dBBlue: Low Diameter and Self-routing Bluetooth Scatternet

Wen-Zhan Song* Xiang-Yang Li* Yu Wang† Weizhao Wang*

*Dept. of Computer Science, Illinois Institute of Technology, 10 W. 31st Street, Chicago, IL 60616. Email songwen@iit.edu, xli@cs.iit.edu, wangwei4@iit.edu. The work of the second author is partially supported by NSF CCR-0311174.

†Dept. of Computer Science, University of North Carolina at Charlotte. Email: wangyu@ieee.org

Abstract

This paper addresses the problem of scatternet formation for single-hop Bluetooth based wireless ad hoc networks, with minimal communication overhead. We adopt the well-known structure *de Bruijn graph* to form the backbone of Bluetooth scatternet, hereafter called *dBBlue*, such that every master node has at most seven slaves, every slave node is in at most two piconets, and no node assumes both master and slave roles. Our structure *dBBlue* also enjoys a nice routing property: the diameter of the graph is $O(\log n)$ and we can find a path with at most $O(\log n)$ hops for every pair of nodes without any routing table . Moreover, the network congestion is at most $O(\log n/n)$, assuming that a unit of total traffic demand is equally distributed among all pair of nodes. We discuss in detail a vigorous method to *locally* update the structure *dBBlue* using at most $O(\log n)$ communications when a node joins or leaves the network. In most cases, the cost of updating the scatternet is actually $O(1)$ since a node can join or leave without affecting the remaining scatternet. The number of affected nodes is always bounded from above by a constant when a node joins or leaves the network. The dBBlue scatternet can be constructed incrementally when the nodes join the network one by one. To facilitate the self-routing and easy updating, a scalable MAC assignment mechanism is designed to guarantee the packet delivery even during scatternet updating. In addition, the structure formed by our method can sustain the faults of 2 nodes and the network is still guaranteed to be connected. If a node detects a fault of some neighboring master node or bridge slave node, it can dynamically re-route the packets and the path traveled by the packet is still at most $O(\log n)$ hops. Previously no method can guarantee all these properties although some methods can achieve some of the properties.

Keywords: Bluetooth networks, scatternet formation, de Bruijn graph, single-hop, low diameter, self-routing, scalable MAC assignment, fault tolerant, scheduling.

I. INTRODUCTION

Bluetooth [7] is a promising new wireless technology, which enables portable devices to form short-range wireless ad hoc networks based on a frequency hopping physical layer. Bluetooth ad-hoc networking presents some technical challenges, such as scheduling, network forming and routing. User mobility poses additional challenges for connection rerouting and QoS services. It has been widely predicted that Bluetooth will be a major technology for short range wireless networks and wireless personal area networks. This paper deals with the problem of building ad hoc networks using Bluetooth technology.

According to the Bluetooth standard, when two Bluetooth devices come into each other's communication range, one of them assumes the role of *master* of the communication and the other becomes the *slave*. This simple one hop network is called a *piconet*, and may include more slaves. The network topology resulted by the connection of several piconets is called a *scatternet*. There is no limit on the maximum number of slaves connected to one master, although the number of active slaves at one time cannot exceed 7. If a master node has more than 7 slaves, some slaves must be parked. To communicate with a parked slave, a master has to *unpark* it, thus possibly parking another active slave instead. The standard also allows multiple roles for the same device. A node can be master in one piconet and a slave in one or more other piconets. However, one node can be active only in one piconet. To operate as a member of another piconet, a node has to switch to the hopping frequency sequence of the other piconet. Since each switch causes delay (e.g., scheduling and synchronization time), an efficient scatternet formation protocol can be the one that minimizes the roles assigned to the nodes, without losing network connectivity.

While several solutions and commercial products have been introduced for one-hop Bluetooth communication, the Bluetooth specification does not indicate any method for scatternet formation. The problem of scatternet formation has not been dealt with until very recently. The solutions proposed in the literature can be divided into single-hop and multi-hop solutions. Several criteria could be set as the objectives in forming scatternet. First of all, the protocol should create degree limited scatternets, to avoid parking any slave node. Secondly, the number of piconets should be minimized to reduce the inter-piconet scheduling and communication cost. Thirdly, the formation and maintenance of scatternet should have small communication overhead. Fourthly, the diameter of the scatternet should be small, i.e., the maximum number of hops between any two devices must be small to provide faster routing. In this paper, we focus on scatternet formation for single-hop ad hoc networks. In a single-hop ad hoc network, all wireless devices are in the radio vicinity of each other, e.g., electronic devices in a laboratory, or laptops in a conference room. A single-hop network can be modeled by a complete graph.

Previous literature on scatternet formation assumed that devices are not able to communicate unless they have previously discovered each other by synchronizing their frequency hopping patterns. Thus, even if all nodes are within direct communication range of each other, only those nodes, which are synchronized with the transmitter, can hear the transmission. Synchronizing the frequency hopping patterns is apparently a time consuming and pseudo-random process [12]. In this paper we assume that the problem of discovering all neighbors within transmission radius of a device is resolved by separate Bluetooth protocol. One such protocol for discovering all one hop networks is described in [12], [3], while a protocol that provides two-hop information to every node is described in [11]. These protocols are applicable as the pre-phase of our scheme.

