Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Communications 28 (2005) 1276–1291
computer communications
www.elsevier.com/locate/comcom

SF-DeviL: an algorithm for energy-efficient Bluetooth scatternet formation and maintenance

Canan Pamuk, Ezhan Karasan*

Department of Electrical and Electronics Engineering, Bilkent University, 06800 Ankara, Turkey

Received 21 July 2004; accepted 21 July 2004
Available online 11 September 2004

Abstract

Bluetooth is a short-range ad hoc networking technology, which enables formation of inexpensive personal area networks with low power consumption. Using Bluetooth technology, a small number of closely located devices can be interconnected within a piconet. Building larger ad hoc networks is possible by interconnecting multiple piconets to form a scatternet. As the Bluetooth topology grows from isolated piconets to a scatternet, energy-efficiency becomes a critical issue since additional power is consumed for multi-hop routing. A scatternet should be formed in such a way that batteries of mobile devices are efficiently used in order to lengthen scatternet lifetime.

We discuss the problem of energy-efficient topology construction and maintenance for Bluetooth scatternets. An energy-efficient, distributed Bluetooth *S*catternet *F*ormation algorithm based on *Dev*ice and *L*ink characteristics (SF-DeviL) is presented. SF-DeviL forms scatternets with tree topologies and increases battery lifetimes of devices by using device types, battery levels and received signal strengths. The topology is dynamically reconfigured in SF-DeviL so that energy efficiency is maintained during the lifetime of the scatternet. It is shown through simulations that even without performing reconfiguration the network lifetime is increased by at least 229% compared to LMS algorithm and increased by at least 10% compared to BlueMesh algorithm in heterogeneous networks.

Keywords: Bluetooth; Scatternet formation and maintenance; Energy-efficient topology construction

1. Introduction

The widespread usage of information intensive consumer devices necessitates introduction of networking technologies for interconnection of these modules. Mobility of devices and variety of applications have led to wireless ad hoc networking solutions, where the network is formed without requiring a manual configuration and a wired/wireless infrastructure. A short-range wireless networking solution is useful in personal area networks to interconnect a laptop with a mouse or a digital camera, in a smart home network to interconnect a gateway/controller with home appliances, in sensor networks, etc. One of the candidate solutions for providing networking services to these types of applications is Bluetooth.

Bluetooth is a short-range (10–100 m) wireless ad hoc network technology that supports both voice and data communication [1]. Bluetooth operates in the unlicensed 2.4 GHz ISM band and employs fast frequency hopping spread spectrum (FHSS) technique which provides robustness against interference and fading. Technical features of Bluetooth such as non-line-of-sight communication, low-power consumption, low cost, and higher security (due to FHSS) are the main advantages of Bluetooth over other competing technologies, such as IrDa, IE801.11b (WiFi) and HomeRF.

The basic network architecture of Bluetooth is a *piconet*, which consists of a master and up to seven active slave nodes. The master controls intra-piconet communication by polling the slaves. Bluetooth also enables inter-piconet communication by forming scatternets. *Scatternet* is the network formed by interconnecting piconets through *bridge* nodes. A sample scatternet architecture with different bridge configurations is illustrated in Fig. 1. Scatternets allow

* Corresponding author. Tel.: +90 312 2901308; fax: +90 312 2664192.
E-mail address: ezhan@ee.bilkent.edu.tr (E. Karasan).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.07.026

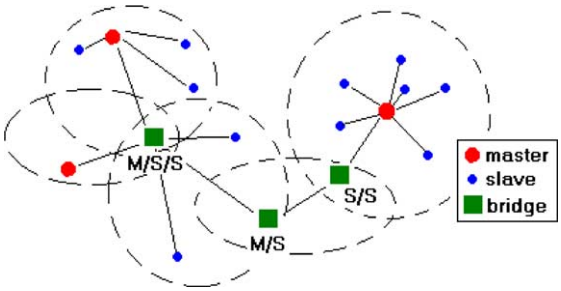

Fig. 1. Illustration of a scatternet with bridge nodes undertaking different roles (M/S, master in one piconet and slave in the other; S/S, slave in both piconets; M/S/S, master in one piconet and slave in the other two).

communication among hundreds of Bluetooth enabled devices. Furthermore, if the scatternet allows multi-hopping, i.e. each node does not have to be within the transmission range of all other nodes, connectivity over distances greater than the short radio range can be provided.

The Bluetooth standard enables formation of scatternets, but it does not define an exact method [1]. The problem of scatternet formation can be stated as the assignment of master, slave and bridge roles to Bluetooth nodes and the determination of links to be established between nodes. Some of the factors that make scatternet formation more challenging are mobility of devices, low computational and energy resources of devices, devices with no prior knowledge about other nodes, requirement to form the scatternet within a tolerable delay, requirement to set up each link before data can be exchanged (due to frequency hopping channel).

Bluetooth technology requires a solution to the scatternet formation problem in order to be considered as a candidate for a larger range of applications. One possible application area for Bluetooth is low-power sensor networks. Experiments conducted with Bluetooth wireless sensor networks point out that the lack of scatternet support is a deficiency for usage of Bluetooth in sensor networks [2–5]. It is also indicated that the scatternet formation method needs to make a distinction among gateways and different sensor types during scatternet formation for preventing link congestion and buffer overflow in intermediate sensor nodes [3].

A wide variety of solutions for the scatternet formation problem are proposed in the literature. Centralized algorithms are not suitable for dynamic Bluetooth networks because knowledge of neighboring nodes and their positions is difficult to obtain on a continuous basis. Some of the proposed algorithms are restricted to single-hop configurations where all nodes are required to directly communicate with each other. The topologies formed by different algorithms also demonstrate variations such as tree or mesh formations.

Energy efficiency is one of the most important aspects of Bluetooth operation since mobile devices rely on batteries. Battery depletion for a given device is undesired from that specific user's perspective, and it may also require reconfiguration of the whole topology when the remaining network becomes disconnected. Energy efficiency can be measured in terms of the *lifetime of a scatternet*, which is defined as the duration until one of the Bluetooth devices exhausts its battery. Both the constructed topology and routing decisions play an important role on the lifetime of the scatternet. Possible methods for lengthening scatternet lifetime are energy-efficient topology construction, dynamic reconfiguration, power-aware routing and scheduling.

In this paper, we present a multi-hop, distributed scatternet formation and maintenance algorithm called SF-DeviL, which efficiently manages battery powers of devices in order to increase the scatternet lifetime. SF-DeviL is compatible with Bluetooth specifications [1]. It uses device characteristics (class of device, battery capacity and level) and link features (received signal strength) together with power control, in order to achieve energy efficiency. Master, slave and bridge roles are assigned based on the device types of the nodes. The links in the topology are determined such that potential links with lower transmit power requirements are given higher priority for establishment. Minimum transmit power for each candidate link is obtained from the quantized measurements of the received signal strength. One of the important features of SF-DeviL is that slave nodes select their masters. SF-DeviL reconfigures the scatternet topology as the battery levels deplete and positions of devices change in order to maintain energy-efficiency. Simulations show that SF-DeviL increases scatternet lifetime by at least 229% compared to the LMS algorithm [6] and by at least 10% compared to the BlueMesh algorithm [7] in heterogeneous networks even topology reconfiguration is not performed. In homogeneous networks, scatternet lifetime is increased by up to 24% compared to LMS, whereas it is decreased compared to the BlueMesh algorithm. Lifetime is increased further by up to 56% in heterogeneous networks and by 75–410% in homogeneous networks when the scatternet topology is reconfigured in response to changing battery levels.

The rest of the paper is organized as follows. Proposed solutions for Bluetooth scatternet formation are reviewed in Section 3. SF-DeviL is introduced in Section 4 as an energy-efficient algorithm for scatternet formation. Scatternet maintenance for SF-DeviL in response to depleting batteries and mobility is described in Section 5. Simulation results are presented for comparing performances of SF-DeviL with other scatternet formation algorithms in Section 6.

2. Scatternet formation algorithms

Bluetooth scatternet formation has recently attracted significant attention, where existing studies can be classified as formation algorithms [6–12] and performance related studies [13–16]. The algorithms for Bluetooth scatternet formation show differences in their approaches.

A centralized approach [8] needs extensive messaging between nodes, and hence it is impractical to use centralized algorithms in dynamic environments. Distributed techniques provide the most appropriate solution for constructing scatternets. In single-hop scatternet formation algorithms, it is assumed that all nodes are within communication range of other nodes [6,9]. LMS [6], which tries to minimize the number of piconets, and TSF [9], are distributed single-hop scatternet formation algorithms that result in tree topologies, and they can maintain topology changes such as node additions and deletions (failures). Algorithms with multi-hop scatternets [7,10–12] do not require the assumption that all nodes are within communication range of other nodes, and thus they have a wider application range.

