffic rate of a pure master is equal to 1, too). This capacity is divided among all slaves in the piconet according to the expressions (9)–(18) that account for the case where we have a pure master and a master&bridge, respectively. For a pure master, we can simply write:

$$p(pm) = 1, \tag{9}$$

where $p(pm)$ denotes the communication capacity allocated to the piconet of pure master pm.

Since master&bridge nodes have to switch between different piconets, we assume that each piconet switching takes two slots, 625 μs long each. We denote the communication capacity of a node wasted for one piconet switching by σ. On average, a node spends in each of its piconets about 40ms, as proposed in [15]. The capacity dedicated to the piconets of a master&bridge node mb is given by,

$$p(mb) = \frac{1}{NrM(mb) + 1} - \sigma, \tag{10}$$

where $p(mb)$ is the communication capacity allocated to the masters of a master&bridge mb as well as its own piconet; $NrM(mb)+1$ is the total number of mb's piconets. Note that the fact that mb is a master&bridge implies that (10) holds only when $NrM(mb) \geq 1$.

To share the communication capacity among the nodes within a piconet, a simple solution is to allocate the same amount of bandwidth for each master-slave link. The problem with this approach is that it allocates the same amount of bandwidth to all nodes, including bridge nodes that can dedicate less of their communication capacity to a particular master since they have to take part in multiple piconets.

To fix the above problem, we define α, the *availability factor of a node with respect to a piconet* (hereinafter simply availability factor), as the ratio of the piconet allocated bandwidth for the node to the node available communication capacity for that piconet. Taking advantage of the availability factor, we observe the following properties. A node is said to be *underloaded* with respect to a particular piconet if it can dedicate more bandwidth to that piconet than the amount of bandwidth that the piconet can allocate to the node, i.e., $\alpha < 1$. Clearly, if $\alpha \geq 1$ then the corresponding node is *overloaded*. Whether a node playing a certain role in the scatternet is underloaded or overloaded can be evaluated as follows.

- *Pure slave (ps)*: a *ps* is connected to its master only, hence $\alpha < 1$ except for a piconet made of two nodes (which is an insignificant case from the scatternet point of view).
- *Pure master (pm)*: since the *pm* dedicates all of its bandwidth to its piconet, we always have $\alpha = 1$.
- *Slave&bridge (sb)*: initially the available communication capacity of a *sb* is uniformly shared among its masters. However, its masters can allocate to the *sb* an arbitrary amount of bandwidth in each of its piconets. In this case α may be either smaller or greater than 1 and, hence, slave&bridges may be either underloaded or overloaded.
- *Master&bridge (mb)*: a *mb* manages the whole bandwidth available for its piconet. Therefore, the availability factor of a *mb* with respect to its own piconet is $\alpha = 1$.
  Note that the availability factor of the *mb* with respect to the piconets of its masters can be calculated similarly to the *sb* case. Thus, in the latter case *mb* nodes may be either underloaded or overloaded.

Based on the above considerations, we can write the number of underloaded nodes in the piconet of the generic master m ($NrUN(m)$) as the sum of the pure slaves ($NrPS(m)$) and the underloaded bridges ($NrUB(m)$),

$$NrUN(m) = NrPS(m) + NrUB(m) \tag{11}$$

Then we can calculate the number of overloaded slaves ($NrOS(m)$) as,

$$NrOS(m) = NrS(m) - NrUN(m) \tag{12}$$

where $NrS(m)$ is the number of slaves of m. Given the number of underloaded and overloaded nodes in a piconet, we can define the link capacities (c) in each piconet as follows. For overloaded links ($\alpha \geq 1$) from masters to slave&bridges we have:

$$c^o_{sb} = \frac{1}{NrM(sb)} - \sigma \tag{13}$$

For overloaded links ($\alpha \geq 1$) from masters to master&bridges we have the same expression as in (10), since master&bridges allocate the same communication capacity for both their masters and piconet:

$$c^o_{mb} = p(mb) = \frac{1}{NrM(mb)+1} - \sigma \tag{14}$$

The capacity of a pure master or master&bridge m that is not used by overloaded links is uniformly redistributed among the underloaded links in its piconet, similarly to the max-min fair technique [12], [7]. For each of such masters m, the obtained capacity fraction after the redistribution is stored in the vector $\rho^m = \{\rho_i^m | i = \overline{0, NrUN(m)}\}$. The fraction of the unallocated capacity that is still not used by the links after the redistribution is stored in ρ_0^m. Note that if the unallocated capacity can be fully redistributed among the links, then $\rho_0^m = 0$. Equation (15) captures the redistributed capacity of an underloaded link, connecting any type of master m to any type of slave s,

$$c_s^u(m) = \left(p(m) - \sum_{i=1}^{NrOS(m)} c_i^o \right) \cdot \rho_s^m \tag{15}$$

where $\sum_{i=1}^{NrOS(m)} c_i^o$ gives the aggregate communication capacity allocated for all overloaded slaves of master m. Notice that $p(m)$ should be expressed as in (9) or (10) for pure masters or master&bridges, respectively. In (15) we subtract from the total communication capacity of the piconet the bandwidth allocated for the overloaded nodes (obtaining the total unallocated capacity of m), then we multiply it by the capacity fraction corresponding to the underloaded link connecting master m to its slave s.

Before terminating the capacity allocation, each node compares its own communication capacity of 1 to the total amount of bandwidth received from other nodes. If the received aggregate bandwidth is smaller than 1, then the node has some residual unallocated capacity. Each node having unallocated capacity tries to allocate it to its neighbors. For each node n, the capacity fraction obtained after this reallocation is stored in the vector $\delta^n = \{\delta_i^n | i = \overline{0, NrN(n)}\}$, where $NrN(n)$ is the number of neighbors of node n with unallocated capacities. After several iterations of this latter phase, each node n will have allocated as much as possible of its residual capacity of δ^n. The updated formulae for (13)–(15) are reported below.

$$c_{sb}^o = \tfrac{1}{NrM(sb)} - \sigma + \delta_{sb}^n \tag{16}$$

$$c_{mb}^o = p(mb) = \tfrac{1}{NrM(mb)+1} - \sigma + \delta_{mb}^n \tag{17}$$

$$c_s^u(m) = (p(m) - \textstyle\sum_{i=1}^{NrOS(m)} c_i^o) \cdot \rho_s^m + \delta_s^n \tag{18}$$

IV. Throughput and Power Estimation

Taking advantage of the packet types and link scheduling model presented in Section III, here we present how the overall scatternet throughput and power consumption can be calculated, also showing the dependence of these performance metrics on the length of the communication paths.

Based on the notation introduced in Section II and the results presented in Section III, we can calculate the maximum usable bandwidth, f_{ij}, of a radio link l_{ij}, as follows:

$$f_{ij} = \begin{cases} c_{sb}^o, & \alpha_{ij} \geq 1,\ i \text{ is master},\ j \text{ is slave\&bridge}, \\ c_{mb}^o, & \alpha_{ij} \geq 1,\ i \text{ is master},\ j \text{ is master\&bridge}, \\ c_p^u(m), & \alpha_{ij} < 1,\ i = m \text{ is master},\ j \text{ is any slave} \end{cases}$$

where α_{ij} is the availability factor of node j with respect to the piconet of master i.

