Noname manuscript No.
(will be inserted by the editor)

Bluetooth Scatternet Formation from a Time-Efficiency Perspective

Ahmed Jedda · Arnaud Casteigts · Guy-Vincent Jourdan · Hussein T. Mouftah

Received: date / Accepted: date

Abstract The Bluetooth Scatternet Formation (BSF) problem consists of interconnecting piconets in order to form a multi-hop topology. While a large number of BSF algorithms have been proposed, only few address time as a key parameter, and when doing so, virtually none of the solutions were tested under realistic settings. In particular, the baseband and link layers of Bluetooth are highly specific and known to have crucial impacts on performance. In this paper, we revisit performance studies for a number of time-efficient BSF algorithms, focusing on BlueStars, BlueMesh, and BlueMIS. We also introduce a novel time-efficient BSF algorithm called *BSF-UED* (for BSF based on Unnecessary-Edges Deletion), which forms connected scatternets deterministically and limits the outdegree of nodes to 7 heuristically. The performance of the algorithm is evaluated through detailed simulation experiments that take into account the low-level specificities of Bluetooth. We show that BSF-UED compares favorably against BlueMesh while requiring only 1/3 of its execution time. Only BlueStars is faster than BSF-UED, but at the cost of a very large number of slaves per master (much more than 7), which makes it impractical in many scenarios.

Keywords Bluetooth, Scatternet Formation, Distributed algorithms, Topology Control.

A. Jedda, G-V. Jourdan, H. T. Mouftah
School of Electrical Engineering and Computer Science
University of Ottawa
E-mail: ajedd@uottawa.ca, gvj@uottawa.ca, mouftah@uottawa.ca

A. Casteigts
LaBRI
University of Bordeaux
E-mail: arnaud.casteigts@labri.fr

1 Introduction

In this paper, we study the problem of forming multi-hop ad hoc networks with Bluetooth, one of the most widespread communication technologies. Point-to-point communication between two Bluetooth devices is relatively straightforward, but the construction of efficient *multi-hop* topologies is challenging. This problem attracted much attention a decade ago, less recently as the question of whether Bluetooth is more than a point-to-point cable replacement technology started to be debated. With the current development of home automation and personal area networks, which involves devices as varied as heart-rate monitors, smart-phones, blood-glucose meters, smart watches, window and door security sensors, car key fobs, or blood-pressure cuffs, there is a revival of interest for Bluetooth in an ad hoc networking context. This revival is also due to the wide adoption of Bluetooth (about 95 % of today's mobile phones are enabled [1]) and the fact that it offers communication at low cost, low energy consumption, and low interference. While the Bluetooth specifications kept evolving (it is now in its seventh version), new marketing trademarks were recently pushed forward by the Bluetooth Special Interest Group (SIG) in anticipation of a new networking trend around Bluetooth, including *Bluetooth Smart* and *Bluetooth Smart Ready*[1].

The Bluetooth Scatternet Formation (BSF) problem: According to the specifications [2], a Bluetooth device can be either *master* or *slave* when it communicates. A master along with its slaves is called a *piconet*. All communications in a piconet is controlled by the master. The number of *active* slaves (that is, slaves that can participate in the piconet's communication) is limited to 7. More slaves are possible in a piconet if some of them are inactive (or *parked*). In such

[1] Bluetooth Smart Ready: http://www.bluetooth.com/pages/Bluetooth-Smart-Devices.aspx. Fetched on Dec. 20, 2012

cases, the master dynamically parks and unparks its slaves to regulate communications, which imposes a penalty on the piconet throughput. Thus, maintaining the number of slaves per piconet below 7 is highly desirable. A piconet that has at most 7 slaves is called *outdegree limited*[2].

The interconnection of several piconets is called a *scatternet*. Scatternets are the solution for multi-hop communication in Bluetooth. A scatternet interconnects several piconets by having some nodes playing a dual role in two piconets, namely, master in one piconet and slave in the other (*M/S* bridge) or slave in both (*S/S* bridge). *M/M* bridges are not allowed (that is, a node cannot be master to more than one piconet at the same time). A bridge node must schedule its time between the different piconets it belongs to (using a so-called interpiconet scheduling algorithms). As a result, a large number of bridges in a scatternet also imposes a penalty on the throughput of the scatternet. Among those, M/S bridges impose a higher penalty than S/S bridges because an M/S bridge causes all its slaves to be inactive when it is itself inactive. It is also preferable to keep the number of piconets a node belongs to – its number of *roles* – to a minimum. A classical metric in this regard is the *average* number of roles per node. A scatternet such that all its piconets are outdegree limited is called an *outdegree limited scatternet*. Limiting the outdegree of a scatternet, which is a main objective in many BSF algorithms, improves the performance of the scatternet significantly.

An algorithm that forms scatternets is called a Bluetooth Scatternet Formation (BSF) algorithm. The way piconets must interconnect to form scatternets is not specified in Bluetooth specifications and left open to the research community. The many possible approaches as well as the number of quality metrics to be assessed on the resulting topology make this problem challenging. One difficulty is that some of the metrics are conflicting (*i.e.,* improving one may deteriorate another). We are interested in BSF algorithms that balance between all metrics under reasonable assumptions, while keeping a light emphasis on the execution time for suitability to changing environments.

It is a challenging task for BSF algorithms to verify such a dosage of performance metrics. BlueStars [3] forms in a time-efficient manner connected scatternets that have a low number of piconets and M/S bridges, but potentially a large number of slaves per piconet. Many attempts have been made to solve the issue of large piconets in BlueStars. For instance, Li et al. [4] introduce an algorithm that generates outdegree limited scatternets using geometric structures, but they assume that knowledge of nodes positions is available. Bluenet [5] forms outdegree limited scatternets with an acceptable number of piconets and M/S bridges, but does not guarantee connectivity of the resulting scatternet. BlueMesh [6] offers similar qualities together with connectivity, but at the cost of a long execution time. Another example is BlueMIS I [7] which generates in a time-efficient manner scatternets that are connected and outdegree-limited but with significantly high number of piconets. The authors of [7] introduce simple rules to be executed over BlueMIS I that were called BlueMIS II in order to improve the qualities of BlueMIS I. However, BlueMIS II suffers from either a long execution time or outdegree unlimited scatternets depending on how these rules are implemented.

Unfortunately, the BSF algorithms which are presented as time-efficient were not evaluated under the complex baseband and link layers of Bluetooth, despite the high specificity of these layers and their impact on performance. The corresponding papers either do not mention what simulator was used, or present simulations that relied on naive simulators such as `simjava` [8] or `bluehoc`[3], which makes it hard to assess the real efficiency of these solutions.

This paper extends a line of work [9, 10] whose focus is the execution time of Bluetooth networks algorithms. In [10], we showed how simple changes in the configuration of Bluetooth devices may either significantly improve or degrade the execution time of a distributed algorithm running over a Bluetooth network. The work in [10] focused on time efficient implementations of communication rounds in Bluetooth networks (that is, we studied the problem of how every node sends and receives a message to and from all its neighbors as fast as possible). The study in [10] introduced us to interesting properties of Bluetooth networks. These properties affect the execution time of distributed algorithms run over Bluetooth networks. These properties must be considered in the design of BSF algorithms in order to improve their execution times. In [9], we used some of the results obtained in [10] to design time-efficient outdegree limited BSF algorithms. The algorithms of [9] do not perform well in term of some performance metrics.

The first main contribution of this paper is a comparative study of major BSF algorithms. The major criteria that we focus on in the studied algorithms are:

1. The execution time of the algorithms, from an empirical and theoretical point of view.
2. The balance between performance metrics: a BSF algorithm shall form scatternets that are efficient with respect to performance metrics other than execution time (e.g., scatternet connectivity, outdegree limitation, number of bridges, average role per node and fault tolerance). and,
3. Other criteria: a BSF algorithm shall not run only in single-hop networks, or depends on knowledge of nodes

[2] In the following, a piconet is modeled as a star graph with a master and slaves. We model a master-slave relationship as a directed edge from the master to slave, whence the number of slaves of a master is its outdegree.

[3] Bluehoc: http://bluehoc.sourceforge.net/. Fetched on Dec. 20, 2012

positions or relative distance to other nodes, or assume the existence of a centralized entity, such a server or a fixed distinct node.

The algorithms that match our criteria are BlueStars [3], BlueMesh [6], BlueMIS I and BlueMIS II [7]. These algorithms form mesh-like scatternets which give more fault-tolerance to the scatternet compared to tree-like scatternets. These algorithms have distinctive theoretical and practical features, as it is shown in the rest of the paper. We believe that these algorithms are the best of their kind. This view is shared with the authors of [11].

The second main contribution of this paper is the introduction of a distributed algorithm that we call Bluetooth Scatternet Formation based on Unnecessary Edges Deletion (BSF-UED). Algorithm BSF-UED uses concepts of BlueMesh, BlueStars and BlueMIS in order to achieve a balance between the advantages of these algorithms. We give special attention to forming outdegree limited scatternets in a time efficient manner without significantly affecting the other quality metrics of the scatternet. We focus mainly on three performance metrics of scatternets, 1) connectivity, 2) execution time, and 3) outdegree limitation. Focusing on a small set of requirements without significantly affecting other quality metrics leads to a better understanding of the problem which, as a result, leads towards the design of more efficient BSF algorithms in the Future. Algorithm BSF-UED is time-efficient, and deterministically forms connected scatternets in multi-hop networks and heursitically forms outdegree limited to 7. It forms scatternets with a low average role per node, low average piconet size using low number of messages. Furthermore, the algorithm does not require any knowledge of position or relative distance between nodes. The idea of the algorithm is to delete edges that are unnecessary for the connectivity of our scatternets using simple local rules. BSF-UED may form scatternets that are not outdegree limited. However, simulation experiments show that using BSF-UED with a heuristic, which we call H1, generates scatternets that are outdegree limited in virtually all the cases (e.g. only one outdegree non-limited scatternet was formed over 5000 runs we performed).

The performance of BSF-UED is studied using simulation experiments. It is compared to that of BlueStars [3], BlueMesh [6] and BlueMIS [7]. We show that the execution time of BSF-UED is approximately 1/3 the execution time of BlueMesh. BSF-UED is shown to form scatternets with similar properties to those of BlueMesh. We show that BSF-UED is better than the algorithms in hand with respect to many performance metrics. We include also a theoretical analysis of the algorithms in hand in Appendix A.

The paper is organized as follows; Section 2 gives some basics of the Bluetooth technology. Section 3 gives a formal definition of the Bluetooth Scatternet Formation problem. Section 4 gives a brief literature review. Section 5 describes in details the algorithm BSF-UED. Section 6 discusses the simulation results, and section 7 concludes the paper.

2 Bluetooth Basics

Bluetooth technology is a wireless technology that uses the ISM band from 2400-2480 MHz divided into 79 channels (1 MHz each). Bluetooth devices use the Frequency Hopping Spread Spectrum (FHSS) technique for communication. A pair of nodes communicating with each other alternates between a set of pseudo-random frequency channels known to both nodes. During this alternation, the nodes exchange their messages. Two basic procedures are of interest to us; the device discovery and link establishment procedures. For a node to discover a neighbor, it must switch to a state called INQUIRY. It broadcasts small packets called ID in different channels to announce its existence. A node that wants to be discovered switches to a state called INQUIRY SCAN. The scanner alternates pseudo-randomly between a set of channels. If a scanner receives one of the packets of an inquirer, it sends back a packet to the same inquirer. The two devices exchange some packets then. The procedure of discovery is terminated thereafter.

Bluetooth is a connection-oriented communication standard (that is, any two communication nodes must build a link before communicating). For a node u to establish a link with a neighbor v, node u switches to a state called PAGE while node v switches to a state called PAGE SCAN. Node u sends packets specifically designated to v in different channels. Node v on the other hand alternate between a set of frequency channels in the PAGE SCAN state and in case it received a packet from u, then both nodes exchange some packets in order to terminate the procedure. Such a link represents a piconet of a master (node u in this example) and one slave (node v in this example).

According to the Bluetooth specifications, for any pair of Bluetooth devices to communicate with each other, they both need to be in the same scatternet or the same piconet. Given the unavailability of scatternet before a BSF algorithm is executed, most BSF algorithms uses the following technique to exchange messages in order to build the scatternets; if a node needs to send a message to a neighbor it builds a temporary piconet with it, exchanges messages and then destroys the piconet. A standard forwarding technique is used for a node to send a message to a non-neighbor node.

3 Problem Statement

The scatternet formation problem can be defined as follows. The input is an undirected graph $G = (V, E)$, where V is the set of Bluetooth nodes and E is the set of edges between the nodes of V such that an edge $(u, v) \in E$ if u and v are within

the radio range of each other, and both nodes have discovered each other during the discovery phase. The objective is to form a scatternet $\mathcal{S} = (V, E')$ such that $\mathcal{S}$ is a directed subgraph of G with V as set of nodes and E' as set of edges, whereas if an edge (u, v) is in E', then $(u, v) \in E$ and $(v, u) \notin E'$. The set E' essentially represents the master-slave relationships between neighbor nodes. We denote n as the size of V (i.e. the number of nodes).

The set of neighbors to any node u is denoted $N(u)$. The degree of a node u is the size of $N(u)$ (i.e. $|N(u)|$). The set of all masters of u is denoted $M(u)$. The set of all slaves to u is denoted $S(u)$. The outdegree of a node u is the size of the set $S(u)$. It is preferred that the outdegree of all nodes in the scatternet to be at most 7. We call such scatternets as outdegree limited scatternets.