This paper addresses the problem of scatternet formation for single-hop Bluetooth based ad hoc networks, with minimal communication overhead. We adopt the well-known structure *de Bruijn graph* to form the backbone of Bluetooth scatternet, hereafter called *dBBlue*, such that every master node has at most seven slaves, every slave node is in at most two piconets, and *no* node assumes both master and slave roles. Our structure *dBBlue* also enjoys a nice routing property: the diameter of the graph is $O(\log n)$ and we can find a path with at most $O(\log n)$ hops between every pair of nodes without any routing table. Moreover, the network congestion is at most $O(\log n/n)$, assuming that a unit total traffic demand is evenly distributed among all pair of nodes. We discuss in detail a vigorous method to *locally* update the structure *dBBlue* using at most $O(\log n)$ communications when a node joins or leaves the network. In most cases, the cost of updating the scatternet is actually $O(1)$ since a node can join or leave without affecting the remaining scatternet. The number of affected nodes is always bounded from above by a constant when a node joins or leaves the network. To facilitate self-routing and easy updating, we design a scalable MAC assigning mechanism for piconet, which can guarantee the packet delivery even during updating. Our method can construct the structure *dBBlue* incrementally when the nodes join the network one by one. In addition, the structure formed by our method can sustain the faults of 2 nodes and the network is still guaranteed to be connected. If a node detects a fault of some neighboring master node or bridge slave node, it can dynamically re-route the packets and the path traveled by the packet is still at most $O(\log n)$ hops. Previously no method can guarantee all these properties although some methods can achieve some of the properties.

The rest of the paper is organized as follows. Section II presents our new Bluetooth formation algorithms for single-hop ad hoc networks. We describe how to build a static scatternet of n nodes based on de Bruijn graph and assign roles and labels to them. Section III proposes a vigorous method to *locally* and

dynamically update the scatternet topology when a node joins or leaves the network. Section IV describes the routing method for our de Bruijn based scatternet which efficiently finds the next node to go without any routing table. The related works are discussed in section V. We conclude our paper in Section VI.

II. dBBlue Scatternet Construction

A. de Bruijn Graph

Our dBBlue scatternet first builds a backbone based on the well-known de Bruijn graph [5]. The de Bruijn graph, denoted by $B(d,k)$, is a directed graph with d^k nodes. Assume that each node is assigned a unique label of length k on the alphabet $\{0,\cdots,d-1\}$. There is an edge in $B(d,k)$ from a node with label $x_1x_2\cdots x_k$ to any node with label $x_2\cdots x_ky$, where $y\in\{0,\cdots,d-1\}$. Figure 1 illustrates $B(2,3)$. It is well-known that the de Bruijn graph enables self-routing intrinsically. The self-routing path from a source node with a label $x_1x_2\cdots x_k$ to the target node with a label $y_1y_2\cdots y_k$ is $x_1x_2\cdots x_k \rightarrow x_2\cdots x_ky_1 \rightarrow x_3\cdots x_ky_1y_2 \rightarrow \cdots \rightarrow x_ky_1\cdots y_{k-1} \rightarrow y_1\cdots y_k$. Observe that, we could find a shorter route by looking for the longest sequence that is both a suffix of $x_1x_2\cdots x_k$ and a prefix of $y_1y_2\cdots y_k$. Suppose that $x_i\cdots x_k = y_1\cdots y_{k-i+1}$ is such longest sequence. The shortest path between the source and the target is $x_1\cdots x_k \rightarrow x_2\cdots x_ky_{k-i+2} \rightarrow \cdots \rightarrow x_{i-1}\cdots x_ky_{k-i+2}\cdots y_{k-1} \rightarrow y_1\cdots y_k$. Clearly, the route between any two nodes is at most k hops, i.e., $B(d,k)$ has diameter $k=\log_d n$, where $n=d^k$ is the number of nodes of the graph.

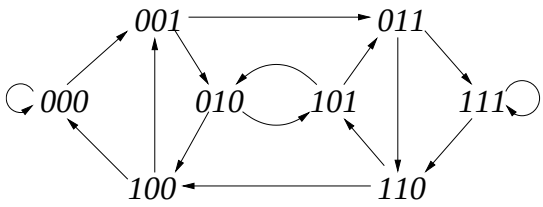


Fig. 1. The de Bruijn graph $B(2,3)$.