The maximum usable bandwidth of a link (f_{ij}) is shared by the traffic connections crossing that specific link as shown in (19). In (19) we denote by $(s,d) \supset (i,j)$ all connections (s,d) crossing link (i,j) and by b_{ij}^{sd} the bandwidth portion allocated to the particular connection (s,d) on link l_{ij}. To compute the b_{ij}^{sd} values, we use the max-min fair bandwidth allocation algorithm.

$$f_{ij} = \sum_{(s,d) \supset (i,j)} b_{ij}^{sd} \tag{19}$$

Let us denote by $B_{ij} = \{b_{ij}^{sd} | (s,d) \in \mathcal{C}\}$ the vector of bandwidth portions allocated to each connection (s,d) on link l_{ij}. We can write the throughput of a traffic connection, $(s,d) \in \mathcal{C}$, as

$$\theta^{sd} = C \cdot \min_{(i,j)\in(s,d)} (b_{ij}^{sd} \cdot q_{ij}) \tag{20}$$

where C is the maximum capacity of a Bluetooth radio link, specific for each DH and DM packet type, $min_{(i,j)\in(s,d)}(b_{ij}^{sd} \cdot q_{ij})$ denotes the smallest bandwidth portion on the links used by a connection (s,d) (i.e., the bottleneck), while q_{ij} is the packet success rate (PSR) of link l_{ij}. PSR can be obtained from the packet error rate (PER), as in (21), while PER can be calculated as a function of the bit error rate (BER), using (22) for the DH and DM packet types [4]. In these formulae we denoted by s the size of packets in bits.

$$q = 1 - PER \tag{21}$$

$$PER = \begin{cases} 1-(1-BER)^s & \text{(DH packets)} \\ 1-((1-BER)^{15} + 15BER(1-BER)^{14})^{s/15} & \text{(DM packets)} \end{cases} \tag{22}$$

The BER can be obtained from the link quality (LQ) value with some vendor-specific formula. However, [3] states that LQ values should be normalized to the range $[0, 255]$ and defines the *Get_Link_Quality* system function call that can be used to obtain these values. In our calculus we use the following CSR (Cambridge Silicon Radio) model:

$$\begin{aligned} BER &= (255 - LQ)/40000, && 215 \leq LQ \leq 255 \\ BER &= 32 \cdot (255 - LQ)/40000, && 105 < LQ \leq 215 \\ BER &= 256 \cdot (255 - LQ)/40000, && 0 \leq LQ \leq 105 \end{aligned}$$

Finally, the aggregate throughput over all traffic connections (i.e., scatternet throughput) can be calculated as:

$$\theta_a = \sum_{(s,d)\in\mathcal{C}} \theta^{sd} = C \cdot \sum_{(s,d)\in\mathcal{C}} \min_{(i,j)\in(s,d)} (b_{ij}^{sd} \cdot q_{ij}) \tag{23}$$

Having obtained the expression of the scatternet throughput, we now demonstrate the relation between θ_a and the hop count (h). Notice that h can be calculated as the sum of the elements of the bandwidth portion vector on all links:

$$h = \sum_{(i,j)\in\mathcal{E}} |B_{ij}| \tag{24}$$

, where $\mathcal{E}$ is the set of all radio links set up in the network. According to (24), each unitary hop count reduction implies that one bandwidth portion of the involved link is released. If the link capacity was not fully utilized before the hop reduction, then the network throughput remains unchanged (however, the power consumption decreases, as we will see later in this section). Instead, if the link capacity was fully utilized, then after the hop reduction the bandwidth used by the old connection is distributed among the remaining ones. In other words, the bandwidth portions b_{ij}^{sd} increase on the involved link. This implies that all connections having their bottleneck on the considered link are allocated new bandwidth, i.e., the minimum b_{ij}^{sd} value grows. From (23), it can be seen that this growth has direct positive impact on the aggregate throughput θ_a, and clearly shows why lower scatternet hop counts can produce higher network throughput.

The second metric of interest for our analysis is the power consumption. We denote with P_t and P_r the power consumption, respectively, for transmitting and receiving data at the full capacity of a radio

link. Data is transmitted and received by all nodes along a path, i.e., all data bits are transmitted and received as many times as the number of hops along the path. Thus, the power consumption of a traffic connection, $(s, d) \in \mathcal{C}$, can be expressed as

$$P^{sd} = (P_t + P_r) \cdot h_{sd} \cdot \min_{(i,j)\in(s,d)} (b_{ij}^{sd}) \tag{25}$$

Notice that the factor $min_{(i,j)\in(s,d)}(b_{ij}^{sd})$ in (25) adapts the power consumption to the bandwidth of the bottleneck link along the path.

The aggregate power consumption through all connections, P_a, is given by:

$$P_a = \sum_{(s,d)\in\mathcal{C}} P^{sd} = (P_t + P_r) \cdot \sum_{(s,d)\in\mathcal{C}} h_{sd} \cdot \min_{(i,j)\in(s,d)} (b_{ij}^{sd}) \tag{26}$$

The dependence of the power consumption on the hop count is evident since h_{sd} appears explicitly in (25) and (26).

A. Results

For the throughput and power consumption evaluation, we implemented our model in C++. We performed experiments with 50 scatternets over which we averaged the obtained results. Each scatternet includes 100 randomly positioned nodes, with communication range of 10 m. The nodes are scattered on a 66x66 m^2 area so that a connected scatternet can be formed with high probability. For every scatternet, we generate 15 to 50 random bidirectional traffic connections. The number and length of the connections are fixed for each particular experiment. We perform experiments varying the length of connections from 1 to 10 hops as well as modifying the link quality value in the range of $[215, 255]$. The lower bound of 215 corresponds to the maximum bit error rate of 0.1% allowed by the Bluetooth Specification at the distance of 10 m with no obstacles. Finally, we set $P_r = 150$ mW and $P_t = 170$ mW, which represent the average values of power consumption computed considering different Bluetooth chips, such as CSR and Oki.

Figure 1.a) presents the average throughput on 15, 25 and 50 bidirectional traffic connections. In this figure we show one of the main objectives of our work, i.e., the throughput decreases with the increasing number of hops. The results show the maximum achievable average throughputs, since we use the two largest packet types, (i.e., DH5 and DM5) and the link quality is set to 255 (i.e., no packet loss). Clearly, the highest average throughput per connection is achieved with 15 connections and using DH5 packets, since in this case more bandwidth can be allocated to each connection.

Figure 1.b shows the dependence of the throughput on the link quality. In this experiment the number of bidirectional connections is fixed to 50 and we use DH5 packets only. However, the connection length is different on each curve. As expected, the throughput increases with the link quality and shorter connections are less affected by the link quality, while the longer ones have a very low throughput.

Figure 2 presents the average power consumption on 15, 25 and 50 bidirectional connections. The packet type in this case has no importance since power is consumed at the same extent by both useful payload bits and error coding bits. We can observe that in the plot initially the power consumption decreases, then it starts increasing again. This is because the shorter the connections the higher the throughput, and the higher the power consumption. In other words, power consumption is high because more traffic is transmitted and not because it is less efficiently used. However, as the number of hops further increases and the throughput goes down, the real trend of power consumption shows up. It can also be seen that the highest amount of power is consumed when we have 15 connections, since in this case the throughput is higher (this, on turn, makes the power consumption increase faster).

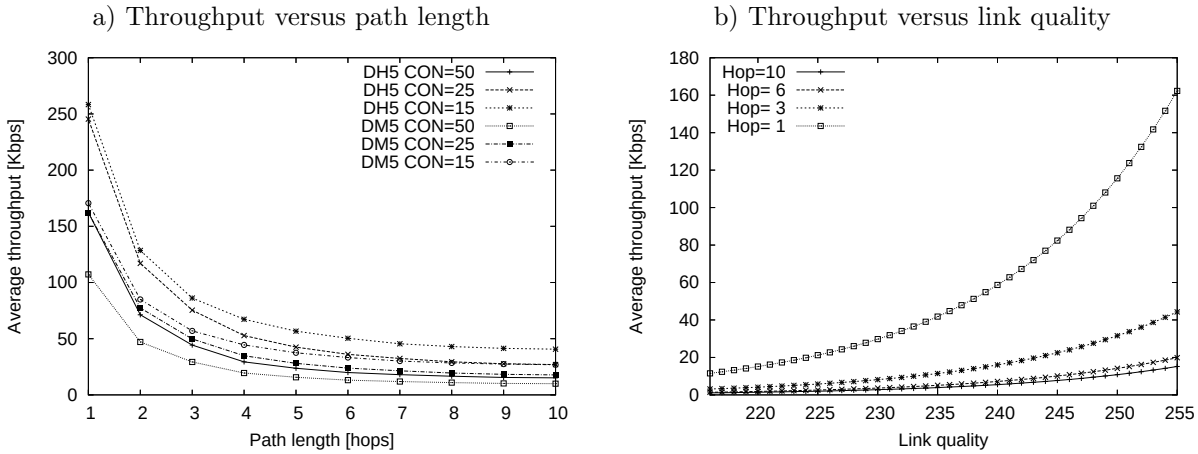

Fig. 1. Estimated throughput evolution as a function of path length (a) and link quality (b)