A piconet u is the set $\rho(u)$ such that $\rho(u) = \{u \cup S(u)\}$. The master of piconet $\rho(u)$ is u. A scatternet $\mathcal{S} = (V, E')$ is a set of interconnected piconets. The set of all piconets $\rho(u)$ in a scatternet $\mathcal{S} = (V, E')$ is denoted $\mathcal{P}_{\mathcal{S}}$. Note that it is not necessary that every node u in the scatternet $\mathcal{S} = (V, E')$ is a master, and whence it is not necessary that there is a piconet $\rho(u) \in \mathcal{P}_{\mathcal{S}}$ for every node $u \in V$. Throughout this paper, we use the notation $G = (V, E)$ for the input graph, $\mathcal{S} = (V, E')$ for the output scatternet, E for the set of edges of the input graph and E' for the set of edges of the output scatternet.

3.1 Performance metrics

The standard performance metrics used to measure the efficiency of BSF algorithms are many. The most important metrics:

1. *Execution time*: The BSF algorithm shall form a scatternet in a short time.
2. *Connectivity:* The subgraph consisting of the scatternet must be connected.
3. *Outdegree limitation:* Each piconet shall not have more than seven slaves. Otherwise they have to be parked and unparked, which imposes a penalty on the performance.
4. *Number of piconets:* The number of piconets should be minimized. A large number of piconets causes a scatternets to consume more energy because a larger number of nodes (that is, the masters) have to control the flows of packets, which consumes substantial energy.
5. *Number of M/S and S/S bridges*: M/S bridges have a higher penalty on the performance of scatternets. However, both should be minimized.
6. *Average role per node*: The number of roles per node is the number of piconets it belongs to. The average role per node should be minimized.

These performance metrics are discussed in [11] and [12] and are used to measure the performance of most BSF algorithms in the literature (see for example, [3][4][5][6][13][14] and [15]).

Algorithm BSF-UED focuses on the execution time, connectivity, and outdegree limitation quality metrics. Focusing on these three quality metrics helps in better understanding of the BSF problem, which as a result leads to the design of more efficient BSF algorithms on the future. It should be noted that the BSF problem remains challenging with only these three performance metrics as shown in Section 4. However, BSF-UED gives an acceptable balance between the other performance metrics.

4 Related Work

We present in this section a brief literature survey for the BSF problem. A more detailed survey can be found in [11]. We categorize existing BSF algorithms into four categories. We discuss each of these categories in the following.

4.1 Centralized BSF algorithms

These BSF algorithms assume that there is a centralized node that has full knowledge of the network topology. The centralized node can be a specific centralized server or an elected leader node. The centralized node collects information about all the nodes in the network and performs a BSF algorithm that assigns to each node a role and to which piconet each node belongs to. For example, algorithm BTCP (Bluetooth Topology Construction Protocol) [16] elects a leader node that collects all nodes information and executes a BSF algorithm. BTCP assumes that every node is in the radio range of all other nodes and that the number of nodes in the network is at most 36. Given the the availability of a centralized node, more sophisticated approaches can be executed such as evolutionary algorithms [17],[18], linear programming optimization [19], or others [20][21]. These algorithms attempts to optimize a larger number of performance metrics simultaneously and often outperforms decentralized algorithms. The main weakness of this category of BSF algorithms is being not decentralized and thus non-scalable. Also, centralized BSF algorithms do not perform well with respect to execution time.

4.2 Single-hop BSF algorithms

Single hop BSF algorithms assumes that all nodes in the network are in the radio range of each others. These algorithms take advantage of the fact that a single hop network can be modeled as a complete graph, making it possible, thus, to mimic known graph topologies. Daptardar introduced in [22] a BSF algorithm that constructs a mesh-like or

cube-like scatternets. Similarly, scatternets formed by algorithm dBBlue [23] follows the structure of the well-known de-Bruijn graph, which limits the diameter of the scatternet to $O(\log n)$[4]. Zhang et al. in [24] introduced an algorithm that forms ring-like scatternets. The nodes of the ring are piconets. A slave in a given piconet has either one role or serves as an S/S bridge to connect a pair of piconets. In their algorithm, no M/S bridges are used. Wang et al. in [25] introduced an single-hop BSF algorithm based on virtual coordinates. Each node selects a random virtual location (position) and shares it with all the other nodes. The authors of [25] suggest the use of a geometric structure to build the scatternet such delaunay triangulation, Gabriel graph, relative neighborhood graph, or minimal spanning tree. Barriere et al. introduced in [26] a sophisticated theoretical algorithm based on projective geometry. The main issue in [26] is the high level of abstraction of the scatternet formation procedure which led to ignoring some important system specifications and high execution time in practical Bluetooth networks. The main, and obvious, weakness of single-hop BSF is the necessity that every node is in the radio range of all others. This assumption reduces significantly the applicability of this type of algorithms.

4.3 Tree-based BSF algorithms

Note that any connected graph contains a spanning tree. This observation is used by the algorithms of this category to create tree-like scatternets. The spanning tree structure can be used as a backbone that guarantees the connectivity of the scatternet, whereas additional links may be added to the tree to improve the routing performance as suggested in [11] or improve the fault tolerance of the tree, as it is the case in [27] and [28].

Law et al. introduced in [29] a tree-based BSF algorithm that assumes that the network is a single-hop network. The algorithm is based on the idea of merging subtrees [30], which is commonly used in distributed spanning trees algorithms. Initially, every node is a subtree that contains itself only. Algorithm TSF (Tree Scatternet Formation) [31] is a similar single hop tree-based algorithm. Both algorithms in [29] and [31] mix the procedure of neighbor discovery with the scatternet formation procedure. That is, initially no node has any knowledge of its neighbors. Each node alternates between inquiring and scanning the environment to find new nodes. As a node discovery a new neighbor, it executes certain rules of the scatternet formation procedure. On the other hand, it is assumed in [25] that a separate neighbor discovery phase is executed. After this phase, each node exchanges information with all the other nodes. Then, all nodes build a minimum spanning tree which forms the scatternet. Note that the assumption of single-hop networks simplifies the procedure of scatternet formation. Another similar single-hop tree based algorithm can be found in [32].

Algorithm Bluetree [33] is a multihop tree-based BSF algorithm that initiated a series of algorithms that used the same approach. There is also a series of algorithms that provided modifications on algorithm Bluetree. There are two versions of Bluetree. In the first version, it is assumed that there is a unique node that initiates the algorithm as a root of the tree. The root node captures its neighbors as slaves. The root then assigns its slaves to become masters (i.e. M/S bridges) and capture their neighbors as slaves. If the network is modeled as a unit disk graph, then it is possible with the use of simple rules to restrict the size of any piconet to at most 5. This version of Bluetree did not introduce a leader election algorithm to select the root node. This, in fact, a major weakness of this version of Bluetree. The second version of Bluetree solves the issue of electing a leader. Each node that has the highest identifier among its neighbors considers itself as a root. A given root initiates a procedure similar to that of the first version of Bluetree, and creates a tree. None of the trees shall have a node from another tree. That is, the result of this procedure is a set of disjoint trees that span over all nodes. A second phase in Bluetree is then initiated. Its goal is to connect the disjoint trees in a single tree. Algorithm Bluetree has been slightly modified in [34], [35] and [27].

Cuomo et al. introduced in [28] algorithm SHAPER. The algorithm is an implementation of the well-known distributed MST algorithm by Gallager et al. [30]. Note that the same algorithm of [30] was used as a base for a tree-based BSF algorithm introduced in [13]. In [28], an optimization of the tree structure is introduced. This optimization is a set of heavy calculations, called DSOA (Distributed Scatternet Optimization Algorithm), that generates a mesh-like scatternet. DSOA is a centralized algorithm. In order to execute DSOA in a distributed manner the nodes of the SHAPER tree must be sequentially visited in a Depth First Traversal manner as shown in [28], which is a time-consuming procedure. Methfessel et al. introduced in [36] a modification of SHAPER that overcomes some practical issues in the implementation of SHAPER.

In general, tree-based BSF algorithms main advantage is the simplicity of design. However, implementing such algorithms on Bluetooth networks is a sophisticated procedure that causes increased execution time and overhead in communication [11]. Moreover, tree-like scatternets suffer from weak fault-tolerance and bottleneck at some nodes in the tree.

[4] Given two functions f, g, we say that g is in $O(f)$ if there is a constant $c > 0$ such that $g(n) < c \cdot f(n)$ for a sufficiently large n.

4.4 Mesh-based BSF algorithms

These algorithms solve the main issue of tree-based BSF algorithms by forming mesh-like scatternets. Algorithms belonging to this category are usually simpler to implement and run on top of a neighbor discovery algorithm (i.e. every node knows its neighbors in advance). Some of the major BSF algorithms belong to this category [11, 14]. BlueStars [3] is an important mesh-based algorithm. In BlueStars, a maximal independent set[5] of the input network is first constructed, denoted $MIS(V)$ where V is the set nodes of the network. The nodes of $MIS(V)$ becomes masters and try to capture all their neighbors that have smaller identifiers. No slave is captured by two masters. This forms a set of disjoint isolated piconets. BlueStars implements the previous procedure as follows. A node u waits for all its larger neighbors (i.e. having larger identifier). If none of the larger neighbors slaved u, then u becomes master and attempts to slave its smaller neighbors. Otherwise, u informs its smaller neighbors that it has been slaved. Note that the nodes with no larger neighbors initiate the algorithm. The worst case time complexity of this algorithm is $O(n)$. This is achieved in a network modeled as a line and the nodes are sorted ascendingly according to their identifiers (see more detail in Appendix A). In a second phase of BlueStars, neighbor piconets are interconnected via bridge nodes (called gateways). Two piconets are neighbors if their masters m_1 and m_2 are separated by either 1) one slave s_x belonging to the piconet of either m_1 or m_2 but neighbor to both, or 2) two slaves s_1 and s_2 which are neighbors to each other, s_1 is a slave of m_1 and s_2 is a slave of m_2. Each pair of neighbor piconets selects, using simple local rules, a unique gateway or pair of gateways to be interconnected.

Algorithm BlueStars advantages is the simplicity and the short execution time. Its main disadvantage is its incapability of forming outdegree limited scatternets. This issue has been tackled using different approaches. One of them is the use of location information as in [4]. Given location knowledge at each node, the nodes form a degree-limited geometric structure such as Yao graph [4]. Algorithm BlueStars is then run over this degree-limited structure. Note however that the use of location knowledge is a strong assumption that significantly simplifies the scatternet formation procedure. Another approach to solve this issue is the use of randomization as it is the case in [37]. This approach trades off degree limitation with connectivity. Similarly, Wang et al. introduced in [5] algorithm Bluenet which solves the outdegree limitation problem but does not guarantee connectivity. Note how achieving connectivity and outdegree limitation at the same time can be a challenging task.

[5] An independent set of a network (or a graph) is a set of nodes that none of which is neighbor to another. A maximal independent set is an independent set that is not a subset of any other independent set.

Basagni et al. introduced a deterministic algorithm, called BlueMesh [6], to solve the outdegree limitation issue of BlueStars. Algorithm BlueMesh runs in iterations. Initially, given an input graph $G = (V, E)$, a maximal independent set of V (denoted $MIS(V)$) is constructed using a similar technique to that of BlueStars [3]. Each dominator u (i.e. $u \in MIS(V)$)) selects a subset $S(u)$ of its smaller neighbors $N_{\prec}(u)$ (that is, having a smaller identifier). $S(u)$ has the following properties: 1) its size is at most seven, 2) every node in $N_{\prec}(u)$ is either in $S(u)$ or a neighbor to a node in $S(u)$, and 3) the set is maximal (that is, there is no other subset in $N_{\prec}(u)$ having the previous two properties. A dominator node u then becomes a master to every node in $S(u)$. The previous procedure creates a set of piconets. We say a pair of piconets are neighbors iff: 1) they share one slave or more (called *connected neighbor piconets*), or 2) if there is a pair of slaves s_1, s_2, each of which belonging to one of the piconets, and both are neighbors in the input graph G (called *unconnected neighbors piconets*). The masters of a pair of unconnected piconets select a unique pair s_1, s_2 following a certain criteria. We call these nodes *unique gateways*. The nodes s_1 and s_2 are not masters. Each BlueMesh iteration i repeats the same procedures above executed over the graph G_i, where G_i is defined as the induced graph of the set of all unique gateways generated in the previous iteration $i-1$. The graph G_0 is equivalent to the input graph G. We give in Appendix A the worst case time complexity of BlueMesh accompanied with an illustrative example of BlueMesh. We also analyze the maximum number of iterations of BlueMesh.