Algorithms also differ in the resulting scatternet topology: some with tree [6,8–11] and some with mesh topologies [7,12]. It is shown that the optimum Bluetooth scatternet topologies are application dependent [13]. Two distributed, multi-hop scatternet formation protocols resulting in tree topologies, called Bluetrees, are proposed in Ref. [11]. A multi-hop solution, called BlueMesh, which generates mesh topologies, is proposed in Ref. [7]. In the BlueMesh algorithm, the scatternet is formed in two phases: first, one and two-hop neighboring devices are discovered, and then the piconets and their interconnection are provided by selected gateways. This protocol assigns a 'weight' to each device, which is used for selection of slaves.

A distributed, multi-hop topology construction algorithm, called SF-DeviL, which forms scatternets with tree topologies, is proposed in Ref. [10]. SF-DeviL has two phases: each node discovers its neighbors and selects its master in the first phase, and disconnected trees obtained at the end of the first stage are merged in the second phase. Battery levels and classes of devices are used to assign roles to Bluetooth units, and received signal power levels from neighboring devices are used for determining the links to be established. The resulting algorithm is shown to form energy-efficient scatternets with increased lifetimes.

A dominating set-based scatternet formation protocol with localized maintenance property is proposed in Ref. [12]. The goal of this protocol is to form a scatternet with localized maintenance such that local position changes do not trigger global updates.

Energy-efficient techniques for routing in Bluetooth scatternets have been investigated, and it is shown that a considerable gain in network lifetime can be achieved by using distance-based power control and battery level-based master/slave switch [17]. This study assumes that all the nodes in a piconet are within listening distance of each other in order to avoid reconfigurations of the topology every time a master/slave switch takes place. It is also assumed that the distance between a master and a slave is known both to the master and slave.

3. SF-DeviL: energy-efficient scatternet formation algorithm

SF-DeviL forms a scatternet such that efficient usage of device batteries throughout scatternet operation is maintained. Battery capacities of devices and transmission powers for potential links are considered while forming the scatternet. In SF-DeviL, each device selects the best master for itself. Each device selects its own master resulting in a tree topology with leaf nodes undertaking slave roles, intermediate nodes being M/S type bridges and the root node undertaking the master role.

Below, we first define the algorithm parameters, and then the main aspects of the best master selection are described. The SF-DeviL algorithm is explained next, and finally it is proven that the algorithm generates connected scatternets.

3.1. Algorithm parameters

SF-DeviL quantifies device and link specific features using two parameters: device grade and received signal strength grade.

3.1.1. Device grade (DG)

Each device in the network is assigned a Device Grade (DG) using the 'class of device' and battery level information. The class of a device can reveal many features of the node such as mobility, traffic generation rate and battery capacity. For example, a laptop has a larger battery capacity than a mobile phone, and it most likely generates more traffic. In a sensor network, a video sensor typically generates more traffic than a temperature sensor.

In SF-DeviL, each Bluetooth unit calculates its DG by using the following expression:

$$\mathrm{DG} = \text{Battery Capacity} * \text{Battery Level} + \text{Traffic Generation Grade}$$

where BatteryCapacity indicates the capacity of the device battery, BatteryLevel represents the fraction of remaining battery energy and TrafficGenerationGrade is a prediction of the amount of traffic generated by the device. Battery Capacity and TrafficGenerationGrade are specific to the class of the device. Devices with larger and/or fuller batteries and higher traffic generation rates have larger DGs.

Each Bluetooth module knows its device class, and this information is exchanged with neighboring devices during connection establishment by using the 24-bit class of device/service (CoD) field in the FHS packet [1]. CoD field consists of major and minor device class fields together with a service class field. Some examples of major classes defined in Bluetooth specifications are computer, phone, peripheral, LAN/Network Access point, sensor, etc. Some minor classes of the sensor major class are video, temperature, motion, pressure, conductance, force, sound, etc. This way various types of devices are identified using

major and minor device classes. We assume that the BatteryLevel information is also embedded onto some reserved bits of the FHS packet. Thus, two devices that establish a connection can calculate DGs of each other.

3.1.2. Received signal strength grade (RSSG)

Bluetooth supports power control, where transmission power can be lowered as long as reliable communication is provided. The power control capability for Bluetooth modules is mandatory when the transmitted power is over 1 mW, and it is optional when transmit power is under 1 mW. Power control can be used for optimizing the system interference and energy-efficiency. A Bluetooth transceiver that supports power-controlled links has a receiver signal strength indicator (RSSI) that measures the strength of the received signal [1]. In SF-DeviL, each device assigns a received signal strength grade (RSSG) to each neighboring device, based on the measured RSSI for each link. RSSG is quantized according to the strength of the received signal as: weak (W), medium (M), strong (S) and very strong (VS).

3.2. Best master selection

Using DG and RSSG, each device chooses itself a master, i.e. slaves choose their master based on DG and RSSG information. The selection of the 'best master' is done by comparing DG and RSSG of a discovered neighbor with the corresponding values for the current master. The flow chart for the BestMaster selection procedure is given in Fig. 2 for the generic node X. The BestMaster selection is done based on the following observations that are also illustrated in Fig. 3:

1. A device with high DG is more appropriate for becoming a master since it has higher battery capacity, battery level and/or traffic generation rate. As illustrated in Fig. 3a, a reasonable topology is constructed if each device chooses its master as a device with higher or equal DG.
2. Establishing links with lower path loss provides advantages since transmission power and interference can be reduced by using power control. Fig. 3b illustrates a case, where mobile phones choose a laptop, which is closer, as their master instead of a desktop, which is further away.

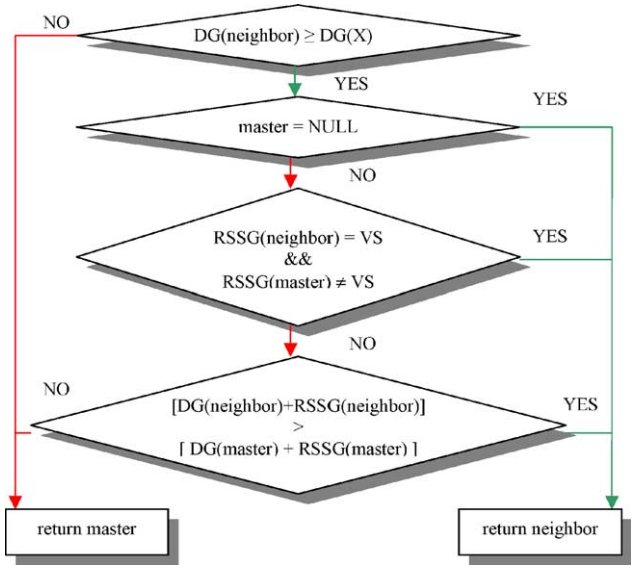

Fig. 2. Flow chart for the BestMaster selection procedure.

BestDevice(master, neighbor) is the procedure for determining the most suitable master for X. The BestMaster selection procedure chooses the better node between the current master and a newly discovered neighbor. A discovered neighbor is selected as the master only if it has a larger or equal DG compared to X. This ensures that a scenario such as the one in Fig. 3a(i) is avoided. When DGs are equal, the device with larger number of slaves or larger BD_ADDR (in case of equal number of slaves) is selected as the master.

A link with RSSG = VS, i.e. a link where a very strong signal is received, has priority over other links. This increases the likelihood that links between devices receiving strong signals from each other are established, so that less power is consumed for transmitting signals, thereby increasing the lifetime of the scatternet and also reducing interference to other systems such as WiFi. The node master with the largest sum of RSSG and DG is chosen as the master. Using this rule, a closer video sensor can be chosen as the master instead of a far away gateway.

The BestMaster selection procedure allows each device to have a single master. This results in a tree topology where only M/S type of bridge nodes exists, i.e. S/S bridges are not used. The details of the algorithm are described in the next part.

3.3. Algorithm for scatternet formation

SF-DeviL is a two-phase algorithm:

I. *Neighbor discovery*. During this phase, each node continuously tries to discover other devices. Each time a new neighbor is discovered, the *better* master for the node is determined by choosing between its current master and the newly discovered neighbor using the *BestMaster* selection procedure. This phase ends when the discovery timeout (discTO) is reached. At the conclusion of this phase, each device should have chosen a master and connected to it.

II. *Merging*. In the beginning of the second phase, each device has either found a master, or it has declared itself as the root of the scatternet. In the Merging phase, paging procedures are initiated by the nodes that have no assigned master, so that disconnected trees resulting after the first phase, are merged.