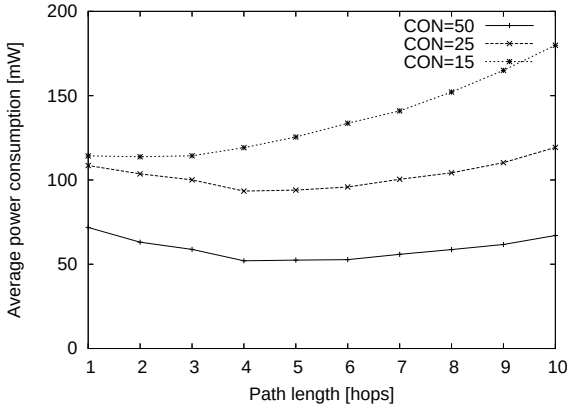

Fig. 2. Power consumption versus connection length

Finally, the power consumption does not depend on the link quality since power is consumed at the same extent for transmissions and retransmissions, because the connections are fully loaded. In case of underloaded connections a relationship between the power consumption and link quality could be observed. However, in our work we opted for heavily loaded connections, since throughput and power consumption issues are more critical in such circumstances.

V. Reducing the Hop Count

We observed that the scatternet throughput and power consumption are in close relation with the communication path lengths. We now present a technique for finding an optimized network topology that can support communications with shorter paths. (More details about these algorithms are available in [10].)

To find an optimized network topology, we first generate a connected and totally functional scatternet by using the algorithm in [2]. After the scatternet formation, based on the nodes' weights, we choose one of the masters, the so-called *optimizer*, to coordinate the optimization procedure. The optimizer collects relevant information about the whole scatternet, such as the identity and the role of the nodes, the nodes neighbors and the communication peers, and feeds them into the optimization algorithm.

The optimizer uses a local search strategy based on a set of possible changes that can be made on the topology, the so-called *moves*. Moves may lead to piconet formation or merging, or just make

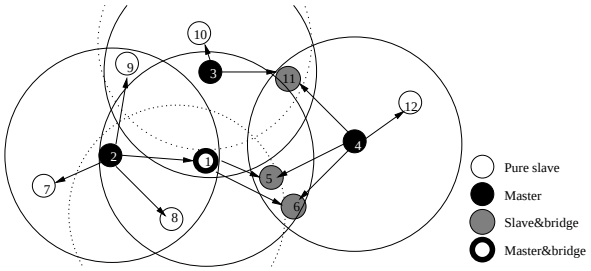

Fig. 3. Scatternet snapshot

slaves move from one piconet to another one. In particular, moves targeting slaves typically increase the number of piconets in the network, while moves targeting masters may merge piconets. Thus, in our optimization procedure we try to reduce the number of hops by first moving slaves and then moving masters. As an example, consider the scatternet shown in Figure 3. If there is a high traffic flow between slaves 8 and 12, then the scatternet can be optimized by removing node 8 from master 2 and assigning it to master 1 instead.

For each move, the optimizer calculates the new value of F, the function to optimize, as defined in Section II. If F decreases after the move, then its value is stored; otherwise it is dropped. At the end of the optimization, the most convenient scatternet configuration, stored during the search, is set.

The optimization algorithm is executed periodically. We call the time between two consecutive executions *optimization period*. Implicit feedback from the scatternet, like the gain of previous optimizations, can be used for dynamically determining the optimization period, e.g., in a scatternet with dynamically changing traffic connections and high node mobility the optimization period will be short, while in quasi-static environments the optimization will be rarely executed.

A. Move Types

A *move* is a set of modifications on the master-slave relationship between nodes in the network. Such modifications are made by link creation, deletion and/or by master-slave role exchange. If, due to these modifications, some nodes get disconnected, the operations necessary to reconnect them to the scatternet are considered as parts of the same move. We identify four kinds of possible moves:

Slave to Slave (SS) – a slave connects to a different master or establishes a new piconet with a node which then exchanges its role from slave to master. Since moving bridge nodes influences considerably the routing scheme of the scatternet we are not moving bridge nodes but only pure slaves. Example (see Figure 3): we want to remove slave 8 from master 2 and assign it to master 1. To this end we set $l_{28} = 0$ and $l_{18} = 1$; i.e., first we cancel the link between master 2 and slave 8, then we create the link between master 1 and slave 8.

Slave to Master (SM) – a slave creates a new piconet by paging another node. Example (see Figure 3): we want to remove slave 8 from master 2, change its role into master and assign slave 5 to it. To this end we set $l_{28} = 0$ then $l_{85} = 1$. This means that we cancel the link between master 2 and slave 8, and create the link between node 8 and slave 5.

Master to Slave (MS) – a master becomes a pure slave. Such a move is possible only if the slaves of the moving master (i.e., the node giving up its role of master) can be assigned to other nodes in the scatternet. The optimizer takes the abandoned slaves (i.e., the slaves of the moving master) one-by-one and assigns them to an already existing master, using SS and SM moves.

Master to Master (MM) – merging two piconets: a master overtakes all of the slaves of another master. Such a move can take place when any node in the two piconets is in the range of the persisting master (i.e., the node maintaining its role of master after the move) and the total number of nodes in the two piconets is not greater than 8. This move can be done by removing from $\mathcal{L}$ all of the 1s

```
1.  OPTIMIZER
2.  L_init ← L
3.  F ← ActualNrHops()
4.  slavelist ← list of slaves (for SX module)
5.  masterlist ← list of masters (for MS and MM)
6.  for k ← 1 to nr_div do
7.  |  call one of SX, MS or MM optimization modules
8.  |  if F > ActualNrHops() then
9.  |  |  F ← ActualNrHops()
10. |  |  L_opt ← L
11. |  L ← L_init
12. |  HUpdate()
13. |  shuffle slavelist (for SX module)
14. |  shuffle masterlist (for MS and MM module)
15. L ← L_opt
16. HUpdate()
17. end OPTIMIZER

18. SX optimization module
19. for each slave i in slavelist do
20. |  execute SS and SM optimization
21. |  perform the best move and update roles

22. MS optimization module
23. for each master i in masterlist do
24. |  execute MS optimization
25. |  update roles

26. MM optimization module
27. for each master i in masterlist do
28. |  execute MM optimization
29. |  perform the best move and update roles
```

Fig. 4. Pseudocode of the optimizer

from the row of the master that is about to become a slave and adding them to the corresponding positions in the row of the persisting master. The old master should be connected to the persisting one through an additional operation. For instance, if node i gives up its role of master and joins the piconet of master j, the additional operation would be $l_{ji} = 1$.

B. The Optimization Procedure

The *optimization procedure* is the core of our work. It determines the modifications to be performed on the scatternet topology in order to reduce the number of hops between communicating nodes.

The optimization algorithm (Figure 4) consists of a main body (lines $1 - 17$) from which our optimization modules, namely SS, SM (either one denoted by SX in Figure 4), MS and MM, can be called. At the beginning of the main body several initializations are performed. First, the initial state of the link matrix $\mathcal{L}$ is saved (line 2). Using the function *ActualNrHops()*, we retrieve the number of weighted hops between each source-destination pair and assign this number to F, our function to be

minimized. In lines $4-5$ all pure slaves and masters are selected and put in *slavelist* and *masterlist*, used later by the SX, MS and MM optimization modules, respectively.

In line 6 we cycle trough the optimization procedure *nr_div* times. At every iteration we must choose only one move, the most favorable one, from a set of mutually excluding moves. Then, we randomly reorder the nodes in *slavelist* and *masterlist* at the end of each iteration, and repeat the search. By doing so, we change the order in which nodes are evaluated, and obtain a new series of moves to evaluate.

Within the cycle, one of the SX, MS or MM optimization modules is executed. Each of these modules performs a local search [6] executing one sequence of moves of the appropriate type. SX evaluates the possibility of moving each slave using both, SS and SM moves (lines $18-21$). The move producing the greatest hop reduction is accepted, and the node roles modified by this move are updated accordingly. Similar operations are performed in the MS and MM modules, except that they act on the *masterlist* performing MS and MM moves, respectively.