Another major BSF algorithm that solves deterministically the outdegree limitation problem is BlueMIS [7]. The novel approach of BlueMIS is to form in short time a connected outdegree limited scatternet that is not necessary efficient with respect to all performance metrics. The formed scatternet is then improved using simple rules. BlueMIS uses a similar idea to those of algorithm XTC [38], which is a topology control algorithm for wireless ad-hoc networks. BlueMIS assumes that the network is modeled as a unit disk graph. The algorithm runs in two phases, BlueMIS I and BlueMIS II. In BlueMIS I, each node passes greedily by its neighbors in an order from the smallest neighbor to the largest neighbor, with respect to the identifier of nodes. A node u adds a neighbor v to $S(u)$ if v is not neighbor to any node in $S(u)$. A node v in $S(u)$ is considered as a slave of u if u is not in $S(u)$ or if u is in $S(u)$ and the identifier of v is smaller than that of u. The execution time of BlueMIS I is improved by algorithm Eliminate introduced in [9]. To our knowledge, BlueMIS I is the first BSF algorithm that has $O(1)$ time complexity (i.e. local). This is of theoretical importance, since the execution of the algorithm does not depends on the number of nodes. The main disadvantage of BlueMIS I is the large number of piconets

(masters) in the formed scatternets. BlueMIS II improves the efficiency of the scatternets by simple rules. The rules are proven to be correct theoretically. However, some implementation details were not included in the description of these rules which resulted in different possible implementations. For instance, there could be cases where a node cannot execute its rules until some other (possibly all other) nodes execute their rules in order to achieve the expected results. This turns the algorithm to be non-efficient with respect to time. Thus, we found that BlueMIS II either suffers from a long execution time or from piconets with large number of slaves depending on the implementation used. More details about BlueMIS can be found in Appendix A.

Lastly, Song et al. introduced in [15] algorithm M-dBBlue. The algorithm idea is to build a connected dominating set S from the input network (that is, any node in the network is either in S or neighbor to a node in S and the network induced by S is connected). The algorithm does not guarantee outdegree limitation, but the authors gave theoretical upper bounds for the formed scatternets maximum outdegree if the input graph is a unit disk graph. The algorithm is based on a heavy tree-based method to construct a dominating set. Furthermore, the algorithm did not include any details of the algorithm implementation.

5 BSF-UED: A New BSF algorithm based on Unnecessary-Edges Deletion

In this section we describe our algorithm, BSF-UED. The algorithm is mesh-based, and runs in two phases. The first forms disjoint outdegree-limited piconets, whereas the second interconnects these piconets. This interconnection in phase 2 may induce scatternets that are not outdegree-limited, but we mitigate this impact by introducing heuristic *H1*, whose main effect is to reduce the number of outdegree unlimited piconets. BSF-UED is inspired by an extensive study of the algorithms BlueStars, BlueMesh and BlueMIS given their interesting distinctive properties.

BSF-UED uses the idea of delegating nodes in a piconet to a different master in order to achieve low average piconet size. BSF-UED gives different colors to the edges of the network and categorizes them into: Type 1) edges that unnecessary for connectivity but may cause exceeding the outdegree limit, Type 2) edges that may be necessary for connectivity but does not cause exceeding the outdegree limit, and Type 3) edges that may be necessary for connectivity and may cause exceeding the outdegree limitation. With this in mind, we attempt to give priority for edges of Type 2 over those of Type 3, and we avoid using edges of Type 1. The coloring of edges is done locally. That is, each pair of nodes decides locally the edge color that they share.

Our algorithm has an $O(n)$ time complexity. We use a wave-like communication rounds implementation adapted to Bluetooth networks in order to decrease the algorithm's execution time. In this implementation, each node is given a unique priority (e.g. its unique identifier). A node waits to receive a message from all its neighbors with higher priority, then it sends a message to all its neighbors with lower priority. This guarantees that each edge is contacted once. This technique, which is used in BlueStars and BlueMesh, was found in [10] to be more efficient with respect to time compared to standard implementations of communication rounds.

5.1 Assumptions

We use the same set of assumptions used in BlueMesh and BlueMIS. Each node in the input graph $G = (V, E)$ has a unique (and comparable) identifier. Using this order on V, we say node u is larger (smaller) than node v if the identifier of u is larger (smaller) than that of v, denoted as $u \succ v$ ($u \prec v$). Given a graph or network, we denote $N(u)$ as the set of neighbors of u, $N_{\prec}(u)$ as the set of smaller neighbors of u and $N_{\succ}(u)$ as the set of larger neighbors (that is, $v \in N(u)$ iff $(u, v) \in E$, and $v \in N_{\succ}(u)$ iff $v \in N(u)$ and $v \succ u$).

Using the total order on V, we consider the acyclic directed graph $\overrightarrow{G}(V, \overrightarrow{E})$ such that $\overrightarrow{E} = \{(u, v) \in E : u \succ v\}$. Note that the graph $\overrightarrow{G}$ is used frequently in this paper. We refer to $\overrightarrow{G}$ sometimes as the *directed version of the input graph*.

The input graph G is assumed to be a unit disk graph (UDG); but some other graphs that we manipulate are not. An interesting property of unit disk graphs is that any node in the graph can *cover* all its neighbors by at most 5 neighbors. In other words, a node u in a unit disk graph G can have at most 5 neighbors that are not neighbors to each other. We assume that a node has no knowledge of its location neither of the distance to its neighbors. As most mesh-based BSF algorithms (if not all), we assume that the nodes are static. Thus, we do not treat nodes joining or leaving the scatternet.

5.2 Phase 1: piconet construction

Given the directed version of the input graph $\overrightarrow{G} = (V, \overrightarrow{E})$, the first phase of BSF-UED generates a forest $\overrightarrow{G}' = (V, \overrightarrow{E}_b)$ of disjoint outdegree-limited piconets such that every node is either master or slave in exactly one piconet. Our technique is inspired from a technique proposed in BlueStars [3] (which does not limit the number of slave per nodes).

5.2.1 Informal strategy

All nodes identifiers are unique and ordered, and therefore some nodes must be local maxima (that is, larger than all

their neighbors). This property is used to initiate a wave-like process whereby larger nodes successively attempt to capture (*i.e.,* to slave) smaller neighbors. Nodes cannot be captured twice nor capture other nodes once they are themselves slaves. In order to limit the outdegree, we adapt a number of delegation rules by which nodes can control the number of slaves they capture and delegate excesses (if any) to other neighbors. We prove that such delegation is feasible sufficiently often to limit the number of slaves to 7, thanks to the UDG assumption. The rules are described formally in the next section.

5.2.2 Detailed strategy

The state of a node u is denoted by $state(u) \in \{none, master, slave\}$. The state of an edge (u, v) is denoted by $c(u, v) \in \{white, black, silver, green, red, blue\}$. Initially, $state(u) = none$ for all $u \in V$, and $c(e) = white$ for all $e \in E$. Given an edge $(u, v) \in \overrightarrow{E}$ (keeping in mind that $u \succ v$), the meaning of each color is as follows:

- $black$: u captured v.
- $silver$: u contacted v, but v was already captured.
- $green$: u was captured by a third node and thus gave up on capturing v.
- red: u delegated the capture of v to another node w such that $u \succ w \succ v$.
- $blue$: u delegated to v the capture of a common neighbor w such that $w \prec v \prec u$.

The first phase of BSF-UED colors all the edges of the graph $\overrightarrow{G}(V, \overrightarrow{E})$ by means of Algorithms 1 to 4. At every node u a variable $\varphi(u)$ denotes its capacity (initially set to 7) accounting for the number of slaves it can capture.

Each node is considered as a potential *prey* to all its larger neighbors. A prey can be captured only by one of its larger neighbors. In case a node u was not captured by any of its larger neighbors, it will consider its smaller neighbors $N_{\prec}(u)$ as preys and attempts to capture them. An attempt of capture fails if the prey is already captured by another node. We denote the set of preys of u as $preys(u)$, initially equal to $N_{\prec}(u)$. We add the following rules to the procedures of capturing in order to limit the number of slaves per master to at most 7. Whenever a node u starts the capture procedures, if $preys(u) \leq 7$, then u attempts to capture all of them. Otherwise, u goes through each of them in decreasing order. For each prey v, u finds a subset of common smaller neighbors $CN(u, v)$ that it shares with v (procedure `FindCommonNeighbors()` on Algorithm 4). If $CN(u, v) = \emptyset$, then u attempts to capture v. Otherwise, u delegates some of the common neighbors in $CN(u, v)$ to v and does not capture v either. Node v and the neighbors in $CN(u, v)$ are then removed from $preys(u)$, and the process repeats until u has enough capacity to capture the remaining neighbors (that is, $preys(u) \leq \varphi(u)$). The details of these procedures are illustrated in Algorithms 1, 2, 3 and 4. A flow diagram of these procedures is available in Figure 12 in Appendix B.

Algorithm 1 Procedure `construct()` at node u

```
1: while ∃v ∈ N≻(u) : c(v,u) = white do
2:     wait
3: if state(u) = slave then
4:     for all v ∈ N≺(u) do
5:         c(u,v) ← green.
6: else
7:     state(u) ← master
8:     capture()
```

Algorithm 2 Procedure `capture()` at node u

```
1: preys ← N≺(u)
2: while (|preys| > φ(u)) do
3:     v ← max(preys)
4:     preys ← preys − {v}
5:     CN(u,v) ←FindCommonNeighbors(v)
6:     if CN(u,v) = ∅ then
7:         contact(v).
8:     else
9:         c(u,v) ← blue
10:        preys ← preys − {CN(u,v)}
11:        for all x ∈ CN(u,v) do
12:            c(u,x) ← red
13:            φ(x) ← φ(x) − 1          {to be explained in phase 2}
14: for all x ∈ preys do
15:     contact(x).
```

Algorithm 3 Procedure `contact(`v`)` at node u

```
1: if state(v) ≠ slave then
2:     state(v) ← slave
3:     c(u,v) ← black.
4:     ρ(u) ← ρ(u) ∪ v
5: else
6:     c(u,v) ← silver.
7: φ(u) ← φ(u) − 1.
```

Remark 1 Procedure `contact(`v`)` of Algorithm 3 is considered to be atomic: if nodes u and w attempts to contact a node v, then only one of them can enter procedure `contact(`v`)` at a time. This is guaranteed in Bluetooth specifications, since a node can communicate only with one node at a time.

5.2.3 Procedure `FindCommonNeighbors(`v`)`

Called at a node u, this procedure is responsible for selecting a set of common smaller neighbors $CN(u, v)$ to be delegated to v. Precisely, given the set $L = \{preys(u) \cap N_{\prec}(v)\}$

of potentially capturable common neighbors between u and v, the procedure selects the largest subset $L' \subseteq L$ of nodes such that:

- $|L'| \leq 7$: no more than 7 nodes are delegated.
- $|L - L'| \geq \varphi(u)$: u does not delegate a number of nodes that would leave it with less than $\varphi(u)$ neighbors to contact.

As a result, u delegates a number of neighbors equal to $\min(|preys(u)| - \varphi(u), 7, |L|)$. By convention, these neighbors are those whose identifiers are the largest. (See Algorithm 4.)

Algorithm 4 Procedure `FindCommonNeighbors(`v`)` at node u

1: $L \leftarrow \{preys(u) \cap N_{\prec}(v)\}$
2: $nb \leftarrow \min(|preys(u)| - \varphi(u), 7, |L|)$
3: **return** largest nb nodes in L

An example illustrating all procedures of Phase 1 is provided next in Section 5.2.4, followed by theoretical analysis of correctness in Section 5.2.5.

5.2.4 Example

We give in this section an example to describe in more details the procedures of Phase 1. Consider the graph in Figure 1 - A. Initially, $\varphi(u) = 7$ for every node u. The nodes that are larger than all their neighbors are 35 and 21 (see Figure 1 - B). Node 21 has only two smaller neighbors, which are 12 and 14. Thus it contacts both of them directly. Since both 12 and 14 are not slaves to any other nodes, 21 captures them (*thicker directed arrows*).

The case of 35 is different since $|preys(35)| = 9$. Since 9 is larger than $\varphi(35) = 7$, node 35 selects the largest neighbor in $preys(35)$ [line 3, `capture()`], which is 30. The set of common neighbors between 35 and 30 is $\{1\}$. Thus, edge $(35, 30)$ is colored blue, while $(35, 1)$ is colored red. This means that node 35 delegated to 30 the responsibility of capturing nodes $\{1\}$. Then, all the remaining preys are captured by 35.

The nodes that were contacted by all their larger neighbors, which are 30, 25, 22 and 12, start the next round (See Figure 1 - C). Node 30 contacts both of its smaller neighbors 1 and 14. Neighbor 1 is captured and colored black, whereas neighbor 14 is not because it was already captured by node 21. Thus, edge $(30, 14)$ is colored silver. Nodes 22 and 12 are already captured by 35. Thus, they inform all their smaller neighbors that they are slaves by coloring the corresponding edges in green. The final result of Phase 1 is the three piconets depicted on Figure 1 - F, that are, $\rho(35) = \{35, 25, 6, 22, 16, 15, 8, 7\}$, $\rho(30) = \{30, 1\}$, and $\rho(21) = \{14, 12\}$.

5.2.5 Correctness

Given the input graph $\overrightarrow{G} = (V, \overrightarrow{E})$, we prove that phase 1 always terminates and that its output is a set of disjoint piconets that are outdegree limited to 7, and every node is either master or slave in exactly one piconet. For clarity, we denote by E_{color} the subset of those edges that are colored $color$ (e.g., $E_{black} = \{(u, v) \in \overrightarrow{E} : c(u, v) = black\}$).

Let us first observe that the algorithm always terminates.