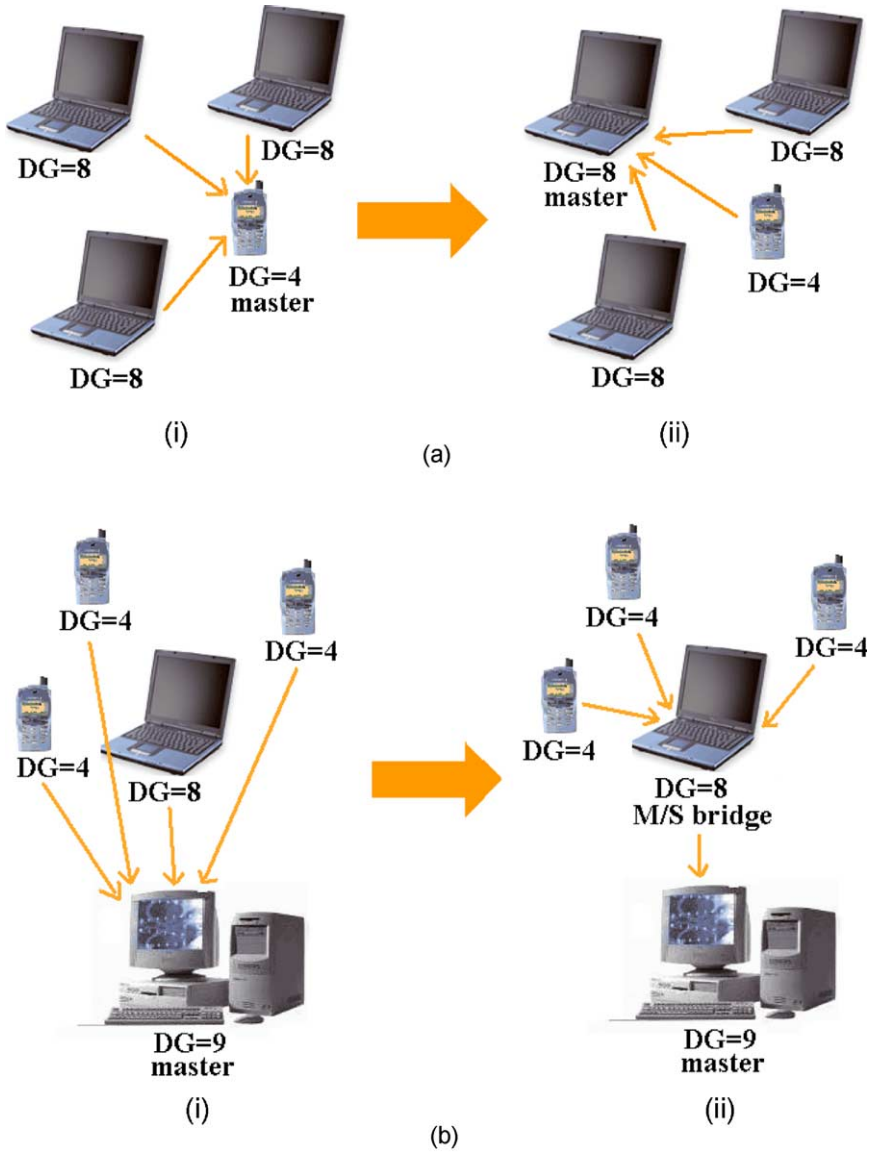

Fig. 3. (a) Piconet (scatternet) formation based on device characteristics, and (b) scatternet formation based on link characteristics.

This phase ends when all disconnected trees are combined into a scatternet.

The details of these two phases are described below.

3.3.1. Neighbor discovery phase

SF-DeviL is a distributed algorithm where each device upon initialization starts the MAIN procedure given in Table 1. The generic device X calculates its DG and starts device discovery by alternating between I/IS modes [1,8] until a neighboring device Y is found. Upon establishment of a link to node Y, X executes the ArrangeRoles(X,Y) procedure given in Table 2.

Using the ArrangeRoles(X,Y) procedure, either the master–slave roles for the established link X–Y are chosen, or the link is deleted. Node X gets the BD_ADDR and DG of Y and computes RSSG of link X–Y. The BS_ADDR and DG are exchanged through FHS packets, whereas RSSG is obtained from RSSI measurements during the connection establishment procedure. X stores the entry of Y in a list, called neighbor_list(X), where all neighboring device information is kept.

In line 2 of the ArrangeRoles procedure, X uses the BestMaster procedure to select itself the best master. If Y is chosen, X frees itself from its current master and becomes the slave of Y. If X is the inquirer (which implies that X becomes the master of Y automatically after connection establishment), X and Y additionally switch master/slave roles for X to become the slave of Y. This is done to ensure that 'better nodes' become masters. According to the BestMaster procedure, Y being the best master of X implies that Y is also a better node than X. If Y is not a better master

Table 1
Main procedure of SF-Devil

```
MAIN:
Phase 1: Neighbor Discovery
1  Upon initialization, calculate DG(X)
2  timer ← discTO
3  while timer > 0 and no neighbor discovered yet
4       Alternate between inquiry and inquiry scan states (I/IS)
5       if a device Y is discovered
6          then  establish link to Y
7                ArrangeRoles(X,Y)
8                timer ← discTO
Phase 2: Merging
9  if  X has no master
10     then execute Semiroot(X)
11      else  exit
```

for X, X requests Y to disconnect from itself. If both nodes X and Y request for disconnection, the newly established X–Y link is broken.

During the BestMaster selection, two devices are considered as candidates for becoming the master, and the link with the worse candidate is terminated after the selection. Finally, each slave has a single master, resulting in a tree topology, where intermediate nodes have M/S bridge roles. The leaf nodes have slave roles, and the root has the master role. Since a bridge participates in just two piconets, it does not become a bottleneck between a large number of piconets.

Node X continues with the discovery of neighbors, seeking the best master for itself by comparing newly discovered neighbors with its current master. X forms a list of its discovered neighbors by adding the discovered devices to its neighbor_list(X). The first phase continues until the discovery timeout (discTO) is reached.

The execution of SF-DeviL is illustrated by an example in Fig. 4. The node locations are shown in Fig. 4a where the nodes are labeled by BD_ADDR.DG. The established links during the first phase are shown as dashed lines and labeled by the time sequence of their establishment and the corresponding RSSG. The values corresponding to RSSGs are assigned as: VS = 3, S = 2, M = 1 and W = 0. Nodes A and B are the first nodes to discover each other. They establish link 1, which has a low path loss corresponding to RSSG = VS, and run the ArrangeRoles procedure. By this procedure, node A adds the entry B.4.VS to neighbor_list(A) and node B adds the entry A.6.VS to neighbor_list(B). Afterwards, B runs the BestMaster selection procedure, by which it chooses node A as its best master since DG(A) > DG(B) and node B has no previous master. If B has been the inquirer during the connection establishment procedure, a master-slave switch is done at link 1 since the inquirer becomes the master by default in the Bluetooth link

Table 2
ArrangeRoles procedure

```
ArrangeRoles( device X, device Y)
1  Add BD_ADDR(Y).DG(Y).RSSG(Y) to neighbor_list(X)
2  if  (Y==BestMaster(current master of X, Y) )
3       then delete link to the current master
4            if  X is the inquirer
5                 then do master/slave switch
6       else  request Y to disconnect from itself
```