A further difference is that for MS moves it is preferable to leave the reassigned slaves at their new master, instead of resetting them at the end of the move and moving them again at a later step of the MS module. This way we can save CPU processing time. Thus, in the MS module the best move should not be re-executed, only the node roles must be updated (line 25).

If the value of F is reduced through the execution of an optimization module, we update F and save the obtained configuration in $\mathcal{L}_{opt}$ (lines $8-10$). Before moving to the next iteration of the *for* loop, we set $\mathcal{L}$ to its initial value (line 11). This requires $\mathcal{H}$ to be updated. *HUpdate()* takes the necessary input data from $\mathcal{L}$, uses Floyd's algorithm for solving the *all-shortest path problem* [5], and stores the result in $\mathcal{H}$. Finally, the nodes in *slavelist* or *masterlist* are reordered (i.e., a *diversification* is done) for the next iteration of the *for* loop.

After the *for* loop, $\mathcal{L}$ is set to the best configuration found, stored in $\mathcal{L}_{opt}$, and the hop matrix is updated.

The optimization algorithm can execute the optimization modules in sequence, combining them in different ways. For instance, if we perform the SX, MS and MM optimization modules, we obtain an optimization algorithm that we refer to as SX_MS_MM. The SX module can also be replaced by an SS or SM module giving the so-called SS_MS_MM and SM_MS_MM optimizations, respectively.

Regardless of the optimization modules used, after executing the optimizer a certain number of times, the so-called diversifications (*nr_div*), we find a scatternet configuration with fewer hops connecting traffic sources to destinations. Our algorithm can guarantee a global optimal configuration only if each optimization module is called for all possible permutations of nodes in the corresponding *slavelist* and *masterlist*. This would take an unacceptably long period of time. Therefore, a good trade-off between the number of diversifications and execution time should be found to achieve acceptable performance in real environments.

C. Reduction of Hops by Using SS and SM Moves

As already mentioned in the previous sections, slave optimizations aim at finding the best possible SS or SM move for reducing the number of weighted hops between a slave *id* and all of its communication peers. During the optimization, *id* is sequentially moved to each node in its radio proximity, except those that are in the same piconet with *id*. Additionally, in the SS algorithm, neighboring masters having 7 slaves already are also excluded from the search.

Consider that slave *id* is connected to master m while the target neighbor, i, is connected to a different master (i.e., $l[m][id] = 1$ and $l[m][i] = 0$). The SS and SM moves alter these settings as follows. In the case of an SS move, i pages slave *id*, while for SM moves slave *id* becomes a master and pages neighbor i. If the value of F after the move is decreased, the move is stored. After the current evaluation, the original links are restored, so that the subsequent move can be executed under the same conditions.

At the end of the procedure, after all neighbors of *id* have been checked, the move minimizing F is returned to the optimizer algorithm.

D. Reduction of Hops by Using MS Moves

The MS optimization algorithm changes the role of a master, *id*, into slave, connects each of *id*'s slaves to a new master and returns a list of performed moves.

In the first phase of the MS algorithm, all slaves of *id* are assigned to a new master that minimizes the number of hops between each slave and its communication peers. If any of the slaves cannot be assigned to some other master, the algorithm terminates, indicating that no MS optimization is possible with master *id*.

In case all slaves were successfully reassigned to masters, *id* itself is also moved to one of its neighboring masters, *i*. To become the new master of *id*, *i* must have less than 7 slaves and minimize the value of F by accommodating *id* in its piconet. Moreover, it should be ensured that after assigning *id* to *i* a path exists between *id* and all of its earlier slaves, in order to keep the scatternet connectivity unaltered. Finally, it should be verified whether *id* is the master of *i* and whether they have a third master in common. The creation of triangles or making two nodes masters of each other must be avoided.

If all of the above conditions are met, master *i* is stored and the search continues with the next neighboring master. After all masters have been checked, the one giving the greatest hop reduction is chosen as the new master of *id*, and the corresponding sequence of moves is returned to the optimizer. If no master observes all of the above conditions, no move with *id*'s slaves is performed, and the algorithm terminates without any hop reduction.

E. Reduction of Hops by Using MM Moves

Although MM moves target masters, the structure of the MM algorithm is similar to the SS and SM procedures. It consists of checking every neighboring master *i* of a master *id* (with the goal of merging their piconets), saving their links to their slaves, checking the hop reduction, and resetting the saved links.

To this end, first we verify whether the total number of slaves in the piconet of *id* and that of a particular master *i* is less than 6. This condition is necessary for conforming to the requirement of at most 8 nodes in the new piconet. Second, we check whether all of the slaves of master *i* are in the range of *id*. If such a master is identified, the piconets can be merged using an MM move.

After all masters have been checked, the master whose piconet merging would reduce the most the number of weighted hops is returned to the optimizer algorithm.

F. Results

To evaluate the performance of our algorithms, we implemented a Bluetooth scatternet simulator in C++. Since the algorithms operate on the scatternet topology, in our simulator we mainly considered topology-related aspects. Thus, we implemented ACL physical links with the a maximum throughput of 723 Kbps; added support for DH1, DH3, DH5, DM1, DM3 and DM5 packet types; links can be set up only if two nodes are at less than 10 m from each other (i.e. they are in radio proximity); six node roles have been considered: free node, pure slave, slave&bridge, master&bridge, pure master and init master (i.e. same as pure master but it initiates piconet formation, see [2] for details since this role is defined for the scatternet formation algorithm described in that paper); slave&bridge and master&bridge nodes can relay data between piconets enabling this way multihop communication; role switch is supported between pure slaves and pure/init masters; the other node types can only change role if they observe the constraints regarding the number of their slaves and masters. During our optimizations role switches happen only in the frame of moves, when verifications are performed

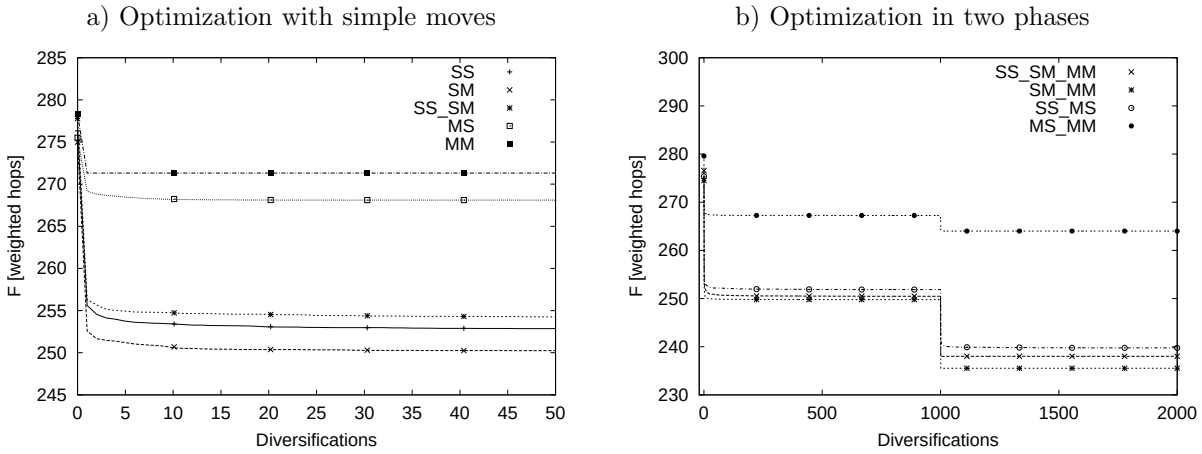

Fig. 5. Optimizations in one (a) and two (b) phases

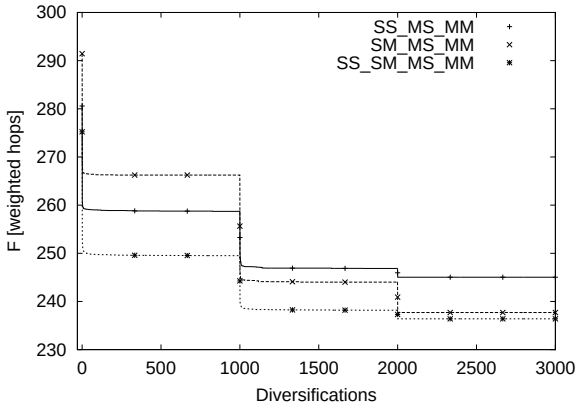

Fig. 6. Optimization in three phases

to ensure the compliance to the specifications. There are up to 7 slaves and one master allowed in a piconet. Low power modes are not yet supported as well as the explicit connection setup procedure that requires the nodes to hop through the inquiry/inquiry scan and page/page scan modes, respectively, before they can communicate.