Lemma 1 (Termination) *Phase 1 of BSF-UED terminates in a finite time.*

Proof Procedure `construct()` is executed over the directed input graph $\overrightarrow{G} = (V, \overrightarrow{E})$. Since the nodes can be ordered with respect to identifiers, then $\overrightarrow{G}$ is a directed acyclic graph (DAG), and there is at least one node u with $N_{\succ}(u) = \emptyset$ (called sources) and v with $N_{\prec}(v) = \emptyset$ (called sinks) and $u \neq v$. In a DAG the set of all paths from sources to sinks cover all the nodes, and each path $\pi = \{u_1, ..., u_k\}$ is decreasing (that is, $u_i > u_{i+1}$ for $1 \leq i < k$). Note that each non-sink node will necessarily color all of its outgoing edges (all possible execution sequences starting at `construct()` eventually lead to such a coloring). Therefore, no edge remains white if all the paths of $\overrightarrow{G} = (V, \overrightarrow{E})$ are used; which is what the activation control in `construct()` guarantees.

Lemma 2 *After the execution of the first phase, each node is either a master or slave.*

Proof This is clear by the content of procedure `construct()`; upon activation, if a node was not yet made slave during its waiting period, then it turns itself as a master.

Lemma 3 shows that Phase 1 results in disjoint piconets. We denote the set of these piconets as $\mathcal{V}$.

Lemma 3 (Disjoint piconets) *Let's $G = (V, E_{black})$ be the spanning subgraph of the input graph $\overrightarrow{G} = (V, \overrightarrow{E})$ with the edges $E_{black} \in \overrightarrow{E}$. Then, after Phase 1 G is a forest of disjoint piconets, denoted as $\mathcal{V}$.*

Proof The proof follows from Lemma 2 and the condition that is not possible for a node to be captured twice (see procedure `contact()`). □

We prove in Lemma 4 that the generated set of piconets $\mathcal{V}$ in $G'(V, E_{black})$ are all outdegree limited to 7, given that $\varphi(u)$ is initialized to 7.

Lemma 4 *Let's $G = (V, \{E_{black} \cup E_{silver}\})$ be the spanner subgraph of the input graph $\overrightarrow{G} = (V, \overrightarrow{E})$ with the set of edges $\{E_{black} \cup E_{silver}\} \in \overrightarrow{E}$. Then, after the Phase 1 G is outdegree-limited to 7, given $\varphi(u)$ is initially set to 7.*

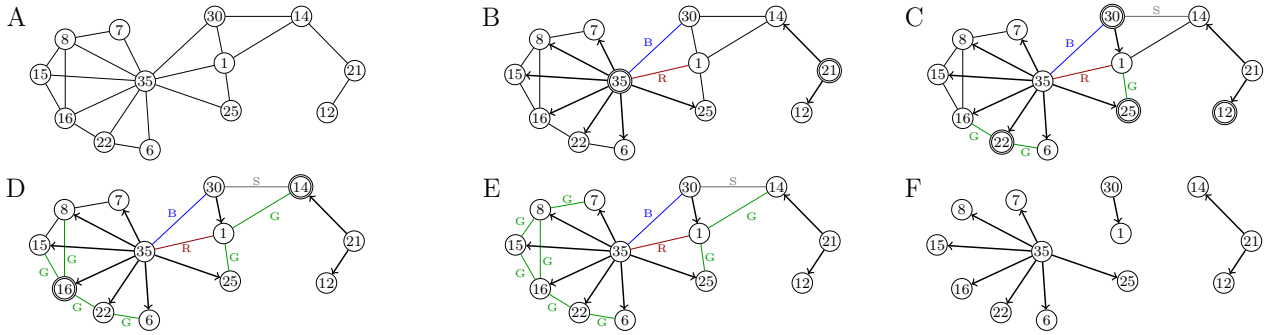


Fig. 1: Example illustrating the procedures of phase 1. (*B: Blue, R: Red, G: Green, S: Silver*)

Proof We need to compute the number of times a node u calls procedure `contact(`v`)`, since we are interested only in black and silver edges. Given that $\varphi(u)$ is initialized to 7, any node u will color at most 7 of its smaller neighbors (preys) with black or silver (that is, attempt to contact them using procedure `capture(`v`)`.

Without loss of generality, assume that $|preys(u)| > 7$. At the while-loop [lines 2 to 13, `capture()`], a node u contacts v if and only if v and u had no common neighbors that are smaller than both u and v. There is at most five neighbors of u having the property of v given the unit disk graph property of our input graph. Therefore, procedure `contact(`v`)` is called from the while-loop [lines 2 to 13] at most 5 times. Since the execution of the loop stops only if $preys(u) < \varphi(u) = 7$, and since each time a node u executes `contact(`v`)`, $\varphi(u)$ is decreased by 1, then u will not execute `contact(`v`)` in the while-loop and the for-loop [lines 14 to 15], combined, more than 7 times at most. □

The combination of Lemmas 1 to 4 allows us to conclude as follows:

Theorem 1 *Phase 1 produces a set of disjoint piconets that are outdegree limited to 7, and such that every node is either master or slave in exactly one piconet.*

We will now introduce an extra Lemma, not strictly relevant to the objectives of Phase 1, but which will be helpful to prove the correctness of Phase 2. It establishes that the set of blue edges are not necessary for the connectivity of the input graph $G = (V, E)$.

Lemma 5 *Let's* $G_1 = (V, \{\overrightarrow{E} - E_{blue}\})$ *be the spanner subgraph of the input graph* $\overrightarrow{G} = (V, \overrightarrow{E})$ *with the set of edges* $\{\overrightarrow{E} - E_{blue}\} \in \overrightarrow{E}$*. Then, after the Phase 1,* G_1 *is connected.*

Proof Note that if an edge $(u, v) \in E_{blue}$, then v is the largest node among $preys(u)$ at that while-loop iteration [lines 2 and 13, `capture()`]. If there is an edge (u, v) that is colored blue, then there must exist two edges (u, w) and (w, v) in $\overrightarrow{E}$ such that $u \succ v \succ w$. Therefore, u, v and w forms a triangle, where edge (u, w) is colored red [line 12 of `capture()`]. We need to show that, following a specific ordering of edges, a minimum spanning tree (MST) of $\overrightarrow{G}$ will not include blue edges.

The ordering of edges that we follow is a lexicographical order in which we assume that $(u_1, v_1) \succ (u_2, v_2)$ if $u_1 \succ u_2$ or if $u_1 = u_2$ and $v_1 \succ v_2$). Note now that for any triangle u,v and w such that (u, v) is blue and (u, w) is red and $u \succ v \succ w$, then the blue edge (u, v) will surely be not included by Kruskal algorithm in the MST as it is the largest edge in that 3-circle. This completes the proof. □

5.3 Phase 2: Piconets Interconnection

The second phase of BSF-UED interconnects the disjoint piconets formed in Phase 1 to form the output scatternet $\mathcal{S} = (V, E')$. This phase guarantees the connectivity of the resulting scatternet $\mathcal{S} = (V, E')$, while maintaining its maximum outdegree to a reasonable value.

5.3.1 Informal strategy

The problem can be formulated as a meta-graph problem in which every piconet formed in Phase 1 is a vertex. The objective is to define edges in this meta-graph in a way that guarantees its connectivity. The edges of the meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are to correspond to edges or paths in the input graph $G = (V, E)$. We then apply to this meta-graph a technique inspired from BlueMIS I [7] to interconnect vertices. The resulting strategy is as follows. Each node u in $\mathcal{V}$ constructs a maximal independent set of its *larger* neighbors, denoted $MIS_{\succ}(u)$. (A set of nodes is said *independent* if it does not contain any pair of neighbor nodes; it is *maximal* if the addition of any node makes it no more independent.) Then, u interconnects only to the nodes in $MIS_{\succ}(u)$. This technique guarantees the connectivity of the new graph. It also guarantees its outdegree limitation in case $\mathcal{G}$ is a unit disk graph,

which unfortunately is not necessarily the case (however it is almost always so in practice).

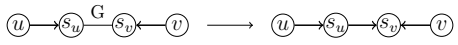
The input of Phase 2 is the graph $\overrightarrow{G} = (V, E_{black})$, which is the graph that contains all the master/slave relationships formed in Phase 1. Keep in mind that an edge $(u, v) \in E_{black}$ indicates that u is the master of v and that $u \succ v$. The output of the Phase 2 is $G(V, \{E_{black} \cup E_{black'}\})$, where $E_{black'}$ is the master/slave relationships formed in Phase 2. It is not necessarily that for each $(u, v) \in E_{black'}$ that $u \succ v$. The procedures of Phase 2 are described in the sequel.

5.3.2 Detailed strategy

Let us consider the meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose vertices $\mathcal{V}$ are the piconets formed in Phase 1 and edges $\mathcal{E} = \emptyset$ are initially an empty set. The objective of Phase 2 is to define $\mathcal{E}$ in such a way that $\mathcal{G}$ becomes connected and every edge in $\mathcal{E}$ corresponds to a real path connecting two piconet masters in G (the input graph). We call two piconets $\rho(u)$ and $\rho(v)$ *neighbors* if their masters u and v can be interconnected through one of the following types of paths: master-to-master (*one-hop interconnection*), master-to-slave (*two-hops interconnection*), or slave-to-slave (*three-hops interconnection*). If each piconet interconnects with all its neighbor piconets, the resulting graph is necessarily connected (see Theorem 2)

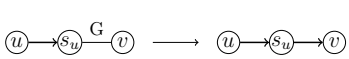
We first discuss the potential interconnection of each pair of neighbor piconets through such a path, and then consider, in a second step, the possibility to actually discard a number of edges that are unnecessary to the connectivity. The following *interconnection rules* are considered, listed in priority order. (We provide more details later on about their concrete implementation.) Note that only one interconnection rule is to be applied relative to a given pair of neighbor piconets, more would be unnecessary with respect to the final scatternet's connectivity. Without loss of generality, we assume below that $u \succ v$.

I-**Rule 1** (*Three-hop interconnection*): Two piconets $\rho(u)$ and $\rho(v)$ may be interconnected through an edge e between two slaves $s_u \in \rho(u)$ and $s_v \in \rho(v)$, where $c(e) = green$. (Operation: s_u captures s_v.)



I-**Rule 2** (*Two-hop interconnection*):

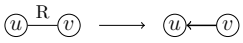
I-**Rule 2a**: through the edge $(s_u, v) \in \overrightarrow{E}$; where $c(s_u, v) = green$, s_u is a slave of u and v is a master of piconet $\rho(v)$. (Operation: s_u captures v.)



I-**Rule 2b**: through the edge (v, s_x) or (u, s_x) where $s_x \in \rho(u)$ or $s_x \in \rho(v)$, and $c((v, s_x)) = silver$ or $c((u, s_x)) = silver$ (that is, s_x is smaller than both u and v, and it belong to either $\rho(u)$ or $\rho(v)$ but not both. Both piconets attempted to slave s_x but only one of them was successful). (Operation: v captures s_x or u captures s_x.)

I-**Rule 2c**: through the edge (u, s_v) where $s_v \in \rho(v)$ and $c((u, s_v)) = red$ (that is, s_v is smaller than both u and v. s_v is slaved by v, and s_v was delegated by u to v). (Operation: s_v captures u.)

I-**Rule 3** (*One-hop interconnection*): through the edge $(u, v) \in \overrightarrow{E}$ where $c((u, v)) = red$. Both u and v are masters of different piconets. (Operation: v captures u as $v \prec u$.) [6]
This case occurs if v delegated to a neighbor w the responsibility of u, and hence (v, u) is red. Then, u becomes master. For instance, there is a node a w' that is larger than u and delegated *to* u the responsibility of slaving common neighbors between u and w'.



We call the nodes that are used to interconnect two piconets *gateways* (e.g, nodes u and s_v in I-Rule 2c). It is of course desirable to try to minimize the number of slaves for any gateway s_u used to interconnect a piconet $\rho(u)$ and $\rho(v)$. This number can however not be strictly limited to 7 because $\mathcal{G}$ is not necessarily a unit disk graph.

This strategy to construct the edges set $\mathcal{E}$ in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ guarantees the connectivity of $\mathcal{G}$. Indeed, the only type of edges we do not consider in these rules are the blue edges, which we prove not necessary for connectivity in Lemma 5.

Once the meta-graph $\mathcal{G}$ is connected, we use a technique inspired from [7] in order to delete further edges from $\mathcal{E}$ that are unnecessary for the connectivity of $\mathcal{G}$. The technique can be implemented using simple local rules executed at the vertices of $\mathcal{G}$. (Keep in mind that in our case the vertices are piconets.) Since each piconet has one master, the local rules will be implemented in the masters of the piconets of $\mathcal{G}$. We describe next this technique.

Let us denote by $\mathcal{N}_{\succ}(u)$ the set of all neighbor piconets of $\rho(u)$ whose master v is such that $v \succ u$. We define $\mathcal{N}_{\prec}(u)$ symmetrically. Each piconet $\rho(u)$ constructs a maximal independent set $MIS_{\succ}(u)$ among $\mathcal{N}_{\succ}(u)$ in $\mathcal{G}$. Then all the edges of G that do not lead to a node of $MIS_{\succ}(u)$ are discarded. The resulting graph is guaranteed to remain connected (see Theorem 3).

[6] In this rule, the fact that v captures u (and not the opposite) is important; it follows the anticipated decrease of $\varphi(v)$ in Phase 1 when the edge (u, v) was colored red. In a sense, v is "more prepared" than u to handle new slaves. The impact of this operation is seen while starting the elimination algorithm from smaller to larger piconets.