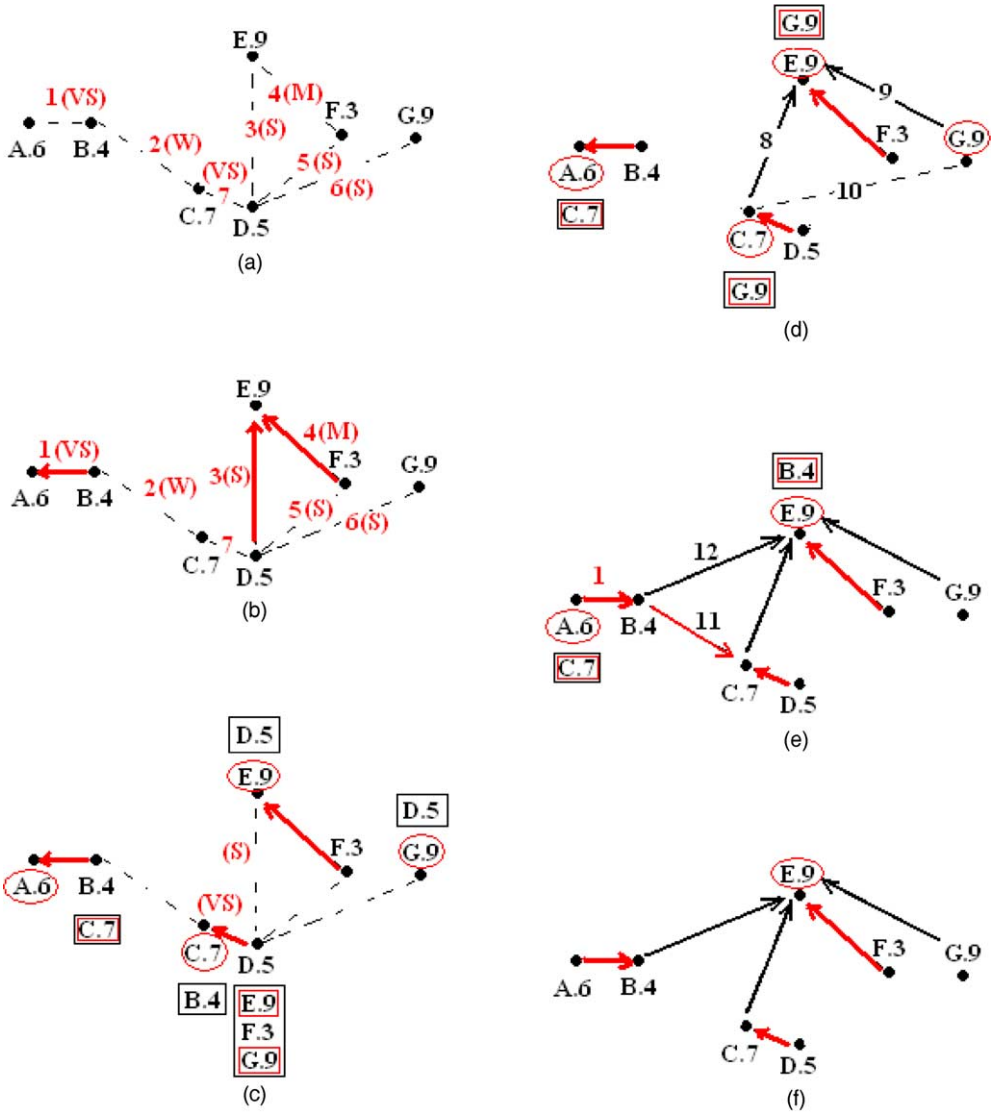

Fig. 4. Illustration of SF-DeviL by an example: (a) nodes labeled as BD_ADDR.DG and dashed links corresponding to discovered neighbors labeled by the sequence of discovery and RSSG; (b) resulting links after establishment of links 1–6; (c) resulting topology at the end of the Neighbor discovery phase and the unconnected but heard nodes in neighbor_lists; (d) topology after the execution of the P–PS(., true) procedure in line 1 of the Semiroot procedure; (e) topology just before the selection of the better path during the execution of the P–PS(., false) procedure in line 4 of the Semiroot procedure; and (f) resulting scatternet.

establishment procedure. Meanwhile, node A decides that node B is not a suitable master for itself through the BestMaster selection procedure since DG(B) < DG(A) and thus requests node B to disconnect from itself by line 6 of ArrangeRoles procedure. Since node B selects node A as its master, this disconnection request is not accepted by B, and link 1 is kept for which nodes A is the master of node B. (The established links in Fig. 4b–f are indicated by arrows directed from slaves to masters.)

Link 2 with RSSG = W, is established after nodes B and C discover each other, and the entry B.4.W is added to neighbor_list(C) and C.7.W is added to neighbor_list(B). Through the BestMaster selection procedure, node B decides that node A is a better master for itself than node C since the link to node A, link 1, has RSSG = VS. Meanwhile, node C decides that node B is not a better master for node C since DG(B) < DG(C). So both nodes B and C request to disconnect and link 2 is terminated (line 6 of ArrangeRoles procedure).

Link 3 is kept where node D chooses node E as its best master, because node D has no previous master. Likewise, link 4 is also kept where node F chooses node E as its best master. After the establishment of link 5, D requests F to disconnect from itself since DG(F) < DG(D). On the other hand, node F, after executing the BestMaster selection procedure, finds out that DG(neighbor D) + RSSG(neighbor D) = 5 + 2(S) is smaller than DG(master) + RSSG(master) = 9 + 1(M). Thus node F also requests node D for disconnection and link 5 is deleted. Link 6 is also deleted since node D selects E as a better master than G. In this case,

Table 3
Semiroot procedure

```
Semiroot (device X):
1  P-PS (X, true)
2  if any unconnected node exists
3     then if X has no master
4             then each member M of the tree of X executes P-PS (M, false)
5          exit
6     else exit
```

both the DGs of the masters E and G and the RSSGs of links 3 and 6 are equal, and thus node D keeps the previously established link. The topology formed up to this point in time is shown in Fig. 4b.

By the establishment of link 7 that has RSSG = VS, node D decides that node C is a better master than node E by the BestMaster selection procedure (due to the fact that a closer device is a better master than a far away node with higher DG). Therefore, the D–E link is deleted, and the link C–D is kept as shown in Fig. 4c. The first phase of SF-DeviL ends by the timeout discTO, up to which links 1–7 are assumed to be established in this example.

3.3.2. Merging phase

If a node has not found a master in the first phase, it declares itself as the *semiroot* (this term is used for a node that may be the actual root of the scatternet but has not proven it yet) and runs the Semiroot procedure given in Table 3. By this procedure, the semiroots that are accessible within a single hop are merged first (line 1), and the semiroots that are accessible in multiple hops are merged next (lines 2–6). The detection of being a semiroot or the actual root, and merging of the disconnected trees, in case the node is a semiroot, are done using the P–PS(X,B) procedure given in Table 4. The Semiroot procedure in the Merging phase of SF-DeviL is executed only by the semiroots. Thus, at least one device goes through this second phase.

Through exchanging messages with the tree members, the semiroot finds out if there is any discovered node appearing in one of the neighbor_lists of its tree members, but not connected to the tree. The check for unconnected discovered nodes in line 3 of P–PS procedure is done as follows:

1. The semiroot X generates a packet named ‘member_list(X)’ which contains the BD_ADDRs of all the members of the tree and floods this packet downwards to all members of the tree.
2. All members of the tree, including X, compare their neighbor_lists with member_list(X). If there is any node

Table 4
P–PS procedure

```
P-PS (device X, boolean B):
1  timer ←pageTO
2  while timer > 0
3     Check if any unconnected but heard node exists
4     if any unconnected node exists
5        then alternate between page and page scan states
6             if connection established with any device Y
7                then if (B == true)
8                        then ArrangeRoles(X,Y)
9                        else if semiroots of X and Y approve,
10                            then ReverseLinks(X,Y)
11                       timer ← pageTO
12       else exit
```

in any of the neighbor_lists which does not exist in the member_list(X), this node is reported back to X.
3. Each member has to report back to the semiroot, even if it does not detect such a node, in which case it will send a NULL report.
4. If the number of NULL reports reaching X is equal to the number of the tree members, i.e. no such node exists, X declares itself as the root, and SF-DeviL terminates.
5. At least one non-NULL report implies that the scatternet is not formed yet, and X cannot declare itself as the root, and it then starts to apply procedures described below in order to get connected to these nodes.

At the conclusion of the above procedure, X either learns that it is the root and SF-DeviL terminates, or X gets the list of all unconnected discovered devices so that it can initiate paging procedures to connect to these disconnected nodes/trees.

If any unconnected node exists, X alternates between page and page scan modes, paging all the unconnected nodes with $DG \geq DG(X)$ (line 5 of P–PS procedure). If no unconnected node with $DG \geq DG(X)$ exists, X only does page scanning. This is done to ensure that other semiroots are paged rather than their descendants (since tree descendants have smaller DGs).

Each time node X connects to any of the other semiroots, it executes the ArrangeRoles procedure. At the conclusion of ArrangeRoles, X decides whether it will keep the link, and if so, which node will become the master. After each merging of trees, X goes to lines 2 and 3 of P–PS procedure in order to check whether there are other unconnected nodes, since the new link may have brought new unconnected discovered devices.

When node X is not in the communication range of the sought devices or the once discovered devices are gone, X may not be able to establish connection with any of the paged devices and may not be paged for a specific paging timeout (pageTO). In this case, if X has a master (X may have a master after merging trees), it gives up the Semiroot procedure and reports the unconnected devices to the semiroot of its tree and exits from the Merging phase. If X has no master, it orders all the tree members to execute the P–PS(., false) procedure, by which all the tree members page the unconnected discovered devices (line 4 of Semiroot procedure). This is done in order to provide multihop connectivity among disconnected trees whose semiroots are not in communication range of each other. All members of the tree page the unconnected discovered devices (also execute PS alternately) for a period of pageTO. If an unconnected device Y is found by the tree member M, a connection is established through that member M (lines 9–10 of P–PS procedure). If the link M–Y is the first link to be established merging the two trees, both X and the semiroot of Y, need to approve the establishment of this link and the ReverseLinks procedure given in Table 5 is executed. The slave of link the M–Y determined by the BestMaster selection procedure, requests its semiroot for master/slave switching at all intermediate nodes from its semiroot down to itself.