We tested the optimizer by generating 50 scatternets, of 100 nodes each, over an area of 66×66 m^2. We set the nodes' radio range to 10 m and $nr\_div = 1000$. We randomly generated 100 source-destination traffic connections (elements of $\mathcal{C}$). The traffic intensity (t_{sd}) on these connections is set to 0.1, 0.25 and 0.5 for 50%, 30% and 20% of traffic connections, respectively. All simulations were run on a Linux PC with a 1.7 Ghz CPU and 256 Mb RAM.

We performed experiments combining the SS, SM, MS and MM optimizations in many different ways. For the sake of brevity, here we present only the results derived from some sample optimizations. In particular, we consider three groups of experiments referring to the case where the optimization algorithm is called one, two and three times, respectively, during the same simulation. Each call to the optimizer corresponds to a different optimization module. Figures 5 and 6 show the evolution of function F during these experiments, against the number of diversifications.

Then, for each group of experiments, the one giving the best performance is evaluated in greater details in Tables I-III. In each table, the following metrics are presented. The *Slaves*, *Slave&bridges* and *Total master* parameters report the number of pure slaves, slave&bridges and total number of pure masters and master&bridges, respectively. The parameter *Links* represents the number of links in the scatternet. The *Weighted hops* row shows the overall optimization achieved after each module

TABLE I

Optimization with SM moves

EXPERIMENT #1	**Before**	**After**	**Diff.**	**%**
Pure slaves	32.92	14.32	-18.60	-56.50
Slave&bridges	32.48	33.42	0.94	2.89
Total Masters	34.60	52.26	17.66	51.04
Links	121.32	121.32	0.00	0.00
SM optimization [wh]	274.97	250.22	-24.75	-9.08
Hops [h]	1222.08	1121.68	-100.40	-8.26
Weighted hops [wh]	274.97	250.22	-24.75	-9.08

TABLE II

Optimization with SM_MM moves

EXPERIMENT #2	**Before**	**After**	**Diff.**	**%**
Slaves	33.34	21.20	-12.14	-36.41
Slave&bridges	32.34	38.02	5.68	17.56
Total Masters	34.32	40.78	6.46	18.82
Links	121.04	131.12	10.08	8.33
SM optimization [wh]	274.48	249.81	-24.67	-9.09
MM optimization [wh]	249.81	234.75	-15.06	-6.14
Hops [h]	1221.88	1054.92	-166.96	-13.76
Weighted hops [wh]	274.48	234.75	-39.73	-14.64

of the optimization is terminated, while the *Hops* row presents the corresponding hop count. The distance weighted (i.e., multiplied) by the traffic intensity is expressed in *weighted hops* ([*wh*]), and the distance is measured in *hops* ([*h*]). The rows referring to the SM, MS and/or MM optimizations in the *Before* and *After* columns contain values of the weighted distance in the scatternet configuration exactly before and after that specific phase of the optimization. All other values in these two columns refer to the beginning and the end of the entire optimization procedure. The last two columns indicate the differences between the values in the *Before* and *After* columns, expressed in the appropriate unit (*Diff.*) as well as in percents (*%*).

First, let us consider Figure 5.a and Table I. In Figure 5.a, for the sake of readability we present only the first 50 diversifications out of 1000 (the curves remain flat from that point onward). The plot shows that the SM optimization produces the greatest weighted hop reduction among the simple moves. Master moves (MS and MM), instead, produce small hop reduction. This confirms that in our initial scatternet masters were selected with care. Table I presents the results of the SM optimization, i.e., the most performing among the simple moves. Observe that the SM optimization gives a weighted hop reduction of 9.08%, at the expense of 51% increase in the number of masters (i.e., piconets). In fact, according to their definition, SM moves transform the moving slave into a master creating a new piconet. The number of links instead is unchanged, since SM moves always tear off a link for another.

To improve the performance in terms of weighted distance and keep the number of piconets small, we perform also master moves after having moved the slaves (i.e., after the 1000th iteration). The results are presented in Figure 5.b. It can be seen that the largest hop reduction (14.64%) is obtained through the SM_MM optimization, whose performance is reported in Table II. Table II shows that the 14.64% gain in hop reduction corresponds to 18.82% and 8.33% increase in the number of masters and links, respectively. Notice that the SS_MS optimization produces a lower hop reduction (12.8%) but it increases the number of masters and links by only 1.49% and 0.54%, respectively. This highlights that master moves counterweight the increase in number of masters produced by slave moves.

Figure 6 presents the results of our third experiment, composed of slave optimizations followed by both MS and MM moves (diversifications $1000 \div 1999$ and $2000 \div 3000$, respectively). We obtained the best results with the SM_MS_MM optimization (see Table III). This optimization gives also the

TABLE III

Optimization with SM_MS_MM moves

EXPERIMENT #3	**Before**	**After**	**Diff.**	**%**
Pure slaves	33.02	24.82	-8.20	-24.83
Slave&bridges	32.84	35.68	2.84	8.65
Total masters	34.14	39.50	5.36	15.70
Links	121.42	125.26	3.84	3.16
SM optimization [wh]	278.74	254.47	-24.27	-8.86
MS optimization [wh]	254.47	241.36	-13.11	-4.48
MM optimization [wh]	241.36	235.06	-6.30	-2.67
Hops [h]	1228.24	1047.88	-180.36	-14.34
Weighted hops [wh]	278.74	235.06	-43.68	-15.26

best overall performance. However, if we take into account the average optimization execution times, we have: 26.94, 42.95 and 82.43 minutes for SM, SM_MM and SM_MS_MM, respectively. Thus, we can conclude that it is not worth performing both, MS and MM moves for additional $1-2\%$ of hop reduction.

Finally, we highlight that the step-like behavior of function F in all of the three plots in Figures 5 and 6 suggests that most of the hop reductions happen at the beginning of each call to the optimization algorithm, namely within the first $10-50$ diversifications. Therefore, we can drastically reduce the number of diversifications and, thus the execution times without any significant impact on the overall performance. For example, repeating the SM, SM_MM and SM_MS_MM optimizations with $nr\_div = 10$, the (execution time, reduction) pairs, expressed in $[s,\%]$, will be of $(15.33, 8.67)$, $(21.71, 13.82)$ and $(42.8, 14.29)$, respectively.

VI. Dynamic Scatternets

After presenting our algorithms for reducing path lengths in scatternets, now we aim at evaluating our technique, when repeated executions of the optimizer algorithm are performed in a dynamic network environment. In the following, we present how dynamic traffic connections and mobility have been embedded in the optimization process as well as their impact on the scatternet performance.

A. Dynamic Connections

To deal with dynamic traffic flows, we assign a lifetime to each connection. A connection is replaced with a new one (i.e., with a communication session between different, randomly selected end-nodes), as soon as its associated lifetime expires. This way, we can keep the number of connections constant during the entire simulation and obtain a clear view of the performance improvement achieved by our optimization algorithms (indeed, the number of traffic connections in the scatternet has a major impact on the aggregate throughput and power consumption).

Communicating nodes use the full bandwidth that has been allocated to them; for the bandwidth allocation we employ the max-min fair algorithm [12], [7], taking into account also the constraints imposed by the master-slave communication model of the Bluetooth technology.

Dynamic traffic connections motivate the periodic re-execution of the optimization procedure, which reconfigures the scatternet so as to support more efficiently the newly evolved traffic patterns. Further details on how dynamic traffic connections are embedded in the optimization process can be found in Section VI-C.