5.3.3 Implementation Details

The detailed implementation is given in Algorithm 5, and illustrated in a flow diagram in Appendix B. We give an explanation in the following. The decision of which edge (rule) should be used to interconnect two piconets $\rho(u)$ and $\rho(v)$ in $\mathcal{G}$ is local to the masters u and v. Each master u of a piconet $\rho(u)$ constructs a *gateway table* (denoted as $\mathcal{T}(u)$). The entries of a gateway table represent all the rules that can be applied to merge with neighbor piconets. Each entry consists of the following elements:

- v: the master of the neighbor piconet $\rho(v)$.
- s_u: the gateway of piconet $\rho(u)$ (note that s_u may be equivalent to u).
- s_v: the gateway of piconet $\rho(v)$ (note that s_v may be equivalent to u).
- $\varphi(s_u)$: the piconet capacity of s_u.
- I: the interconnection rule of the tuple (that is, I-Rule 1, ..., I-Rule 3).
- `role`: the role to be played in the new relation (either M or S). If `role` is M, s_u becomes master to s_v according to interconnection rule I. If `role` is S, s_u becomes slave to s_v.

As already mentioned, a given neighbor piconet $\rho(v)$ can be interconnected to $\rho(u)$ by multiple rules. Therefore, the gateway table $\mathcal{T}(u)$ may contain several entries for a same neighbor piconet $\rho(v)$, with different s_u and s_v.

The construction of the gateway table $\mathcal{T}(u)$ is straightforward. It is sufficient that each master u collects from its slaves s_u information about their neighbors v_i; namely, their master $m(v_i)$, capacity $\varphi(v_i)$, and the color of the edge (s_u, v_i). Using such information, u can infer all the possible rules of interconnection.

The next step is to let each piconet $\rho(u)$ construct a maximal independent set of its larger piconet neighbors ($MIS_{\succ} \subseteq \mathcal{N}_{\succ}(u)$). The construction is done on-the-fly and is described in procedure `interconnect()`; each time a piconet $\rho(u)$ interconnects to a larger neighbor piconet $\rho(v)$, it does not interconnect with any neighbor piconet $\rho(w)$ that is common neighbor to both $\rho(u)$ and $\rho(v)$ and has a larger identifier than both. Any master/slave relationship between nodes s_u and s_v added by procedure `interconnect()` is represented as an edge (s_u, s_v), where s_u is the master of s_v regardless of their identifiers. The set of all edges interconnected by `interconnect()` is denoted $E_{black'}$. The output of `interconnect()` therefore is the graph $G = (V, \{E_{black} \cup E_{black'}\})$ (remind that E_{black} is the set of master/slave relationships formed in Phase 1).

An issue that needs more clarification in `interconnect()` is how to select the best pair of gateways to interconnect two piconets $\rho(u)$ and $\rho(v)$ [line 7]. In order to do this, a node u ascendingly sorts the rows of $\mathcal{T}(u)$ in a lexicographical order of $(\rho(v), d(I), -\varphi(s_u))$, where $d(I)$ is a number

Algorithm 5 Procedure `interconnect()` at piconet u

```
1: U_l ← N_≻(u), U_s ← N_≺(u), B ← ∅
2: while (U_s ≠ ∅) do
3:     wait for gateways of smaller piconets to contact.
4:     upon receipt of a message from a gateway v;
       U_s ← U_s − {v}.
5: while (U_l ≠ ∅) do
6:     v ← min(U_s)
7:     select best row (v, s_v, s_u, φ(s_u), I, role) ∈ T(u)
8:     if (role = M) then
9:         u orders s_u to be master of s_v
10:        φ(s_u) ← φ(s_u) − 1
11:        update T(u)
12:    else
13:        u orders s_u to be the slave of s_v
14:        φ(s_v) ← φ(s_v) − 1
15:    B ← {B ∪ {N_≻(u) ∩ N_≻(v)}}
16:    U_l ← U_l − {v ∪ N_≻(v)}
17: for all (piconet x ∈ B) do
18:    select any tuple (v, s_v, s_u, φ(s_u), I, role) ∈ T(u)
19:    u asks s_u to contact s_x without any operation of M or S.
```

given to the interconnection rule I (that is, $d(I$-Rule 1) = 1, $d(I$-Rule 2a) = 2, $d(I$-Rule 2b) = 3 etc ..). We say a set $(x_1, .., x_k)$ lexicographically succeeds the set $(y_1, .., y_k)$ if $x_i = y_i$ and $x_j \succ y_j$ for $j = i + 1$ and for all $1 \leq i < k$. Whenever master u attempts to select the best gateway to a neighboring piconet $\rho(v)$ with master v, it simply finds the first occurrence of v in $\mathcal{T}(u)$. That is, master u starts interconnecting with the smallest neighbor piconets to the larger ones. In case of multiple choices, master u prefers the lower rules (that is, I-Rule 1 to I-Rule 2a etc ..), and in case of multiple choices, master u selects the slave to interconnect with $\rho(v)$ via the slave s_u with the maximum piconet capacity $\varphi(s_u)$. An example illustrating the procedures of Phase 2 is given in Figure 2.

We should note that each time a gateway s_u becomes master to a gateway s_v, the piconet capacity of s_u ($\varphi(s_u)$) is decreased by one [line 10 - `interconnect()`]. Thus, the gateways table $\mathcal{T}(u)$ should be updated after each of such changes by simply sorting it again [line 11 `interconnect()`]. Note that using this method a gateway s_u with higher capacity $\varphi(s_u)$ is always preferred. Also, this method preserves the priorities of the interconnection rules.

5.3.4 Correctness

We prove that the output of Phase 2, which is $G = (V, E' = \{E_{black} \cup E_{black'}\})$, is a connected scatternet. First, we prove the connectivity of the meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Recall that the vertices of $\mathcal{G}$ are the piconets formed in Phase 1, while $\mathcal{E}$ contains edges to interconnect neighbor piconets based on the rules of Section 5.3.2.

Theorem 2 (Connectivity) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be such that $(u, v) \in \mathcal{E}$ iff u and v are two piconets that can be inter-*

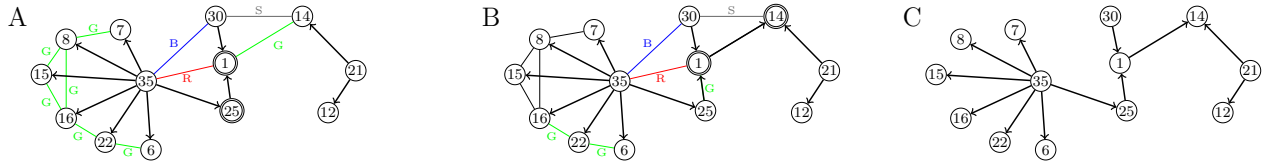


Fig. 2: Example illustrating the procedures of phase 2, taking over the resulting graph of the example in Phase 1 (Figure 1). (*B: Blue, R: Red, G: Green, S: Silver*). Those piconets that have no larger piconets are the ones to start, while others wait. Piconet 35 starts the execution. The smaller neighbor piconets are piconet 30. The highest priority interconnection between these piconets is through the edge $(25, 1)$ (I-Rule 1). Node 25 captures node 1 since its master 35 is larger than 30. Piconet 30 then starts. It creates a connection with its smaller neighboring piconet 21 through the edge $(1, 14)$ (I-Rule 1). In this case, 1 becomes the master of 14 because the master of 1 is larger than the master of 14.

connected with any of the interconnection rules. *Then, $\mathcal{G}$ is connected.*

Proof The interconnection rules consider all cases of green, silver and red edges (*i.e.,* all edges in the sets E_{green}, E_{silver} and E_{red}). By definition, the endpoints of any edge (u, v) with silver or red color must belong to two different piconets, while green edges may be between nodes belonging to the same or different piconets. Thus, any such edge interconnects two different piconets. The interconnection rules does not consider blue edges, which are not needed for the connectivity of $G = (V, E)$ according to Lemma 5. □

In Theorem 3, we prove that the graph $\mathcal{G}$ remains connected as long as each vertex $u \in \mathcal{V}$ keeps an edge (u, v) for each $v \in MIS_{\succ}(u)$, where $MIS_{\succ}(u)$ is the maximal independent set of larger neighbors of u.

Theorem 3 *If a graph $G = (V, E)$ is connected, then the graph $G' = (V, \{(u, v) \in E : v \in MIS_{\succ}(u)\})$, where $MIS_{\succ}(u)$ is the maximal independent set of larger neighbors of u, is also connected.*

Proof For simplicity, let us refer to this technique as *Algorithm A*. We show that the graph G' resulting from execution of Algorithm A contains a minimum spanning tree if we follow a specific ordering of the edges E. We give a lexicographical order to the edges such that $(x_1, y_1) \prec (x_2, y_2)$ iff $y_2 \succ y_1$ or $y_1 = y_2$ but $x_2 \succ x_1$. For every triangle in G of three edges (x, y), (y, z) and (x, z) of three vertices x, y, z where $x \succ y \succ z$, Algorithm **A** deletes only the edge (x, z). Note that $(x, y) \prec (y, z) \prec (x, z)$. We follow Kruskal algorithm for minimum spanning tree, we order the edges ascendingly, and select greedily the edges that form a tree. Edge (x, z) will never be considered in the MST of G as it is always the maximum edge in the 3-circle $\{(x, y), (y, z), (x, z)\}$. □

Theorem 4 *The graph $G = (V, \{E_{black} \cup E_{black'}\})$ is connected.*

Proof $G = (V, \{E_{black} \cup E_{black'}\})$ is the output of `interconnect()`. Note that `interconnect()` let each node u in the graph $\mathcal{G}$ connects to all its neighbors $v \in MIS_{\succ}(u)$. Recall that each node in $\mathcal{G}$ is a piconet formed in Phase 1. According to Theorem 2, $\mathcal{G}$ is connected. Therefore, the connectivity follows from Theorem 3.

Theorem 5 *The graph $G = (V, \{E_{black} \cup E_{black'}\})$ is a scatternet.*

Proof We prove that no two pair of nodes are slaves and masters to each other at the same time. $G = (V, E_{black})$ is a set of disjoint piconets. Thus, we consider only the added master/slave relationships at Phase 2, which are in $E_{black'}$. The proof follows from three arguments. First, If two piconets $\rho(u)$ and $\rho(v)$ are interconnected in procedure `interconnect()`, then u is larger than v (see line 6 in `interconnect()`). Second, note that $\rho(u)$ contacts a neighbor piconet $\rho(v)$ only once (see line 16 in `interconnect()`). Third, according to Lemma 3, a slave s_u belongs to only one piconet and each node is either a slave or a master according to Lemma 2. Therefore if a gateway s_u was assigned to be a master or slave to another gateway s_v, then s_v cannot have the same role in a later stage of `interconnect()`. □

5.4 Heuristic optimization

We introduce in the following a heuristic that improves the qualities of the generated scatternets.

5.4.1 Heuristic H1: Decreasing the Number of Outdegree Unlimited Piconets

One property of the previous interconnection procedure is that if no red edges used for interconnection, then the algorithm forms outdegree limited scatternets. Note that black and silver edges do not cause excess on the outdegree limitation of the scatternet (see Lemma 4). Note also that blue edges are not used in the scatternet construction and do not cause disconnectivity of the scatternet. Therefore, it remains to show that green edges do not cause excess in the outdegree limitation of the scatternet. Consider a scenario in which a node u is master to node v. In the interconnection piconet, node v shall use more than 7 green edges to connect

with neighboring piconets. Assume that node v must be the master of all the nodes on the other end of its green edges (call them green nodes of v). Let assume that interconnection rules not involving red edges are not used. Then, there is a maximal independent set of the green nodes of v that is of size at most five. This is because edges between these green nodes are either silver or green. This result, however, that red edges are not used in the interconnection process.

We adapt to these results by introducing a heuristic that improves the properties of the scatternet formed. It should be noted that the number of outdegree limited piconets in scatternets formed by BSF-UED is acceptable as simulation experiments show. This heuristic eliminates virtually all outdegree unlimited piconets, while not increasing the execution time of the algorithm. The heuristic is as follows.

Assume that piconet u assigned gateway s_u to become master of gateway s_v. Assume s_u is already a master to 7 slaves. Note that s_v may have slaved some nodes in the interconnection phase. Node s_u checks if s_v is a master to less than 7 slaves. If this is the case, then s_v becomes the master of s_u instead. If s_u had less than 7 slaves, this heuristic is not executed. Interestingly, such a simple rule can improve the properties of the scatternet significantly, as shown by the simulation results in Section 6.

6 Simulation experiments

We study the performance of our algorithm with simulation experiments. We compare BSF-UED against BlueStars, BlueMesh, BlueMIS I and BlueMIS II which are considered as the reference BSF algorithms from the literature. The performance metrics we consider are: 1) execution time, 2) maximum piconet size (or maximum outdegree), 3) average piconet size, 4) number of piconets (masters), 5) number of M/S bridges, 6) number of S/S bridges and 7) average number of role per node.

Our simulation experiments are conducted with the UCBT (University of Cincinnati BlueTooth) simulator [39], which is an NS-2 [40] based library for Bluetooth networks simulation. The networks are modeled as unit disk graphs. Each graph is constructed by placing points uniformly at random in a $30 \times 30m^2$ plane. An edge connects two points if the euclidean distance between them is less than a threshold t set to 10 m, which is generally taken as the radio range of Bluetooth. A graph is considered for experiment only if it was connected. We form five sets of graphs, each with a different size (30, 50, 70, 90 and 110 nodes). Each set consists of 1000 graphs. Unfortunately, theoretical analysis is hard in the case of BSF-UED as it is in most BSF algorithms in literature.