On the other hand, if X and the semiroot of Y are already connected via other tree members, a comparison of paths connecting these two semiroots is done and the establishment of the M–Y is approved if link M–Y belongs to the 'better' path. The 'better' path is defined as the one with the smaller number of intermediate nodes (in case of equal number of intermediate nodes, the path with a larger average DG, which is computed by averaging the DGs of all nodes on the path). If the establishment of link M–Y is approved, the previous path is cut by deleting the link for which the node with the smallest DG is a slave. In the case that there is a tie, the link with the highest path loss is deleted among such links. This way, two disconnected trees are connected via the 'better' path, enhancing energy-efficiency where the semiroots are not within communication range of each other.

Each connection to an unconnected discovered device provides merging of disconnected trees and SF-DeviL terminates when no more unconnected device exist. If none of the unconnected discovered devices is found while execution of line 5 of P–PS(., false) procedure up to the timeout pageTO, X declares itself as the root, and SF-DeviL scatternet formation terminates.

The Merging phase of SF-DeviL is illustrated by the example in Fig. 4. At the conclusion of the Neighbor discovery phase, nodes B, D and F have chosen a master as shown in Fig. 4c and they exit the MAIN procedure of

Table 5
ReverseLinks procedure

```
ReverseLinks (device X, device Y):
1  if (Y==BestMaster(X, Y))
2    then Y becomes the master of X
3         do master/slave switch at nodes from X to semiroot of X
4     else X becomes the master of Y
5         do master/slave switch at nodes from Y to semiroot of Y
```

SF-DeviL. Nodes A, C, E and G that have no masters execute the Semiroot procedure. Through line 1 of this procedure, A, C, E and G find out that there are unconnected discovered neighbors as shown within boxes in Fig. 4c. Thus A, C, E and G find out that they are not the actual roots, just semiroots. Node A, finds out that C.7.W, which is an entry in neighbor_list(B), is an unconnected discovered node. Thus, node A alternates between page and page scan modes in which it pages C since $DG(C) \geq DG(A)$. Likewise, node C discovers that B, E, F and G are unconnected discovered nodes and pages nodes E and G that have $DG \geq DG(C)$. This is done to ensure that node C pages E, which is the root of the tree instead of F, which is a member of a tree. Node G finds out that D is an unconnected discovered node but does only page scanning since $DG(D) < DG(G)$. Similarly, node E finds out that D is an unconnected discovered node, and it only executes page scanning. The semiroots execute the P–PS(., true) procedure in line 1 of the Semiroot procedure in order to merge the disconnected trees.

While C is paging E and E is in page scan mode, assume that link 8 is established as shown in Fig. 4d. By the ArrangeRoles procedure, this link is kept, and E becomes the master of C since $DG(E) > DG(C)$. The establishment of link 8, i.e. merging of two disconnected trees, results in new unconnected discovered nodes for node E (line 3 of P–PS procedure). Node E learns that G is an unconnected discovered node, and node E alternately page scans and pages node G since $DG(G) \geq DG(E)$. Assuming that link 9 is established next, node E becomes the master of G since the device with larger number of slaves, in case of equal DGs, becomes the master. Link 10 which is established next is deleted since C has a current master, E, which is a better master than node G (the DGs and RSSGs of E and G are the same, thus the link to the current master is kept). The establishment of link 10 merges previously unconnected node G with one of the disconnected trees, resulting in a disconnected tree with semiroot E. Assuming that the nodes A and C are out of the communication range of each other, node A that pages node C cannot find C through the P–PS(., true) procedure.

After execution of P–PS(., true) for a duration of pageTO, node A finds out that C is still the unconnected discovered node. Nodes C, E and G discover that B is the only unconnected discovered node (line 2 of the Semiroot procedure). Nodes C and G that have found a master exit from the Semiroot procedure, while A and E execute line 4 of the Semiroot procedure. Fig. 4e illustrates how the disconnected trees of the out of range semiroots A and E are merged. The semiroot A and its descendant B page C, whereas E and the descendants of E page B while entering the page/page scan (P/PS) mode alternately. Assume that link 11 in Fig. 4e is established first. Since this link merges two disconnected trees, both semiroots A and E approve link 11. The better master among A and E, which is E, becomes the root of the merged tree, and master/slave roles of link 1 are reversed by the ReverseLinks procedure. Assume that link 12 is established next. The semiroots A and E compare the paths formed by links 11 and 12 from one semiroot to the other, i.e. from A to E, and find out that the path formed by link 12 is better since there are fewer number of intermediate nodes. Thus, A and E approve link 12, and link 11 is deleted. P–PS(., false) procedure continues until pageTO is reached, and the resulting scatternet topology formed by SF-DeviL is shown in Fig. 4f.

3.4. Deletion of the worst slave

Each master is allowed to have a maximum of seven active slaves. These slaves are the *first discovered* neighbors that selected a particular node as their master. It is possible that a master with seven slaves discovers an eighth neighbor that can only be connected to the scatternet through that specific master. In a different scenario, the link with the 8th neighbor may have a higher RSSG than existing links with other slaves. The procedure used by SF-DeviL in handling these situations is described below.

In both Neighbor discovery and Merging phases, if after a connection is established, the number slaves of a node, e.g. X, increases to seven, X deletes its 'worst' slave. The deletion of the worst slave is done as follows:

1. First, X determines its 'worst' slave by the WorstSlave selection procedure, which is the inverse of the Best-Master selection. Among the slaves of X, the one with the lowest DG+RSSG sum is selected as the worst slave. In case of a tie, the slave with lower RSSG, i.e. more path loss, is selected as the worst slave.
2. Secondly, X requests the worst slave, S_1, to check its neighbor_list for other masters with $DG \geq DG(S_1)$. If there is at least one such master, S_1 accepts deletion of $X–S_1$ link and S_1 initiates paging procedures to connect to its second (or third, fourth, etc.) best master. If there is no entry in neighbor_list(S_1) with $DG \geq DG(S_1)$, S_1 rejects deletion of $X–S_1$ link. In this case, X tries to delete its 'second worst' slave, third, fourth, and so on.

The deletion of the worst slave enhances connectivity and energy-efficiency of SF-DeviL, by replacing better links with previously established links.

3.5. Connectivity of the scatternet topology

The connectivity of the scatternet depends on the timeout values: discTO, which is used to terminate the Neighbor discovery phase, and pageTO, which is used to terminate the Merging phase. The greater the values of discTO and pageTO, the probability for a device to discover more neighbors is higher, and the scatternet formation delay is longer. We assume that these timeouts are sufficiently large

so that adequate number of neighbors necessary to form a connected topology can be discovered or paged.

We first prove that any single node that has discovered at least one node of a connected node set will merge with that set. Then, we prove that two disconnected sets of nodes will be merged if there exists a pair of nodes, one from each set, which can hear each other.

Proposition 1. Given a node w and a set of nodes S, where nodes in S are already connected, w can be connected with the nodes in S if and only if $\exists v \in S$ such that $w \in$ neighbor_list(v).

Proof. If node w is not in any neighbor_list of nodes in set S, then none of the members of S are in neighbor_list(w). Since w has no member of S in neighbor_list(w), it cannot select itself a master that belongs to S. Likewise, none of the nodes in S can select w as a master. This results in w being disconnected from S.

Suppose now that w has discovered at least one member of S, node v, at some time. In this case, three situations may occur:

Case (*1*). If w selects v as its best master, after executing the ArrangeRoles(w,v) procedure, w keeps the link to v through which w becomes connected to S.

Case (*2*). If v selects w as its best master, and if v is the semiroot of S, then w becomes connected to S as the semiroot of S. But if v is not the semiroot of S, then after executing the ArrangeRoles(w,v) procedure v keeps the link to w and deletes the link to its current master, which results in nodes w and v becoming disconnected from S. This causes the formation of two disconnected trees: S and S_{vw} (S_{vw} contains nodes v and w only). In the Merging phase, the semiroot of S will discover v to be an unconnected discovered node, whereas the semiroot of S_{vw}, which is w, will discover ex-master of v to be an unconnected discovered node (line 3 of P–PS procedure). If DG(ex-master of v) $\geq$ DG(w), the semiroot of S_{vw}, which is w, will page ex-master of v until pageTO is reached. Likewise, if DG(v) $\geq$ DG(semiroot of S), the semiroot of S will page v until pageTO expires (line 5 of P–PS procedure). Two situations may occur at this point: (a) if ex-master of v is a semiroot, it will be page scanning while w is paging it, and a connection between w and the ex-master of v will be established after by the P–PS(., true) procedure in line 1 of the Semiroot procedure (assuming that pageTO is large enough); (b) if the ex-master of v is not a semiroot, it will not be listening for the pages. Thus the sets S and S_{vw} remain disconnected after execution of line 1 of the Semiroot procedure. The semiroots of S and S_{vw} move to line 4 of Semiroot procedure, since v is an unconnected discovered node for the semiroot of S, and ex-master of v is an unconnected discovered node for w. In line 4 of the Semiroot procedure, the semiroot of S orders all its descendants, including ex-master of v, to alternately page v and execute page scan. Likewise, w orders v to alternately page and page scan for a period of pageTO. Consequently, either node v establishes a connection to any member of S, or ex-master of v establishes a connection to any member of S_{vw}. Master/slave switches take place at nodes from the slave of this connection upto the semiroot of the slave and node w becomes connected to S by the ReverseLinks procedure.