B. Mobility

The second element that we consider to study dynamic scatternets is mobility. In our mobility model, during each time unit of the optimization process the location of the nodes may change.

```
1.  OPTIMIZATION PROCESS
2.  set time_unit, sim_time, T
3.  read node parameters
4.  build topology
5.  read initial connection set
6.  while t < sim_time do
7.      call scatternet optimization procedure
8.      calculate link capacities
9.      allocate bandwidth to connections
10.     tper ← 0
11.     while tper < T do
12.         calculate throughput & power cons.
13.         forall connections c do
14.             if t ≥ c.expiration_t then
15.                 remove c
16.                 generate new connection
17.         forall nodes n do
18.             update location of n
19.             fix possible link disruptions of n
20.             update role of n & of its neighbors
21.             reset broken connections of n
22.         remove unused links
23.         calculate link capacities
24.         allocation bandwidth to connections
25.         tper ← tper + time_unit
26.         t ← t + time_unit
27.         if t ≥ sim_time then stop
28. end OPTIMIZATION PROCESS
```

Fig. 7. Pseudo code of the optimization process

Nodes move with walking speed in random directions. Since in this work we only study connected networks, we do not allow nodes move to such new locations that would cause network partitioning. Investigating on the effects of network disruptions on throughput and energy consumption might be an interesting topic for future research.

When we evaluate the scatternet performance without using our hop reduction algorithms, a node, which is disconnected from one of its masters due to mobility, sets up a new link to the first node that it discovers within its neighborhood. Instead, when the optimization algorithms are used, the node sets up a new link to that neighbor that reduces the most the length of its traffic connections.

Further details on how we embedded mobility in the optimization process can be found in Section VI-C.

C. Optimization Process

In the following, we present the operation of a scatternet over time when our optimization algorithms are implemented and in the presence of dynamic traffic flows and node mobility. We refer to this series of scatternet optimization and management operations as the *optimization process* (Figure 7).

The optimization process starts out by setting and initializing several important parameters. In particular, we set the *time_unit* between two consecutive evaluations of the scatternet state; the

time *sim_time* during which we analyze/simulate the behavior of the network; and the optimization period T, which separates two subsequent optimizations. Further, we read all the a priori generated node-specific parameters (including node position and degree of mobility) and after building an initial topology in line 4 (based on the algorithm in [2]), we read from a file an initial set of traffic connections.

In line 6 the *optimization loop* starts, which provides the timestamp, denoted by t, for the optimization process. At each iteration of the optimization loop, t is incremented by one *time_unit* (line 26) until *sim_time* is reached, i.e., the optimization process terminates.

The topology is optimized for the first time at the beginning of the optimization loop (line 7), when the optimization procedure (containing the hop reduction algorithms) is called. Since the optimization procedure modifies the scatternet topology, it should always be followed by the re-calculation of the link capacities (line 8). Link capacities are calculated using the technique presented in Section III. The modified link capacity pattern requires also the reallocation of the available bandwidth among the connections of the scatternet, hence the max-min fair bandwidth allocation algorithm is performed (line 9).

After the bandwidth re-allocation, the optimized scatternet is ready to operate. However, the changing traffic patterns and node mobility gradually modify the configuration for which the network was optimized, reducing the efficiency of the scatternet over time. This requires the optimization procedure to be repeated periodically. The *dynamicity management loop*, starting at line 11, aims at managing the aforementined dynamic behavior of the scatternet. The *dynamicity management loop* starts after the execution of an optimization procedure and when it ends the subsequent optimization procedure is called. Each optimization period (i.e., the time between two optimizations) is T seconds long.

The purpose of the optimization process is to study the variations of the scatternet throughput and power consumption; we calculate these values in line 12.

Two *for* loops follow after the performance calculations. In lines $13-16$ the connection expiration times (denoted by $c.expiration\_t$) are checked and each expired connection is replaced with a new one, with different (random) end nodes. Each newly created connection is initialized with an end-to-end path, expiration time and is allocated a fair amount of bandwidth.

On the other hand, the second *for* loop (lines $17-21$) manages the node mobility. At every iteration of the dynamicity management loop each node is moved to a new location, based on the preset walking speed, the length of the *time_unit* and a random direction (line 18). In case a link is disrupted due to node displacement and the limited Bluetooth radio range, a new link is searched for to reset the connectivity of the node. If such a link is not available, the node relocation is cancelled. However, if the node can be reconnected and its new location is acceptable, the role of the nodes that are involved in the new link creation is updated (line 20). Finally, if a link used by any connection was disrupted, and hence removed, the affected connections are repaired as well, by finding new paths between their end nodes (line 21).

Before recalculating the link capacities and reallocating the bandwidth among the new connections, in line 22 we remove those links from the scatternet that are not used for a predefined time interval (namely, 20 s) by any connections. This is useful to avoid wasting node computational capacity on links that are not used, as well as to counterweight the side-effect of master moves that tend to increase the number of links in the scatternet.

Another side-effect of our optimizations is that slave moves tend to generate new masters. This is not a problem when the optimization procedure is executed only once in a scatternet (as in Section V), because slave moves produce just few new masters. However, when the optimization procedure is executed periodically (as in this section), slave moves may gradually transform all nodes into masters. It is easy to see the undesirable effects of such a phenomenon. Although not explicitly shown in the pseudo code of Figure 7, we counterweight this shortcoming by executing slave moves followed by master moves, which reduce the number of masters by merging piconets.

The optimization process will keep on altering between scatternet optimizations and dynamicity

management periods until *sim_time* expires.

D. Results

To evaluate the performance of scatternets also in dynamic environments we embedded in our simulator dynamic traffic flows and node mobility. We performed experiments with 50 scatternets of 100 nodes each over an area of 66x66 m^2; we assume a radio range of $10m$, DH5 packets, link quality of 255 and 50 bidirectional traffic connections. Furthermore, we set the number of diversifications $nr\_div = 5$, the simulation time $sim_time = 3600$s with $time_unit = 1$s, the duration of connections in the range of $60 - 120$s and the optimization period $T = 120$s.

While in Section V-F we executed the optimization procedure only once during a simulation and the traffic intensity was constant, here the traffic intensity depends on the bandwidth available for each connection and varies over time. Since during a simulation the available bandwidth increases due to hop reductions, the traffic intensity increases accordingly. Thus, we cannot consider anymore the weighted number of hops as a valid performance metric since hop reduction may increase the value of function F (1). Thus, in this section our goal is to reduce the number of pure hops instead of the weighted hops.

Using the above settings, we evaluate the performance of the optimization algorithms with the optimization process presented in Section VI-C, in the presence of dynamic traffic connections and node mobility.

1) Dynamic Connections: Here we consider dynamic traffic connections and static nodes. Averaged simulation results are shown separately for each kind of optimization in Table IV. The most important performance metrics of a sample optimization (SS_MS) are compared to the non-optimized scatternet performance in Figure 8.

TABLE IV

SCATTERNET PERFORMANCE WITH DYNAMIC CONNECTIONS AND STATIC NODES

EXP #1	**Hops**	**Throughp.**	**Power**	**Eff.**	**Gain [%]**	**Masters**	**Links**
No optim.	7.10	18.10	50.14	0.362	–	31.00	101.66
SS_MS	5.08	25.47	43.15	0.591	+63.26	34.46	103.46
SS_MM	5.17	25.16	43.33	0.581	+60.50	35.62	103.98
SM_MS	5.60	24.72	44.42	0.557	+53.87	50.09	100.81
SM_MM	5.16	24.78	43.11	0.575	+58.84	46.47	106.99
SS_MS_MM	5.04	25.12	42.49	0.592	+63.54	33.34	103.61
SM_MS_MM	5.14	25.38	42.58	0.596	+64.64	43.74	104.03

In Table IV the performance of non-optimized scatternets is shown along with the performance of six different types of optimizations, in terms of average path length (i.e., hop count), average throughput, average power consumption, and energy efficiency. We define the scatternet energy efficiency as the ratio of the average throughput to power consumption, thus representing the amount of bits that can be transmitted per energy unit.

As additional metrics, in the Table we show the average number of masters and number of links during the simulations. In the *Gain* column we calculate the efficiency improvement of the different optimizations with respect to the simulations with no optimization. This column clearly shows that all of the six optimization types can provide performance improvements of about 60% with respect to the non-optimized scatternets. We should also notice that the the average number of masters and links are on favor of the non-optimized scatternets. When optimizations are performed, the number of masters and links grow for the reasons explained earlier in this section, however with optimizations containing SS moves this growth is negligible.