Our findings show that BSF-UED is a time-efficient BSF algorithm. It has a similar execution time to BlueMIS I and about 1/3 the execution time of BlueMesh. Algorithms BlueStars cannot be compared with the other algorithms because BlueStars does not guarantee outdegree limitation, which as a result significantly simplifies the algorithm design. We include BlueStars in our comparison study a a benchmark. In term of other performance metrics, our results show that BSF-UED is always considered among the best algorithms. In term of outdegree limitation, most of the scatternets BSF-UED forms are outdegree limited to 7. Only one experiment among 5000 generated a scatternet that is not outdegree limited. In our analysis, any node considering itself a master without having any slaves is treated as a non-master. This is applied to all algorithms. All figures are represented as bars plots. Deviation from the mean value is shown with error bars that represents the standard deviation.

6.1 Execution time

Figure 3 shows a comparison of the execution time of the algorithms in hand. BlueStars outperforms the other algorithms because of its simplicity. This is obtained with the cost of having piconets with very large size. The execution time of BSF-UED is about 1/3 of that of BlueMesh. BSF-UED and BlueMIS I have similar execution times. However, note that this is an optimized version of BlueMIS I which we introduced in [9]. The execution time of the original BlueMIS I is about 2 times what is indicated in Figure 3. This difference should be taken into consideration when analyzing BlueMIS II since it runs on top of BlueMIS I. In fact, we introduced in [9] algorithm Eliminate which forms scatternets that have very similar properties to those formed by BlueMIS but with an execution time significantly shorter than that of the optimized version of BlueMIS I.

The simulation experiments show an interesting property of Bluetooth networks. First, note that the time complexity of algorithm BlueMIS (I and II) is $O(1)$ whereas the time complexity of the other algorithms is $O(n)$ on average. A detailed analysis is given in Appendix A. This analysis states that in large sized networks, it is expected that BlueMIS outperforms the other algorithms in term of execution time. Yet, this is not the case in our simulation experiments. This phenomenon is related to another result studied in [10]. We argued in [10] that the implementation of communication rounds in Bluetooth networks affect significantly the execution time of BSF algorithms. We define a communication round as a period of time in which each node sends and receives a message to and from all its neighbors. The link establishment procedures of Bluetooth complicates the implementation of communication rounds. We studied two implementations: RandomExchange and OrderedExchange. In RandomExchange, each node randomly alternates between the PAGE (i.e., establishing a

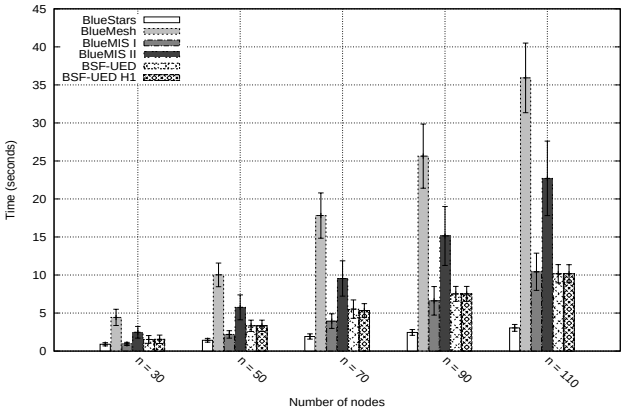


Fig. 3: Comparison of the execution time

connection) and PAGE SCAN (i.e., listening to a connection) states and during this time the node attempts to contact all its neighbors. The alternation is required since a node cannot be in both states at the same time. In OrderedExchange, the communication round is initiated by the nodes which have the largest identifiers among their neighbors. These nodes send messages to all their smaller neighbors (with respect to identifier). If a node receives a message from every larger neighbor, the node starts sending messages to all its smaller neighbors. This guarantees that every edge in the network is visited exactly once. We found that OrderedExchange is significantly faster than RandomExchange in relatively small networks, which is typical to Bluetooth networks, although RandomExchange is theoretically better.

BlueStars and BlueMesh use a maximal independent set algorithm that, by its nature, uses OrderedExchange. BlueStars requires three communication rounds for 1) forming the piconets, 2) identifying the neighbor piconets, and 3) interconnecting the neighbor piconets. On the other hand, each phase of BlueMesh requires four rounds: 1) a communication round to exchange the 1-hop neighborhood relationship between neighbor nodes in order to construct the 2-hop neighborhood relationships, 2) a communication round to form outdegree limited piconets, 3) a communication round for masters to discover their neighboring piconets (i.e. slaves sends information about their neighbors to their masters), and 4) a communication round for each node surviving the current phase to find its neighbors in the next phase (i.e. its neighbors in G_i where i is the number of the next phase). Therefore, BlueStars requires less communication rounds. This explains why BlueStars is fast, whereas BlueMesh is not. We suggest therefore that calculating the number of communication rounds required by a certain algorithm gives a good indication of its empirical execution time.

BlueMIS I originally uses RandomExchange. We implemented it using OrderedExchange. More details can be found in [9]. The OrderedExchange BlueMIS implementation requires two communication rounds (see Appendix A). This is equivalent to BlueStars. Nevertheless, simulation experiments show that BlueStars is faster. The main reason behind this is that, in BlueMIS, a node is required to visit its neighbors in order from the smallest to largest neighbors. This order causes an increased delay, since the probability of contacting a node that is busy is higher, where a busy node is a node that is in communication with another node. For example, assume a network where nodes v_{10} (i.e. with identifier 10) and v_9 share the neighborhood of node v_1. Assume that v_{10} and v_9 have other different smaller neighbors all with identifier larger than v_1. In BlueMIS, both v_{10} and v_9 must first contact v_1, but v_1 can be contacted by one node at the same time. Therefore, one of the nodes v_9 or v_{10} is delayed until v_1 is free. This causes an extended time execution. BlueStars does not have this order. BlueMIS II is analyzed using the same principles mentioned above. A BSF-UED node, on the other hand, requires sometimes an order of contacting its neighbors (such in line 3, `capture()` in Algorithm 2), while in other cases this order is not required. Also, BSF-UED requires only 3 communication rounds, similar to BlueStars (See Appendix A for more details).

6.2 Number and size of piconets

We study the number of piconets in the formed scatternets. This is shown in Figure 4. BlueStars outperforms all the other algorithms in this metric. However, this is a trade-off with the number of outdegree unlimited piconets. We performed the following experiment in order to study this trade-off. Essentially, each master node in BlueStars becomes a slave and each slave node becomes a master. This led to a significant increase in the number of masters in BlueStars and to heuristically limiting the size of BlueStars piconets to about 3 slaves per piconet at the most. BlueMIS I is the worst algorithm in term of the number of piconets. This is because all nodes initially consider themselves masters in BlueMIS I. A node u becomes slave only in one case; which occurs if each slave v of u has larger identifier and has also considered u as a slave. In such case, each slave v becomes the master of u and thus u is left with no slaves. Whence, u become a slave only with no master role. The rules of BlueMIS II significantly decreases the number of piconets of BlueMIS I. We should note, however, that this significant decrease caused the piconets of BlueMIS II to be not limited to 7 slaves. BlueMesh forms scatternets with a logical number of piconets (about 50%). This number increases as the number of nodes increases. BSF-UED and BSF-UED H1 have approximately similar results to BlueMesh. Heuristic H1 causes an increase in the number of piconets as the number of nodes increases.

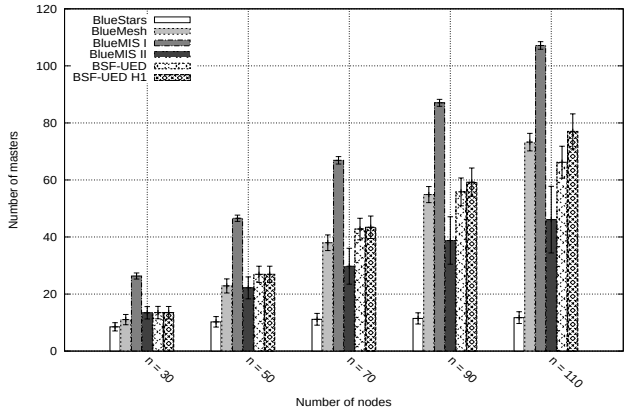


Fig. 4: Comparison of the number of piconets

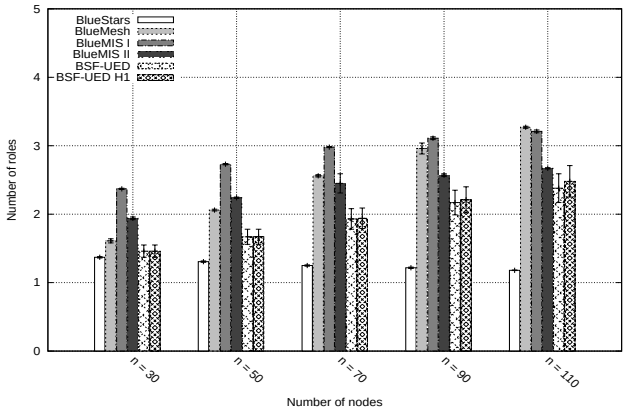


Fig. 5: Comparison of the average role per node

We study the maximum size of piconets. We compute the average maximum outdegree of the formed scatternets. Note that BlueMIS II and BlueStars are the worst algorithms with respect of this metric as shown in Table 1. This is a significant weakness of these algorithms since a piconet with more than 7 slaves introduces a penalty in its throughput and the throughput of the scatternet in general. This is because a master with more than 7 slaves must park all of them except 7 of them. Parked nodes are part of the piconet but do not collaborate in its activities and do not send or receive messages within the piconet. Outdegree limitation is one of the most important quality metrics. This is why it has been the main concern of many BSF algorithms (see Section 1, Section 4 or [11]).

BlueMIS I forms scatternets with the smallest average maximum piconet size. This is because the slaves of a master in BlueMIS I is a maximal independent set of its neighbors, which is of size at most 5 in unit disk graph and in average is less. BlueMesh on average forms scatternets of at most 7. For BSF-UED, the average maximum size of piconet can reach 14.37 slaves. Piconets with such large size are few. For instance, our simulation analysis show that only 4 piconets out of 66 piconets have more than 7 slaves in the case of networks with 110 nodes networks. Table 2 shows that a piconet on average has 2.4 slaves in BSF-UED. When applying the heuristics H1, we find that most scatternets formed are outdegree limited. In fact, in our experiments, we find that only one scatternet (out of 5000 experiments) was outdegree unlimited (with maximal degree 8!). BSF-UED is therefore very close to the optimum, which makes us believe the heuristics could be further improved to achieve this deterministically. Regarding the average piconet size, BSF-UED is only outperformed by BlueMIS I. This is one of the advantages of BSF-UED.

6.3 Average number of roles per node

The number of roles of a node is the number of piconets it belongs to. The average number of roles per node is the sum of number of roles among all nodes divided by the number of nodes. The results are shown in Figure 5. First, note that heuristics H1 of BSF-UED does not change the average number of role per node significantly. We see that BlueStars outperforms all other algorithms in this metrics. The superiority of BlueStars is caused by the small number of piconets, the large size of piconets and the condition that phase 1 forms disjoint piconets. The second best algorithm is BSF-UED.

6.4 Number of bridges

We study in this section the number of bridges and M/S bridges in the formed scatternets. The results are given in Figure 6 and Figure 7. One of the main weaknesses of BSF-UED is the number of M/S bridges. BlueStars and BlueMesh outperforms BSF-UED in this metric. We relate this result to the delegation process followed by BSF-UED. That is, 1) the procedure `FindCommonNeighbors()` and 2) the fact that nodes cannot be captured twice. This delegation process generates a large number of piconets that shall be interconnected. On the other hand, the same delegation process improves the average size of the piconets. Moreover, BSF-UED interconnection rules (see Section 5.3.2) give priorities to rules that generates new M/S bridges. On the other hand, the priorities order of these rules improve the execution time of BSF-UED. For instance, note that a master m, and not one of its slaves, may be involved in the interconnection rule I-Rule 2b and I-Rule 2c. That is, assume that m is waiting for a message from a gateway of a neighbor piconet to be interconnected via one of these rules. In such case, m is blocked until the interconnection occurs. Meanwhile, all slaves of m waits for messages from m that inform

Table 1: Comparison of the average maximum piconet size with standard deviation (in brackets)

Number of nodes	BlueStars	BlueMesh	BlueMIS I	BlueMIS II	BSF-UED	BSF-UED H1
30	8.64 (1.70)	6.90 (0.25)	3.02 (0.31)	8.21 (1.6)	5.99 (0.76)	5.99 (0.76)
50	13.28 (2.38)	7.00 (0)	3.48 (0.49)	12.86 (3.04)	6.39 (0.54)	6.38 (0.52)
70	18.42 (3.55)	7.00 (0)	3.79 (0.40)	18.06 (4.40)	6.94 (1.37)	6.51 (0.49)
90	23.37 (3.76)	7.00 (0)	3.94 (0.23)	22.06 (4.90)	10.16 (2.87)	6.91 (0.27)
110	27.87 (4.09)	7.00 (0)	3.98 (0.18)	27.62 (6.33)	14.56 (3.58)	7 (0.03)

Table 2: Comparison of the average piconet size with standard deviation (in brackets)

Number of nodes	BlueStars	BlueMesh	BlueMIS I	BlueMIS II	BSF-UED	BSF-UED H1
30	3.92 (0.57)	2.36 (0.48)	1.71 (0.12)	3.46 (0.54)	2.28 (0.32)	2.28 (0.320)
50	5.56 (0.97)	3.54 (0.29)	1.93 (0.13)	4.18 (0.75)	2.13 (0.18)	2.13 (0.18)
70	7.13 (1.27)	3.42 (0.52)	2.12 (0.12)	4.99 (1.03)	2.17 (0.18)	2.14 (0.16)
90	8.78 (1.40)	3.78 (0.27)	2.21 (0.12)	5.21 (1.17)	2.52 (0.22)	2.37 (0.15)
110	10.41 (1.73)	3.92 (0.21)	2.31 (0.11)	5.72 (1.40)	2.96 (0.30)	2.53 (0.19)

them to become gateways or not. This causes a higher execution time in general. The high number of M/S bridges is the cost of achieving heuristically outdegree limited scatternets in short execution time which was our main objective when designing BSF-UED. At the same time, we believe that this result is acceptable given that BSF-UED forms piconets with fewer number of slaves on average (see Table 2) and the nodes have less roles on average (see Figure 5). As a result, an M/S bridge can achieve a balance between the role of being a master and the role of being a slave. Also, many of these slaves that are mastered by an M/S bridge will not be significantly affected when their masters works as a slave. That is, let assume that v is an M/S bridge and it has u as its slave and w as its master. When v is active in the piconet of its master w, the slaves of piconet v (including u) are set to inactive in the piconet of v. This is because the master v controls the flow of packets in its piconet. However, because there is a large number of M/S bridges, it is possible that the slave u belongs also to another piconet. Therefore, u is active in a piconet other that of v while its master v is not active in its own piconet.