Case (*3*). If neither w nor v is selected as the best master for each other, w–v link is broken by the ArrangeRoles(w,v) procedure. This results in the same case as 2, explained above. □

Having proven that a single node discovered by any member of a tree is connected to that tree at the end of SF-DeviL, we will now prove that any two disconnected trees will be merged if there exists a pair of nodes that discovered each other.

Proposition 2. Given two disjoint sets of nodes S_1 and S_2, where nodes in S_1 and S_2 are already connected among themselves, S_1 and S_2 can be merged if and only if $\exists v \in S_1$ and $\exists w \in S_2$ such that $w \in$ neighbor_list(v).

Proof. The three cases explained in Proposition 1 are possible again.

Case (*1*). Node v chooses w as its master and becomes connected to S_2, resulting in two sets $S_2 \cup v$ and S_1/v.

Case (*2*). Node w chooses v as its master and becomes connected to S_1, resulting in the two disconnected sets $S_1 \cup w$ and S_2/w.

Case (*3*). None of v and w chooses the other as best master.

In all three cases, at least one node in both disconnected sets have at least one unconnected discovered node in its neighbor list. Thus, these disconnected sets are merged the same way as explained in Case 2 of the Proof of Proposition 1. □

4. Topology maintenance in SF-DeviL

The energy-efficient scatternet formation protocol SF-DeviL is extended in this section to handle topology maintenance due to decreasing battery levels and mobility.

4.1. Maintenance with depleting battery levels

Scatternet topology is reconfigured by SF-DeviL when battery levels are depleted. If BatteryLevel of a device (except for leaf nodes) reaches a threshold value, a scatternet update request is sent to the root. The root orders all members to re-calculate their DGs and collects the updated DGs from all descendants. The root sends a packet, which includes BD_ADDR and DGs of all tree members, to its descendants. Upon receiving this packet, each tree member starts paging and page scanning devices with higher or equal DGs. This way the devices with decreasing battery levels are pushed downward towards the leaf positions in the tree to increase their battery lifetimes.

4.2. Maintenance with mobility

SF-DeviL scatternets are updated also in response to node deletions and additions as explained below:

Node deletions. A master receiving no reply from one of its slaves for a specific number of times, starts P/PS by paging that node until the paging timeout (pageTO) expires. If no reply is received, the root is informed that the connection with a node is lost. The root sends a packet to all nodes and asks them to execute P/PS for finding the lost node. Nodes that receive this packet, except leaf nodes, may just forward this request without executing P/PS if their traffic load is high, i.e. if they have too many packets waiting in their transmission queues. If the lost node cannot be found within pageTO, the node is deleted from routing tables.

Any node that has lost connection from an SF-DeviL scatternet, executes P/PS where it first pages its master for a duration of pageTO. If the lost node cannot find its master, it pages any entry in its neighbor_list with DG > DG(X) and enters PS alternately. If no connection is established, the lost node starts inquiry/inquiry scanning (I/IS), returning to device discovery phase.

Node additions. The leaf nodes of a tree are the nodes that have the lowest transit traffic load. They do not switch between piconets and do not forward packets like the root and bridges. In SF-DeviL, each *leaf-master* (masters that have at least one slave as a leaf node) orders one of their slaves, the one with the largest DG, to execute device discovery. The leaf nodes that are assigned for device discovery execute I/IS to allow addition of new nodes. Consequently, the nodes that are least likely to have a battery depletion are assigned with the additional task of new node discovery, which in turn increases the energy efficiency of the scatternet operation.

SF-DeviL forms scatternets in case of node deletions occurring during the scatternet formation process. The timeout used during the Merging phase of SF-DeviL, pageTO, ensures that paging of a deleted (or failed) node is not carried on indefinitely.

Table 6
Quantization of RSSG-based nominal received powers

Nominal received power (NRP)	RSSG
$-30\ \text{dBm} \le \text{NRP}$	VS (very strong)
$-40\ \text{dBm} \le \text{NRP} < -30\ \text{dBm}$	S (strong)
$-50\ \text{dBm} \le \text{NRP} < -40\ \text{dBm}$	M (medium)
$\text{NRP} < -50\ \text{dBm}$	W (weak)

5. Simulation results

A C++-based simulator compliant with Bluetooth specifications [1] is developed in order to evaluate the performance of SF-DeviL. The lifetime, number of piconets, network diameter, average number of hops between source–destination pairs, average link length and formation delay of SF-DeviL are compared with the tree structured LMS [6] and mesh structured BlueMesh [7] scatternets. The effects of changing discTO on SF-DeviL scatternet formation and maintenance performance are also investigated. Two different networking scenarios are considered in this study: a network with identical devices (corresponding to a *homogeneous sensor network*) and a network with devices of different classes (corresponding to multiple PANs or a *heterogeneous network*).

In the simulations, nodes are randomly distributed in an area of $10\ \text{m} \times 10\ \text{m}$. Although SF-DeviL supports multi-hop operation, nodes are positioned such that all nodes can communicate with each other since this is required by the LMS algorithm. For a given number of nodes, the averages of the performance metrics obtained for five randomly generated node locations and traffic patterns are reported.

The following classes of devices are used: laptops, mobile phones, PDAs, headsets, peripherals and sensors. The devices are initially assigned with full batteries. At each node, traffic is generated randomly with a rate proportional to the TrafficGenerationGrade that is assigned to each device based on the kind of traffic it generates.

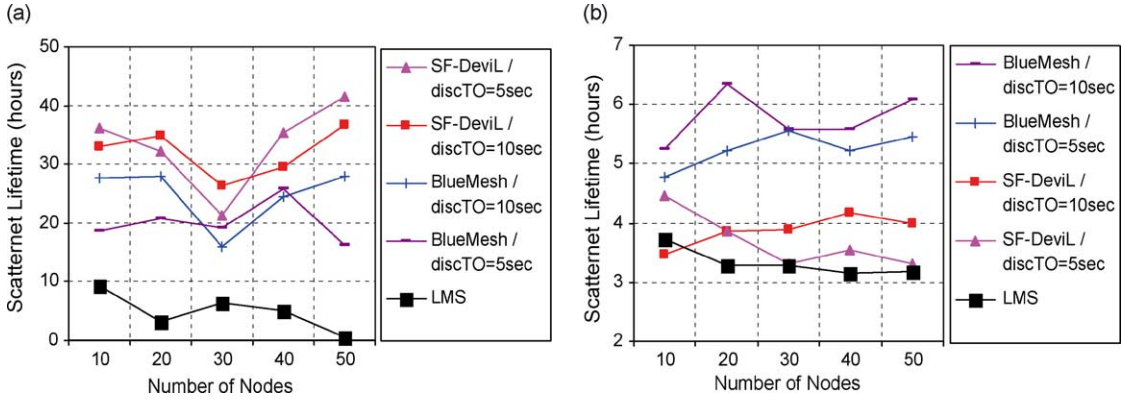

Fig. 5. Scatternet lifetime comparison of SF-DeviL with LMS and BlueMesh algorithms when devices are of: (a) different types and (b) same type.

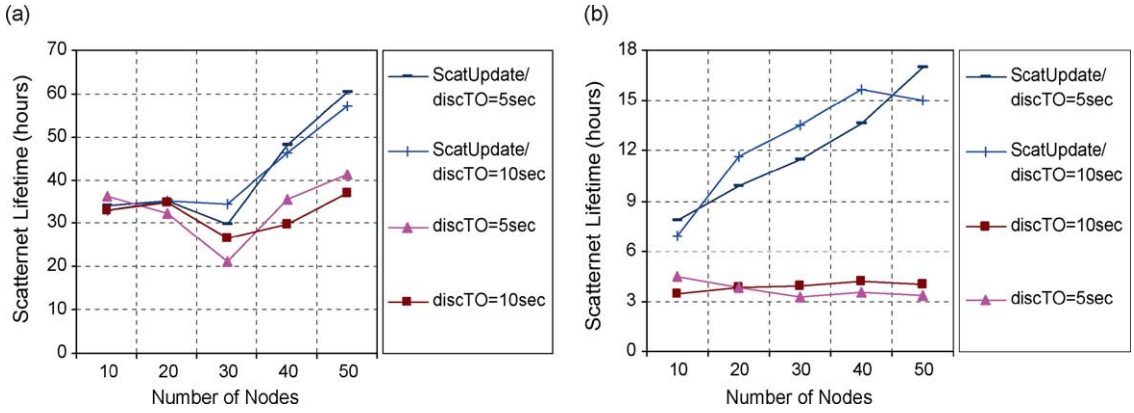

Fig. 6. Scatternet lifetime of SF-DeviL with and without topology maintenance when devices are of: (a) different types and (b) same type.