It is worth noticing that all six optimizations provide roughly the same performance improvements in the mentioned experiments. If we had to select the most performant optimization type, one should consider that SM_MS_MM optimizations give slightly better results but consist of three phases, while

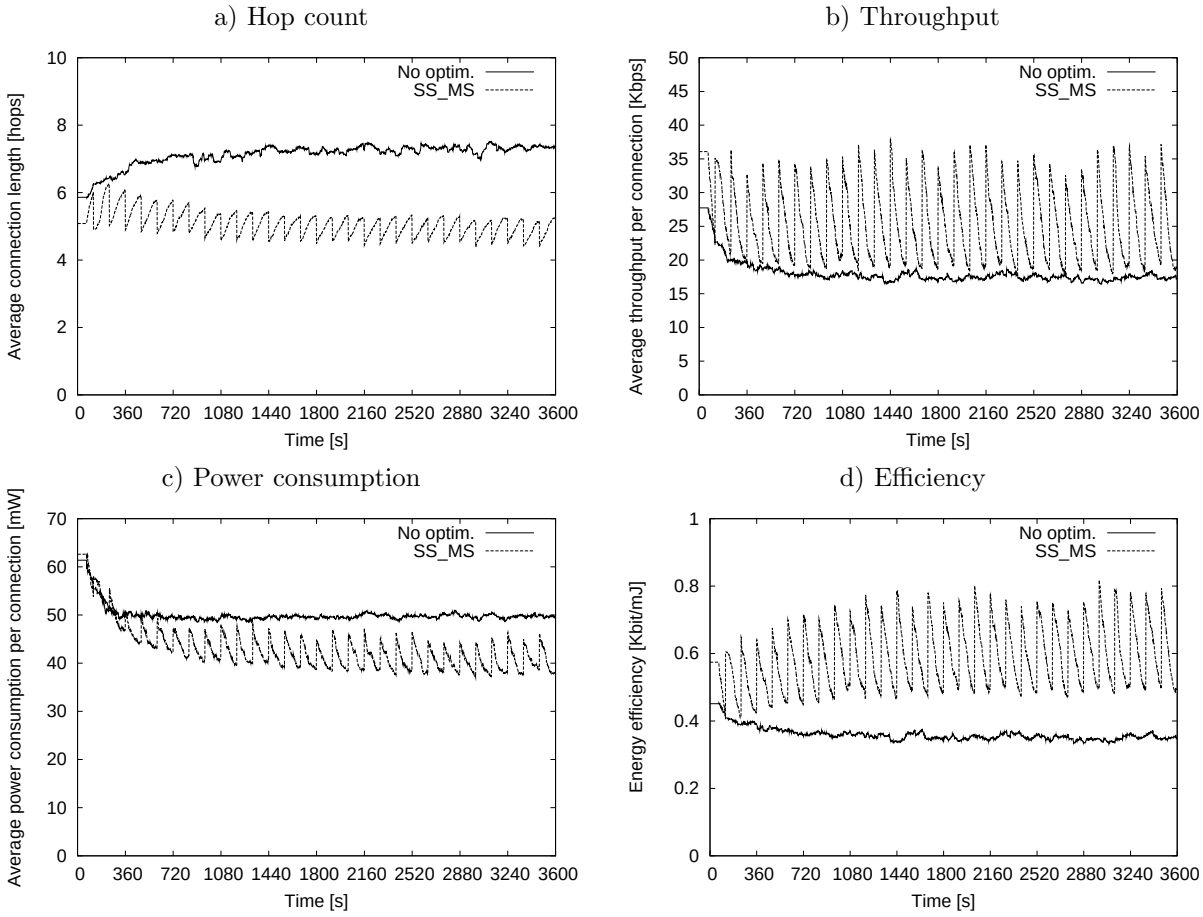

Fig. 8. Scatternet performance in the presence of dynamic connections with and without optimization; a) average hop count; b) average throughput; c) average power consumption; d) energy efficiency.

SS_MS optimizations achieve very close performance in two phases only. In Figure 8 therefore we focus on the performance of the SS_MS optimization.

The optimization process starts with the scatternet formation and the selection of the 50 connections. At the beginning all of the metrics are constant until the first connection expires (notice the short constant segment at the beginning of each curve in Figure 8). As the traffic connections expire and new ones take their places, the performance of the non-optimized scatternets degrades. The throughput (Figure 8.b) decreases and, hence, the power consumption (Figure 8.c) decreases too. The fact that power consumption decreases does not mean that the scatternet performance improves, indeed this decrease is caused by the lower number of transmitted bits. This is clearly explained by the energy efficiency metric (Figure 8.d), which decreases over time.

Looking instead at the optimized scatternets (with the SS_MS optimization in this case), we can see that all metrics, except power consumption, improve each time we execute an optimization. The impact of the optimizations can be seen on the evolution of the hop count (Figure 8.a), which is then reflected in the other performance metrics. Power consumption is an exception because the increased throughput after the optimizations requires more energy to be consumed for transmitting the higher amount of bits. However, the gain on the energy efficiency metric clearly demonstrates that the available energy is used more efficiently after the optimizations.

The plots in Figure 8 clearly show that the scatternet performance between two optimizations gradually degrades due to the changing traffic flows. By periodically executing the hop reduction algorithms, the network performance can be improved by about 60%.

2) Dynamic Connections and Mobility: The experiments with mobile nodes did not provide that good results as those with dynamic connections. This is motivated by the very rapidly changing scatternet topology, since 100 mobile nodes cause frequent link disruptions. It can be seen in the *Gain* columns of Table V and VI that most of the optimizations provide negligible performance improvements and some of them perform somewhat worse even than the non-optimized simulation run. This behavior can be explained considering that the topology of a scatternet, whose nodes are mobile, is not only shaped by the hop reduction algorithms, but also by the link disruptions and creations caused by the mobile nodes. In a nutshell, it is difficult to keep the average number of hops low with hop reduction algorithms because the mobile nodes reconfigure the topology continuously.

Only the SS_MM and SM_MM optimizations resulted in significant performance improvements, however they achieved these results at the cost of a very high number of new links (about the double of the non-optimized case) and masters. This behavior can be explained as follows. The number of masters in the presence of mobility increases even without optimizations since the mobile nodes often get disconnected from their masters and hence they have to form their own piconet made of two nodes, when there is no other master in their range. Thus, if a slave A gets disrupted from its master and forms a new piconet with bridge node B, this small piconet can be dissolved with MS moves as soon as a master C enters the radio proximity of master A. Note that bridge B does not need be reconnected to any master since it participates in multiple piconets. However, with MM optimizations the slaves of A must all be in the range of the same master C. Therefore, with MM optimizations is harder to reduce the number of masters in the scatternet. If both the MM and MS modules are applied, the MS module will dominate the MM one because it has looser constraints. This explains the high number of new masters generated by the two optimizations containing only the MM master module.

TABLE V

SCATTERNET PERFORMANCE WITH STATIC CONNECTIONS AND MOBILE NODES

EXP #2	**Hops**	**Throughp.**	**Power**	**Eff.**	**Gain [%]**	**Masters**	**Links**
No optim.	7.76	23.34	58.94	0.396	–	55.48	121.34
SS_MS	8.53	25.20	64.73	0.389	–1.77	58.48	111.02
SS_MM	5.56	31.53	58.01	0.543	+37.12	73.95	207.75
SM_MS	8.73	25.44	65.74	0.387	–2.27	58.10	107.59
SM_MM	5.47	31.79	57.61	0.552	+39.39	74.65	213.09
SS_MS_MM	8.05	25.56	62.99	0.406	+2.52	59.80	118.16
SM_MS_MM	8.02	25.50	62.89	0.406	+2.52	60.19	119.02

TABLE VI

SCATTERNET PERFORMANCE WITH DYNAMIC CONNECTIONS AND MOBILE NODES

EXP #3	**Hops**	**Throughp.**	**Power**	**Eff.**	**Gain [%]**	**Masters**	**Links**
No optim.	7.94	22.21	59.29	0.375	–	55.60	121.34
SS_MS	8.62	23.97	64.67	0.371	–1.07	58.46	110.99
SS_MM	5.54	31.33	57.78	0.542	+44.53	74.11	209.10
SM_MS	8.96	23.80	66.06	0.360	–4.00	58.09	107.46
SM_MM	5.46	31.53	57.60	0.547	+45.87	74.42	211.38
SS_MS_MM	8.18	24.46	63.21	0.387	+3.20	59.93	118.38
SM_MS_MM	8.09	24.47	62.59	0.391	+4.27	60.08	119.11

The high number of links originates from the tendency of master optimizations of generating new links. It looks like most of these newly generated links are used by the traffic connections, because otherwise our mechanism that removes unused links would have eliminated them. Indeed, the high energy efficiency obtained by the SS_MM and SM_MM optimizations can be explained only with the fact that in mesh topologies with many links shorter paths can be found between the communicating nodes than in scatternets with few links.