BlueMIS I does not suffer from a large number of M/S bridges despite the significantly large number of piconets in its scatternets. This is because there are some nodes in the scatternet that have many masters (i.e. very high indegree). Such nodes may cause bottleneck in the scatternet. The involvement of these nodes as bridges to multiple piconets at the same time may delay the transmission of messages between these piconets, causing thus issues in the scatternet throughput.

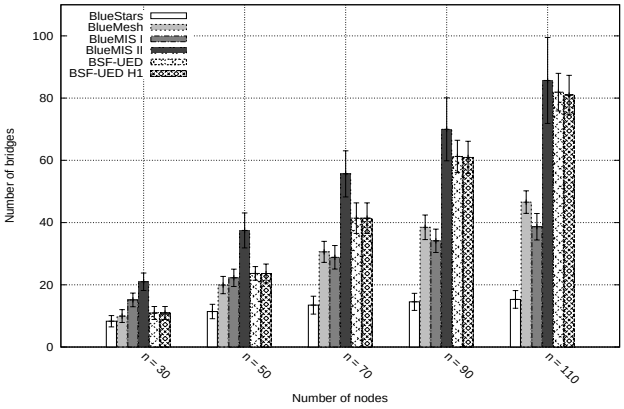


Fig. 6: Comparison of the number bridges

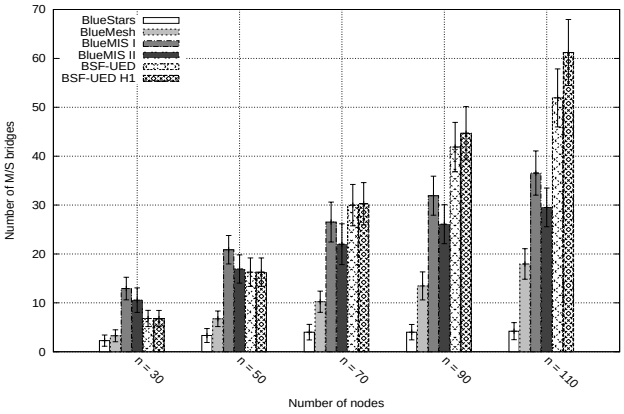


Fig. 7: Comparison of the number of M/S bridges

6.5 Average shortest path

Figure 9 shows the results of the average shortest paths of the formed scatternets. The measure gives an indication of the cost of routing in networks, in general. It is calculated by the minimum number of hops between every pair of nodes on average. In Bluetooth scatternets, the path connecting two nodes is not affected only by the number of hops, but also by the properties of the path nodes (e.g., masters, slaves, bridges, average role per node and number of slaves per master). We see that BlueMIS outperforms the other algorithm in this measure. BSF-UED and BlueStars have ap-

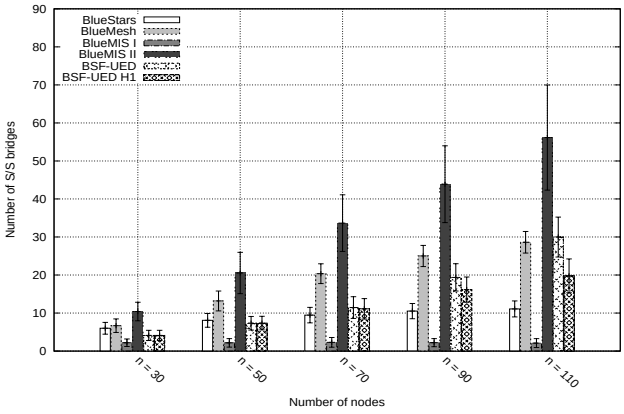


Fig. 8: Comparison of the number of S/S bridges

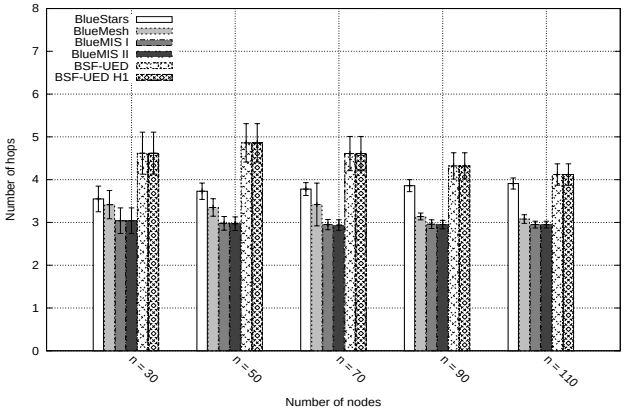


Fig. 9: Comparison of the average shortest path

proximately similar measurements. For BSF-UED, this is a direct result of deleting the unnecessary edges in BSF-UED. We may improve this measure by increasing the number of edges in the scatternet. For instance, masters that do not have their piconets filled with 7 slaves may slave random neighbors. Note however that the difference between BSF-UED and the best algorithm with respect to this metric is about 1 to 1.5 hops on average. This difference is not significant in our opinion.

6.6 Number of messages

We study the number of messages sent by all the network nodes. This is an approximative measure of the energy consumed by the network *during the execution of* the algorithm. Note however that the study of the energy consumption of the algorithms may be more accurate if a detailed energy model of Bluetooth is used. Moreover, this measure is common to asynchronous distributed algorithms. The results are listed in Figure 10. A theoretical analysis shows that the number of messages in each of the algorithms is within a constant to the number of edges of the network. We find

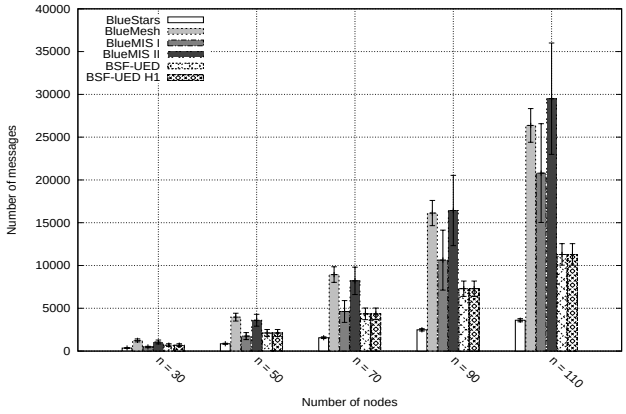


Fig. 10: Comparison of the number of messages sent in total

algorithm BSF-UED significantly outperforms algorithms BlueMesh, BlueMIS I and BlueMIS II. This is one of the advantages of BSF-UED.

7 Conclusion and Future Work

We introduced in this paper a comparative study of some of the major algorithms in the literature. We also introduced algorithm BSF-UED, which is a time-efficient algorithm for the Bluetooth Scatternet Formation problem. The algorithm forms connected scatternets deterministically, and outdegree limited scatternets heuristically. The algorithm can be seen as a modification of BlueStars, BlueMesh and BlueMIS in order to form outdegree limited scatternets. BSF-UED was compared against BlueStars, BlueMesh, BlueMIS I and BlueMIS II. Our algorithm outperformed the other algorithms in a number of important performance metrics. Of particular significance is the fact that BSF-UED outperforms BlueMesh in the average piconet size, average role per node and number of messages; it is also about 3 times faster in execution time. The main weakness of BSF-UED is the high number of M/S bridges. This weakness is tolerable to some degree as we explained in Section 6.4, especially as the average number of slave per piconet and the average roles per node are low.. In term of other metrics, BSF-UED and BlueMesh generate similar results to a certain degree. A future research direction is to solve the issues of BSF-UED, and to adapt it to mobile scenarios.

The comparative study that we performed in this paper shows that it is a challenging task to achieve a perfect balance between the many performance metrics of Bluetooth scatternets. Every algorithm that we studied is found to suffer from at least one weakness with respect to a certain performance metric. In order to solve this issue, we suggest forming scatternets by using simple heuristics. These heuristics shall take into account the execution time as a main priority. This approach leads in general to faster BSF al-

gorithms as it is the case of BlueStars and BSF-UED. We believe that more work should be done in the field of BSF as the results of this comparative study show that many issues shall be addressed despite the numerous studies in the field.

We suggest a change in the specifications of Bluetooth in order to make it more suitable for ad-hoc networking. A change in the link establishment procedures is necessary in order to achieve faster BSF algorithms. We also suggest that the restriction of outdegree limited piconets shall be relaxed. We believe that more focus shall be given to inter-piconet and intra-piconet scheduling algorithms.

The nature of Bluetooth scatternets led to the adoption of quality metrics that are different than those used in non-Bluetooth networks. However, the large number of quality metrics of Bluetooth scatternets complicates the study of the scatternet performance. Therefore, we see that a set of new generalized quality metrics to study the performance of Bluetooth scatternets must be introduced in future work. These quality metrics shall be few in number, be easy to implement, and give a good indication of the scatternet performance [7]. We believe that such generalized quality metrics shall take into consideration the inter-piconet and intra-piconet scheduling in the scatternets. Solving this problem would simplify the design of new BSF algorithms and leads to the introduction of more efficient BSF algorithms.

References

1. Nokia. 5 surprising facts about Bluetooth, May 2011.
2. Bluetooth SIG. *Bluetooth Specifications ver4.0*, 2010.
3. C. Petrioli, S. Basagni, and I. Chlamtac. Configuring BlueStars: Multihop Scatternet Formation for Bluetooth Networks. *IEEE Transactions on Computers*, 52:779–790, 2003.
4. X. Y. Li, I. Stojmenovic, and Y. Wang. Partial Delaunay triangulation and degree limited localized Bluetooth scatternet formation. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):350–361, 2004.
5. Z. Wang, R. K. Thomas, and J. Haas. Performance comparison of Bluetooth scatternet formation protocols for multi-hop networks. *Wireless Networks*, 15(2):209–226, 2009.
6. C. Petrioli, S. Basagni, and I. Chlamtac. BlueMesh: degree-constrained multi-hop scatternet formation for Bluetooth networks. *Mobile Networks and Applications*, 9(1):33–47, 2004.
7. N. Zaguia, Y. Daadaa, and I. Stojmenovic. Simplified bluetooth scatternet formation using maximal independent sets. *Integrated Computer-Aided Engineering*, 15 (3):229–239, 2008. ISSN 1069-2509.
8. F. Howell and R. McNab. SimJava: A discrete event simulation library for java. *Simulation Series*, 30:51–56, 1998.
9. A. Jedda, G.-V. Jourdan, and H.T. Mouftah. Time-efficient algorithms for the outdegree limited bluetooth scatternet formation problem. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC) 2012*, pages 000132–000138, july 2012. doi: 10.1109/ISCC.2012.6249281.
10. A. Jedda, G.-V. Jourdan, and N. Zaguia. Towards better understanding of the behaviour of Bluetooth networks distributed algorithms. *International Journal of Parallel, Emergent and Distributed Systems*, 27(6):563–586, 2012.
11. I. Stojmenovic and N. Zaguia. Bluetooth scatternet formation in ad-hoc wireless networks. In *Chapter 9 in: Performance Modeling and Analysis of Bluetooth Networks: Network Formation, Polling, Scheduling, and Traffic Control (J. Misic and V. Misic)*, pages 147–171. 2006.
12. R. M. Whitaker, L. Hodge, and I. Chlamtac. Bluetooth scatternet formation: A survey. *Ad Hoc Networks*, 3(4):403–450, 2005. ISSN 1570-8705. doi: 10.1016/j.adhoc.2004.02.002.
13. E. Vergetis, R. Guérin, S. Sarkar, and J. Rank. Can Bluetooth succeed as a large-scale ad hoc networking technology? *IEEE Journal on Selected Areas in Communication*, 23(3):644–656, 2005.
14. S. Basagni, R. Bruno, G. Mambrini, and Chiara Petrioli. Comparative performance evaluation of scatternet formation protocols for networks of bluetooth devices. *Wirel. Netw.*, 10(2):197–213, 2004. ISSN 1022-0038. doi: 10.1023/B:WINE.0000013083.41155.fa.
15. W-Z. Song, Y. Wang, C. Ren, and C. Wu. Multi-hop scatternet formation and routing for large scale Bluetooth networks. *International Journal of Ad Hoc and Ubiquitous Computing*, 4(5):251–268, 2009.
16. T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed Topology Construction of Bluetooth Personal Area Network. In *IEEE INFOCOM 2001*, 2001.
17. H. Sreenivas and H. Ali. An evolutionary Bluetooth scatternet formation protocol. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 8–pp. IEEE, 2004.
18. L. E. Hodge, R. M. Whitaker, and S. Hurley. Multiple objective optimization of bluetooth scatternets. *Journal of combinatorial optimization*, 11(1):43–57, 2006.
19. M A. Marsan, C-F. Chiasserini, A. Nucci, G. Carello, and L. De Giovanni. Optimizing the topology of Bluetooth wireless personal area networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 572–579. IEEE,