Power consumed for transmission/reception at each slot is taken as P_{transmit} for transmission and P_{receive} for reception. Power consumed in the standby mode is ignored. Based on the specifications of Bluetooth chips currently available in the market, the maximum transmit power and P_{receive} are assumed to be equal. Power control is done at each node assuming a receiver sensitivity of -60 dBm. We assume that P_{transmit} can be reduced by at most 30% by the power control. The following path loss model is used [18]:

$$\text{PL}(d) = \text{PL}(d_0) + 10\gamma \log(d/d_0) + X_\sigma$$

where PL(d) denotes the path loss, in dB, for a path of length d, PL($d_0 = 1$ m) $= 30$ dB, $\gamma = 2.5$, $X_\sigma = N(0,\sigma)$ with $\sigma = 5$ dB. RSSGs of links are quantized as given in Table 6. In this table, the nominal received power corresponds to the received power level when the maximum transmit power is used.

We assume that nodes are fixed, and the topology is reconfigured only in response to battery level depletions. Each device, other than the leaf nodes, initiates a scatternet update when its battery is halved, i.e. BatteryLevel $\leq 1/2$.

For the implementation of the BlueMesh algorithm, the BD_ADDR of devices are assigned as weights. Routing in the BlueMesh scatternets is done via the shortest paths computed by Dijkstra's algorithm.

The average scatternet lifetimes of SF-DeviL without topology maintenance are compared in Fig. 5 with the LMS and BlueMesh algorithms, as a function of the network size, for heterogeneous and homogeneous networks. Two different values of discTO are used commonly for SF-DeviL and BlueMesh. For different device types, the lifetime is increased substantially by 229–6314% with respect to the LMS algorithm, whereas the lifetime is increased by 10–154% for discTO $= 5$ s and 19–66% for discTO $= 10$ s with respect to the BlueMesh algorithm. When all devices are of the same type (homogeneous network), the lifetime is still increased by 1–24% compared to LMS. For the homogeneous network, BlueMesh increases the lifetime by up to 86% for discTO $= 5$ s and up to 81% for discTO $= 10$ s with respect to SF-DeviL. This is due to the fact that SF-DeviL forms the scatternet by making use of RSSG only, since all devices have the same DG during the initial formation of the scatternet. Since the tree topology generated by SF-DeviL typically has a larger number of bottleneck nodes compared with the mesh topology of BlueMesh, the network lifetime decreases for the homogeneous network.

Fig. 6 shows the effect of scatternet maintenance due to battery level depletions on scatternet lifetimes of SF-DeviL

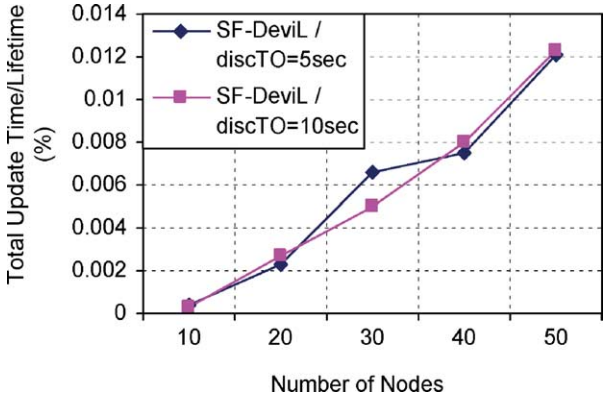

Fig. 7. Percentage of total update duration.

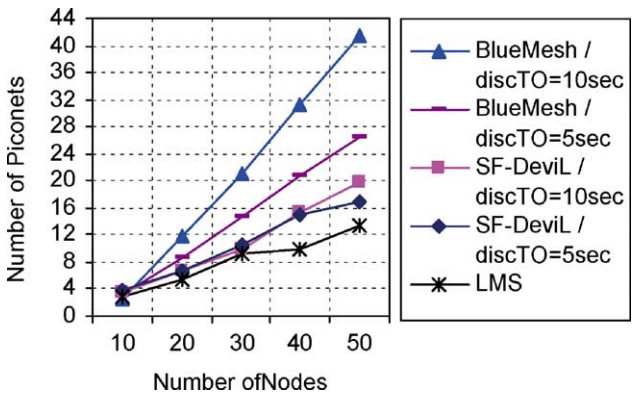

Fig. 8. Number of piconets.

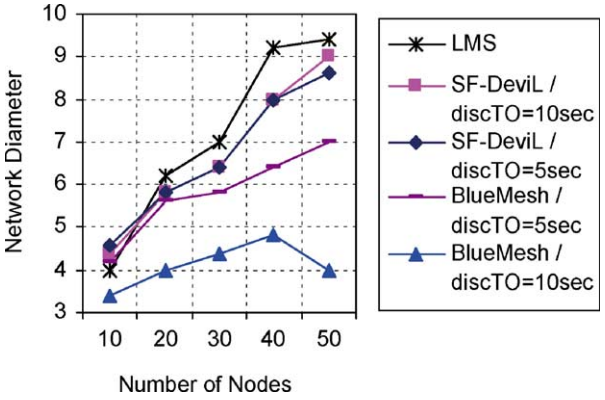

Fig. 9. Network diameter.

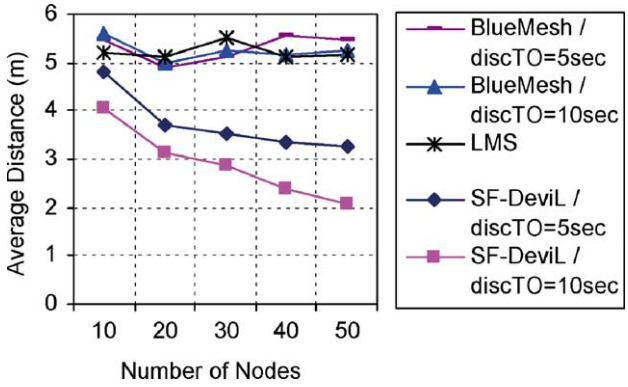

Fig. 11. Average length of links in the scatternet.

for different values of discTOs. For different device types, topology maintenance increases lifetimes of large scatternets. The effect of maintenance is more pronounced in the homogeneous network where an increase of 75–410% is observed with respect to SF-DeviL without scatternet update. When the scatternet topology is reconfigured in response to depleting battery levels, SF-DeviL achieves lifetimes that are significantly higher than the BlueMesh algorithm even for the homogeneous network.

These results show that SF-DeviL increases lifetime for both homogenous and heterogeneous networks, but more for the heterogeneous case. It is observed in the simulations that batteries of leaf nodes are the first ones to be depleted for some of the relatively small-sized heterogeneous networks, since devices with higher DGs are assigned as the root and bridge nodes. Since leaf nodes cannot trigger topology updates, the network lifetime does not improve significantly with maintenance. When the network becomes larger, the nodes in the upper layers of the tree topology are carrying more transit traffic, and these nodes become more likely to trigger topology reconfigurations. Thus scatternet updates in response to decreasing battery levels provide more significant lifetime improvements for larger sized heterogeneous networks.

Average lifetimes of SF-DeviL scatternets for different values of discTO exhibit similar behavior. Large discTO results in longer I/IS intervals and more battery dissipation during discovery, which may decrease the lifetime in some cases. On the other hand, with a small discTO, e.g. discTO <5 s, a smaller fraction of neighboring devices can be discovered and a connected scatternet topology cannot always be formed. Simulations show that with discTO $=5$ s, 50–70% of the neighbors are discovered, whereas with discTO $=10$ s, almost all neighbors can be discovered.

The total time required for topology updates as a percentage of the scatternet lifetime is shown in Fig. 7, which is at most 0.012% of the scatternet lifetime. The maximum rate of topology updates occurs for the 50-node network, which is 2.9 updates/hour.

SF-DeviL, unlike LMS, does not have the explicit goal of forming scatternets with small number of piconets. As shown in Fig. 8, the number of piconets with LMS is smaller than SF-DeviL, and there is not a significant difference in the number of piconets when different values of discTO are used with SF-DeviL. SF-DeviL forms scatternets with smaller number of piconets compared with the BlueMesh algorithm.