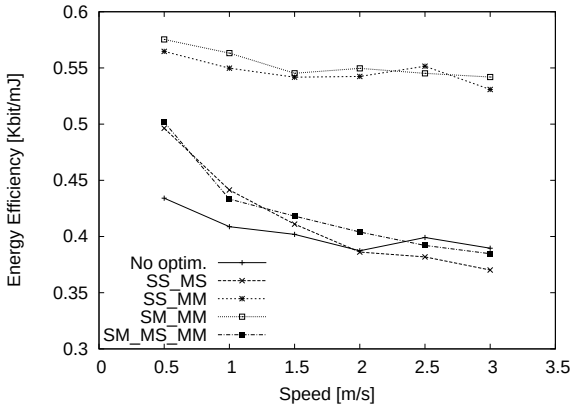

Fig. 9. Energy efficiency versus moving speed

We also performed experiments as the nodes speed varies. The results presented in Figure 9 refer to walking speeds in the range $[0.5, 3]$m/s. As expected, optimizations including MM moves and no MS moves, provided more significant performance improvements for the reasons explained above. Figure 9 also shows that the scatternet performance degrades as the nodes speed increases.

We can conclude that even with low moving speeds it is hard to maintain an optimal scatternet toplogy; this is certainly an interesting problem to address in future research.

VII. Conclusion

In this paper we addressed scatternet optimization techniques that aim at reducing the path length between communicating nodes. Node mobility and changing traffic connections generate permanently changing traffic flows; thus, to provide efficient communications it is not enough to form an optimal network topology, rather a scatternet that best supports the current traffic flows as they vary in time is required.

Intuitively, the scatternet performance, in terms of throughput and power consumption, highly depends on the hop count separating the communicating peers. In this paper we demonstrated the above intuition analytically. Then, simulations were required to get an insight in the actual scatternet performance variations as a function of the hop count. To this end, we developed a heuristic algorithm suite that reconfigures the nodes role and network links so as to minimize the number of hops between communicating nodes in the scatternet. As part of the algorithm suite, we presented two algorithms for reducing the length of connections belonging to slaves and other two for master connections. These algorithms are based on four "move" types (i.e., elementary modules for reconfiguring scatternets) and a matrix-based scatternet model, both defined in this paper.

Finally, to study the scatternets behavior over time, we extended our optimization algorithms to support dynamic traffic flows and node mobility. The resulting optimization process provides a systematic approach to scatternet optimization not only right after the network formation, but also later, when the scatternet performance degrades because of the dynamic user behavior.

Simulation results showed that our hop reduction algorithms can reduce the aggregate hop count in the scatternet by about 15%, and lead a performance improvement of above 60% under moderately dynamic scenarios.

The centralized nature of our optimizations is their main weakness since they require to collect all scatternet information at a single node and distribute the solutions to all other nodes; they are therefore suitable for static or slowly changing environments. To add more flexibility to our approach as well as for improving its efficiency for the future we plan to devise a decentralized solution for our optimization technique. In that approach we also plan to provide additional evaluations about our solution, mainly regarding the overhead and the operation with underloaded connections.

Acknowledgements

This work was partially funded by the Autonomous Province of Trento through the WILMA project and the "Mesh meets ad hoc: The urban vehicle grid" project funded by the ST Microelectronics Inc.

References

[1] Marco Ajmone Marsan, Carla-Fabiana Chiasserini, Antonio Nucci, Giuliana Carello, and Luigi De Giovanni. Optimizing the topology of Bluetooth wireless personal area networks. In *INFOCOM*, New York, USA, June 2002.

[2] Stefano Basagni and Chiara Petrioli. A scatternet formation protocol for ad hoc networks of Bluetooth devices. In *IEEE Vehicular Technology Conference (VTC)*, pages 424–428, 2002.

[3] Bluetooth SIG. *Bluetooth Specification v1.2*. May 2003.

[4] Ling-Jyh Chen, R. Kapoor, M. Y. Sanadidi, and Mario Gerla. Enhancing Bluetooth TCP throughput via link layer packet adaptation. In *International Conference on Communications (ICC'04)*, Paris, France, June 2004.

[5] Ian Foster. *Designing and Building Parallel Programs*, chapter 3.9. On-line edition, 1995.

[6] Johann Hurink. *Introduction to Local Search*. Lecture notes, 2001.

[7] Jeffrey Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, July 1981.

[8] M. Kalia, S. Garg, and R. Shorey. Scatternet structure and inter-piconet communication in the Bluetooth system. In *IEEE National Conference on Communications*, New Delhi, India, 2000.

[9] Csaba Kiss Kalló, Sewook Jung, Ling-Jyh Chen, Mauro Brunato, and Mario Gerla. Throughput, energy and path length tradeoffs in Bluetooth scatternets. In *Accepted for publication at the IEEE International Conference on Communications (ICC'05)*, Seoul, Korea, May 2005.

[10] Csaba Kiss Kalló, Carla-Fabiana Chiasserini, Roberto Battiti, and Marco Ajmone Marsan. Reducing the number of hops between communication peers in a Bluetooth scatternet. In *IEEE Wireless Communications and Networking Conference (WCNC'04)*, Atlanta, GA, USA, March 2004.

[11] Ching Law and Kai-Yeung Siu. A Bluetooth scatternet formation algorithm. In *IEEE Symposium on Ad Hoc Wireless Networks 2001*, San Antonio, TX, USA, November 2001.

[12] Qingming Ma, Peter Steenkiste, and Hui Zhang. Routing high-bandwidth traffic in max-min fair share networks. In *SIGCOMM*, pages 206–217, Stanford, CA, USA, August 1996.

[13] Elena Pagani, Gian Paolo Rossi, and Stefano Tebaldi. An on-demand Bluetooth scatternet formation algorithm. In Roberto Battiti, Marco Conti, and Renato Lo Cigno, editors, *Wireless On-Demand Network Systems*, pages 130–143. Springer-Verlag, 2004.

[14] Chiara Petrioli, Stefano Basagni, and Imrich Chlamtac. BlueMesh: Degree-constrained multi-hop scatternet formation for Bluetooth networks. *Mobile Networks and Applications*, 9:33–47, February 2004.

[15] András Rácz, György Miklós, Ferenc Kubinszky, and András Valkó. A pseudo random coordinated scheduling algorithm for Bluetooth scatternets. In *ACM Symposium on Mobile AD Hoc Networking and Computing*, volume 99, pages 1–100, Long-Beach, CA, USA, 2001.

[16] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed topology construction of Bluetooth personal area networks. In *IEEE INFOCOM*, Anchorage, April 2001.

[17] Ivan Stojmenovic. Dominating set based Bluetooth scatternet formation with localized maintenance. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, USA, April 2002.

[18] Godfrey Tan, Allen Miu, John Guttag, and Hari Balakrishnan. An efficient scatternet formation algorithm for dynamic environments. In *Proceedings of IASTED Communications and Computer Networks (CCN)*, Cambridge, MA, USA, November 2002.

[19] Z. Wang, R.J. Thomas, and Z. Haas. Bluenet – a new scatternet formation scheme. In *35th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, 2002.

[20] Gergely. V. Záruba, Stefano Basagni, and Imrich Chlamtac. Bluetrees - scatternet formation to enable Bluetooth-based ad hoc networks. In *ICC 2001*, volume 99, pages 273–7, 2001.