[7] We thank the anonymous reviewer for directing our attention to this important issue in the field of Bluetooth Scatternet Formation algorithms

2002.
20. L. Huang, H. Chen, T. Sivakumar, T. Kashima, and K. Sezaki. Impact of topology on Bluetooth scatternet. *International Journal of Pervasive Computing and Communications*, 1(2):123–134, 2005.
21. C. Kiss Kalló, C-F. Chiasserini, S. Jung, M. Brunato, and M. Gerla. Hop count based optimization of Bluetooth scatternets. *Ad Hoc Networks*, 5(3):340–359, 2007.
22. A. Daptardar. Meshes and cubes: Distributed scatternet formations for Bluetooth personal area networks, 2004.
23. W. Z. Song, X. Y. Li, Y. Wang, and W. Z. Wang. dBBlue: Low diameter and self-routing Bluetooth scatternet. *Journal of Parallel and Distributed Computing*, 65 (2), 2 2005.
24. H. Zhang, J. C. Hou, and L. Sha. A Bluetooth loop scatternet formation algorithm. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 2, pages 1174–1180. IEEE, 2003.
25. Y. Wang, I. Stojmenovic, and X-Y Li. Bluetooth scatternet formation for single-hop ad hoc networks based on virtual positions. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 1, pages 170–175. IEEE, 2004.
26. L. Barrière, P. Fraigniaud, L. Narayanan, and J. Opatrny. Dynamic construction of bluetooth scatternets of fixed degree and low diameter. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 781–790. Society for Industrial and Applied Mathematics, 2003.
27. C-M Yu and J-H Lin. Enhanced Bluetree: a mesh topology approach forming Bluetooth scatternet. *IET Wireless Sensor Systems*, 2(4):409–415, 2012.
28. F. Cuomo, T. Melodia, and I. F. Akyildiz. Distributed self-healing and variable optimization algoritms for QoS provisioning in scatternets. *IEEE Journal on Selected Areas in Communication*, 22(7):1220–1236, 2004.
29. C. Law, A. K. Mehta, and K. Siu. A new Bluetooth scatternet formation protocol. *Mobile Networks and Applications*, 8(5):485–498, 2003.
30. R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983.
31. G. Tan, A. Miu, J. Guttag, and H. Balakrishnan. An efficient scatternet formation algorithm for dynamic environments. In *Proceedings of the IASTED Communications and Computer Networks (CCN)*, pages 4–6, 2002.
32. T-C. Huang, C-S. Yang, C-C. Huang, and S-W. Bai. Hierarchical Grown Bluetrees (HGB): an effective topology for Bluetooth scatternets. *International Journal of Computational Science and Engineering*, 2(1):23–31, 2006.
33. G. Zaruba, S. Basagni, and I. Chlamtac. BlueTrees - Scatternet formation to enable Bluetooth-based personal area networks. In *Proceedings of the IEEE International Conference on Communications*, volume 1, pages 273–277, 6 2001.
34. Y. Dong and J. Wu. Three Bluetree formations for constructing efficient scatternets in Bluetooth. In *Proc. of the 7th Joint Conference on Information Sciences*, pages 385–388, 2003.
35. E. Pagani, G. Rossi, and S. Tebaldi. An on-demand Bluetooth scatternet formation algorithm. *Wireless On-Demand Network Systems*, pages 51–61, 2004.
36. M. Methfessel, S. Peter, and S. Lange. Bluetooth Scatternet Tree Formation for Wireless Sensor Networks. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 789–794. IEEE, 2011.
37. D. Dubhashi, O. Häggström, G. Mambrini, A. Panconesi, and C. Petrioli. Blue pleiades, a new solution for device discovery and scatternet formation in multihop bluetooth networks. *Wireless Networks*, 13:107–125, January 2007. ISSN 1022-0038.
38. R. Wattenhofer and A. Zollinger. XTC: a practical topology control algorithm for ad-hoc networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 216, April 2004.
39. Q. Wang. *Scheduling and Simulation of Large Scale Wireless Personal Area Networks*. PhD thesis, University of Cincinnati, Cincinnati, USA, 2006.
40. VINT. The VINT Project, The Network Simulator - ns-2, URL: http://www.isi.edu/nsnam/ns/. Page accessed on (9-7-2013). In *Zealand: University of Otago*, pages 181–186, 2003.

A Appendix 1: Time complexity analysis

In this section, we study the time complexity of BlueStars, BlueMesh, BlueMIS and BSF-UED. We start with BlueStars.

Theorem 6 *The time complexity of BlueStars is $O(n)$ where n is the number of nodes in the network.*

Proof The most expensive procedure in BlueStars is the procedure of forming disjoint piconets (i.e. the first phase). A node v_i with identifier i starts executing this phase if all its larger neighbors (i.e. with larger identifiers) sent a message to v_i. We say v_i waits for v_j if v_j is a larger neighbor of v_i. According to this definition, v_i may wait for v_k where v_k is a larger neighbor of v_j. We may construct thus a chain $v_1, ,v_n$ of length n such that v_i waits for v_j if $i > j$. Therefore, a node v_i waits for all the larger nodes in the network, and respectively the smallest node v_1 waits for all the other nodes. Therefore, the time complexity of this phase is $O(n)$. In the second phase of BlueStars, there is a message exchange between 1) the slaves and masters (in order for the masters to identify the neighbor piconets), 2) the masters and gateways

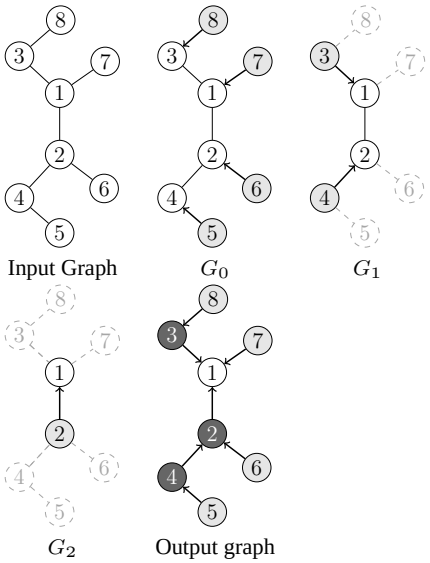


Fig. 11: An example scenario with Bluemesh.

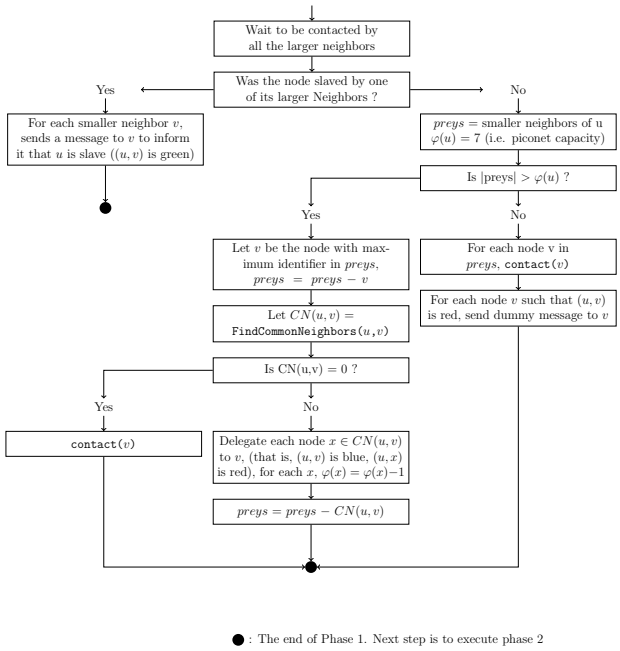


Fig. 12: Flow diagrams of Phase 1 of BSF-UED

(to inform the gateways which piconets they must interconnect), and 3) the gateways and neighbor gateways (to interconnect the neighbor piconets). This requires $O(1)$ time complexity, and therefore the time complexity of BlueStars is $O(n)$.

BlueMesh runs in iterations (as explained in Section 4.4). Each iteration is similar with respect to time complexity to BlueStars, since both constructs a maximal independent set in a similar manner. Therefore, each iteration requires $O(n)$. In the following, we analyzes the worst case number of iterations of BlueMesh.

Theorem 7 *The number of iterations run by BlueMesh is in the worst case $O(\log n)$ in arbitrary graphs and $O(1)$ in unit disk graphs, where n is the number of nodes in the network.*

Proof We build the worst case scenario as follows. We start with the simple graph of two nodes u and v linked by the edge (u, v). Let's assume that nodes u and v are the last surviving nodes in iteration k, where k is the index of the last phase of the algorithm. Note that if a node u survived iteration $k-1$ and moved to iteration k, then it must have a larger neighbor u'. This means that there is at least 4 nodes u, u', v and v' in iteration $k-1$. Therefore, the maximum number of nodes that move to iteration i is $|P_{i-1}|/2$ where $|P_i|$ is the number of nodes in iteration i. Therefore, the maximum number of phases is at $O(\log n)$. The worst case scenario of BlueMesh is shown in Figure 11.

Note that if a node has more than 5 neighbors in a unit disk graph then at least two of them are also neighbors. Following the previous argument, if a node survived k iterations then it must have at least k largest neighbors that are not neighbors to each other. This means that k is at most 5 in unit disk graphs. Therefore, the maximum number of BlueMesh iterations if run over unit disk graphs is $O(1)$.

The result of Theorem 7 matches simulation results in this paper and in [6], whereas a theoretical analysis of this aspect of BlueMesh is first studied here.

Therefore, the time complexity of BlueMesh in unit disk graphs is $O(n)$. In arbitrary graphs it is in the worst case $O(n \log n)$.

BlueMIS I complexity is straightforward. Each node exchanges its neighbors list with all its neighbors in a first round. In a second round, each node u sends to all its neighbors the set $S(u)$ which is the set of nodes that u may be master to. Therefore, the time complexity of BlueMIS I is $O(1)$. Thus, BlueMIS I is a local distributed algorithm. Local distributed algorithms has the advantage that their execution time does not depends on the size of the input network, making them suitable to solve scalability issues. The case is different in BlueMIS II. To achieve the exact same results given in [7], BlueMIS II time complexity may be at least in $O(n)$. This is because, in order to avoid certain worst-case conflicting scenarios, each node must execute the rules of BlueMIS II while every other is waiting (that is, in a sequential manner). This is one of the major issues in BlueMIS II. In our implementation, every node executes its rules locally after collecting sufficient neighborhood information. Therefore, we assumed that the time complexity of BlueMIS II remains $O(1)$.

BSF-UED time complexity is similar to that of BlueStars. The first phase is similar to the first phase of BlueStars and thus has $O(n)$ time complexity. In the second phase of BSF-UED, there is a communication exchange between, 1) slaves and masters, 2) masters and gateways, and 3) gateways and gateways from neighbor piconets. Therefore, the time complexity of BSF-UED is $O(n)$.

B Appendix 2: Flow Diagrams of BSF-UED

In this appendix section, we present the flow diagrams of our algorithm BSF-UED without the heuristics. We believe that this simplifies the understanding of our algorithm and its pseudocode.

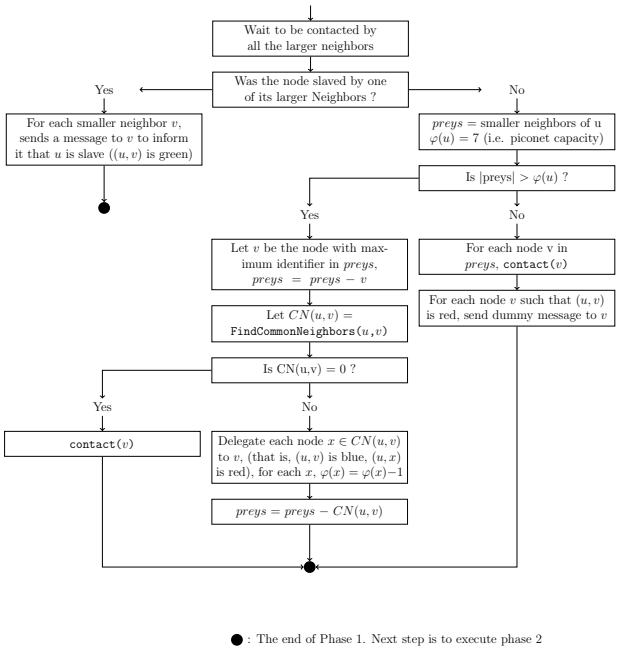


Fig. 13: Flow diagrams of Phase 2 of BSF-UED