The network diameter, defined as the maximum number of hops between two nodes, is slightly lower for SF-DeviL

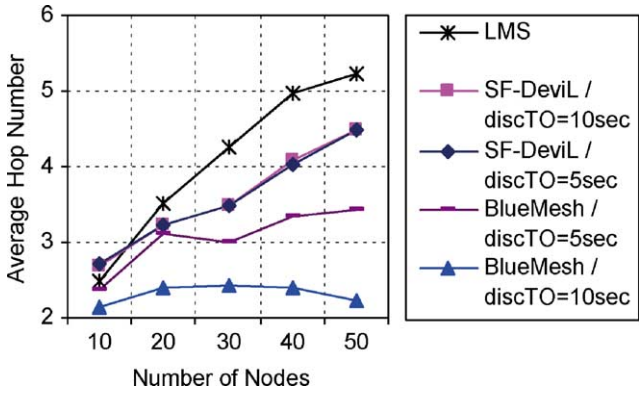

Fig. 10. Average number of hops.

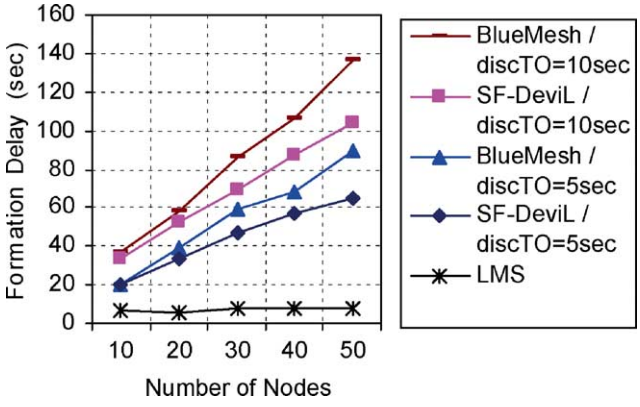

Fig. 12. Scatternet formation delay.

compared to the LMS algorithm, and larger compared with the BlueMesh algorithm, as shown in Fig. 9. The average number of hops between source–destination pairs is shown in Fig. 10. The average number of hops in SF-DeviL is smaller than LMS and larger than BlueMesh. The network diameter and the average number of hops for BlueMesh are smaller than SF-DeviL since the mesh topology of BlueMesh provides larger nodal degrees and a denser connectivity.

SF-DeviL forces each node to connect to the closest node with the highest DG. For this reason, the average distance of links is smaller compared to LMS and BlueMesh as shown in Fig. 11. As the number of nodes increases, the nodal density increases resulting in shorter links. SF-DeviL scatternets formed by discTO = 5 s contain longer links compared to the case of discTO = 10 s since only a subset of neighbors is discovered by each node. Shorter links result in less transmit powers and less interference for other wireless technologies such as IEEE 802.11b which also uses the ISM band.

In Fig. 12, the formation delay for SF-DeviL is shown as a function of the network size for different values of discTO. We observe that the formation of the scatternet takes longer than LMS and slightly shorter than BlueMesh. The connection delay for SF-DeviL increases with the network size due to the increase of number of discovered neighbors. Increasing discTO increases connection delay, thus there is a trade-off between discovering more neighbors and formation delay. Simulations show that discTO = 5 s provides a good compromise between these two trends.

6. Conclusions

Energy-efficiency in scatternet formation and maintenance is an important issue in developing services using the Bluetooth technology. SF-DeviL is a tree-based Bluetooth scatternet formation and maintenance algorithm targeting low-power consumption in multi-hop wireless networks. Power control capability, received signal strength indication and availability of device class information are used for energy-efficient communications. SF-DeviL reconfigures the topology in response to depleting battery levels. SF-DeviL produces scatternets where a node participates in just two piconets so that bridge nodes do not become bottlenecks between multiple piconets.

Simulations show that using class and link characteristics during scatternet formation and performing topology maintenance in response to changing battery levels, network lifetimes can be substantially prolonged with respect to existing algorithms. The total time durations spent during topology reconfiguration is only a small fraction of the network lifetime. SF-DeviL is a viable solution for building energy-efficient scatternets especially in heterogeneous environments. In homogeneous networks, using mesh topologies instead of trees can enhance energy-effieciency even further. Moreover, SF-DeviL also forms topologies with number of piconets close to the minimum within reasonable formation delays.

References

[1] Bluetooth SIG, Specification of the Bluetooth System, Version 1.1, http://www.bluetooth.com.
[2] A. Rodzevski, J. Forsberg, I. Kruzela, Wireless sensor network with Bluetooth, Smart Objects Conference (SOC'2003), Grenoble, France, May 2003.
[3] S. Krco, Bluetooth based wireless sensor networks—implementation issues and solutions, TELFOR'2002, Belgrade, Yugoslavia, November 2002.
[4] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, L. Thiele, Prototyping wireless sensor network applications with BTnodes, First European Workshop on Wireless Sensor Networks (EWSN 2004), Berlin, Germany, January 2004.
[5] M. Leopold, M.B. Dydensborg, P. Bonnet, Bluetooth and sensor networks: a reality check, First ACM Conference on Sensor Networks (SenSys'2003), November 2003.
[6] C. Law, A.K. Mehta, K.-Y. Siu, A new Bluetooth scatternet formation protocol, Mobile Networks and Applications 8 (2003) 485–498.
[7] C. Petrioli, S. Basagni, I. Chlamtac, BlueMesh: degree-constraint multi-hop scatternet formation for Bluetooth networks, Mobile Networks and Applications 9 (2004) 33–47.
[8] T. Salonidis, P. Bhagwat, L. Tassiulas, R. LaMaire, Distributed topology construction of Bluetooth personal area networks, Proceedings of the IEEE Infocom, April 2001; 1577–1586.
[9] G. Tan, A. Miu, J. Guttag, H. Balakrishnan, An efficient scatternet formation algorithm for dynamic environments, IASTED Communications and Computer Networks (CCN), Cambridge, November 2002.
[10] C. Pamuk, E. Karasan, SF-DeviL: distributed Bluetooth scatternet formation algorithm based on device and link characteristics, 8th IEEE Symposium on Computers and Communications (ISCC'2003), Antalya, Turkey, June 2003.
[11] G.V. Zaruba, S. Basagni, I. Chlamtac, Bluetrees-scatternet formation to enable Bluetooth based ad hoc networks, ICC 2001, Helsinki, June 2001; 273–277.
[12] I. Stojmenovic, Dominating set based scatternet formation with localized maintenance, International Parallel and Distributed Processing Symposium, Ft. Lauderdale, April 2002.
[13] R. Kapoor, M.Y.M. Sanadidi, M. Gerla, An analysis of Bluetooth scatternet topologies, Proceedings of ICC 2003, Anchorage, May 2003.
[14] F. Cuomo, T. Melodia, Ad hoc networking with Bluetooth: key metrics and distributed protocols for scatternet formation. Journal of Ad Hoc Networks, to appear.
[15] Gy. Miklos, A. Racz, Z. Turanyi, A. Valko, P. Johansson, Performance aspects of Bluetooth scatternet formation, Proceedings of the First Annual Workshop on Mobile Ad hoc Networking and Computing (MobiHOC) 2000; 147–148.
[16] S. Basagni, R. Bruno, G. Mambrini, C. Petrioli, Comparative performance evaluation of scatternet formation protocols for networks of Bluetooth devices, Wireless Networks 10 (2) (2004) 197–213.
[17] B.J. Prabhu, A. Chockalingam, A routing protocol and energy efficient techniques in Bluetooth scatternets, ICC 2002, New York 2002; 3336–3340.
[18] T.S. Rappaport, Wireless Communications—Principles and Practice, 2nd ed, Prentice Hall, New Jersey, 2002.

Canan Pamuk received her BS degrees with honors in electrical and electronics engineering and physics in 2001 from Boğaziçi University, Istanbul, Turkey, and MS degree in electrical and electronics engineering in 2003 from Bilkent University, Ankara, Turkey. She is currently a PhD student and a research assistant in the Electrical and Electronics Department at Bilkent University. Her current work focuses on Bluetooth and energy-conserving protocols in ad hoc and sensor networks. She is a member of IEEE.

Ezhan Karasan received the BS degree from Middle East Technical University, Ankara, Turkey, the MS degree from Bilkent University, Ankara, Turkey, and the PhD degree from Rutgers University, Piscataway, New Jersey, USA, all in electrical engineering, in 1987, 1990, and 1995, respectively.During 1995–1996, he was a post-doctorate researcher at Bell Labs, Holmdel, New Jersey. From 1996 to 1998, he was a Senior Technical Staff Member in the Lightwave Networks Research Department at AT&T Labs-Research, Red Bank, New Jersey. Since 1998, he has been an assistant professor in the Electrical and Electronics Engineering Department at Bilkent University, Ankara, Turkey.His current research interests are in the application of optimization and performance analysis tools for the design and analysis of optical and wireless ad hoc/sensor networks.