The classical de Bruijn graph is *balanced* in the sense that the labels of all nodes have the same length. The de Bruijn graph can be generalized to any set of vertices whose labels form a universal prefix set. In [6], Fraigniaud and Gauron proposed a novel method to construct an efficient topology for P2P network based on the generalized de Bruijn graph defined on a universal prefix set. *"A universal prefix set is a set S of labels on an alphabet Σ such that, for any infinite word $w\in\Sigma^\star$, there is a* unique *word in S, which is a prefix of w. The empty set is also a universal prefix set."*[6] For instance, $\{00,01,100,101,110,111\}$ is a universal prefix set on alphabet $\Sigma=\{0,1\}$, but $\{00,01,10\}$ and $\{00,01,100,1000,101,110,111\}$ are not. There is a directed edge from node $u=x_1x_2\cdots x_k$ to another node v in the generalized de

Bruijn graph if $x_2 \cdots x_k$ is the prefix of the label of node v. We denote a generalized de Bruijn graph as *pseudo-balanced* if the lengths of the node labels are different by at most one, which includes the *balanced* de Bruijn graph. For simplicity, we still denote a pseudo-balanced de Bruijn graph on alphabet $\{0, 1\}$ by $B(2, k)$ if the node labels have length at least k bits and at most $k + 1$ bits. We also say that a node from $B(2, k)$ is at level k if its label has k bits.

In this paper, we only consider the *pseudo-balanced* binary de Bruin graph $B(2, m)$. Node labels in a *pseudo-balanced* de Bruijn graph correspond to all the leaf nodes in a *full* binary tree, in which the depth difference between any two leaf nodes is at most one and each internal node has two children, Figure 2 illustrates the correspondence between them. In the figure, the pseudo-balanced de Bruijn graph is defined on the leaf nodes and directed edges.

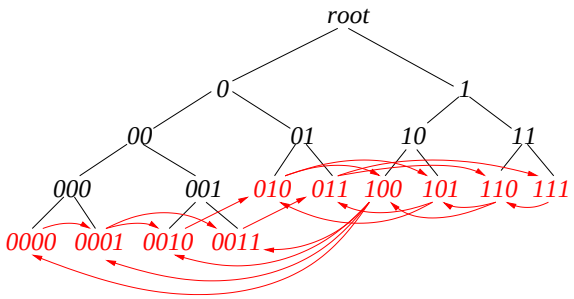


Fig. 2. The correspondence between full binary tree and pseudo-balanced de Bruijn graph.

In a pseudo-balanced de Bruijn graph $B(2, k)$, each node has at most 4 out-neighbors and 2 in-neighbors. To route a packet from a node u with label $x_1x_2 \cdots x_{s-1}x_s$ to another node v with label $y_1y_2 \cdots y_{t-1}y_t$, where $s, t \in [k, k + 1]$. Node u will forward the packet to its neighbor node with label $x_2 \cdots x_{s-1}x_s$, or $x_2 \cdots x_{s-1}x_sy_1$, or $x_2 \cdots x_{s-1}x_sy_1y_2$, whichever exits. Notice that since the labels of the nodes form a universal prefix set, we know that *exactly* one of these three labels does exist. The following nodes keep forwarding the packet similarly until it reaches node v. Consequently, the diameter of pseudo-balanced de Bruijn graph is still $O(\log n)$. In this paper, we propose a scalable scatternet structure based on pseudo-balanced de Bruijn graph $B(2, k)$.

In a pseudo-balanced de Bruijn graph $B(2, m)$, two nodes are called *critical pair* if they only differ in the least significant bit of their labels. Let $u_0, u_1, \cdots, u_p$ be the sequence of nodes visited by a traversal of all leaf nodes in the corresponding binary tree of $B(2, m)$. A node u_i is called the *successor* of another node u_{i-1} and u_{i-1} is called the *predecessor* of another node u_i. Here $i - 1$ takes value $(i - 1) \mod (p + 1)$. For example, in Figure 2, nodes 0010 and 0011 is a critical pair; node 010 is the successor of the node 0011.

B. MAC Address Assignment for Piconet

Our dBBlue protocol always maintains a *pseudo-balanced* de Bruijn graph $B(2, m)$ as the backbone of the network. Here the choosing of the integer m will be discussed later.

Every node in the backbone of dBBlue scatternet will be assigned a *master* role. We will add a *bridge slave* node for every pair of master nodes that are connected in the backbone. Thus, every master node will bring at most *six* bridge slave nodes so far since every node in a pseudo-balanced $B(2, m)$ has at most 6 edges incident.

We then add some free slave nodes to each master node, and call them *pure slave* nodes.

Before we discuss in detail our scatternet construction methods, we present our novel rule of assigning the MAC address in a piconet. In our dBBlue scatternet, when we route a packet to a destination node v, we only know the piconet ID of the node v, say $y_1y_2 \cdots y_k$, which is same as the label of its master node, and the MAC address, say $z_1z_2z_3$, of this node in that piconet. The detail routing mechanism will be discussed in Section IV. When some node joins or leaves the scatternet, we often have to reorganize some piconets and thus re-assign the MACs of some nodes. Our method of assigning MAC addresses in a piconet and reorganizing the piconets guarantees that the new piconet (even the new MAC address) can be found by a simple appending or deleting the least significant bit, which keeps the label prefix of updated nodes unchanged so that the delivery of the packets on the way to those updated nodes will not be interrupted.

In a piconet, MAC address 000 is always reserved for the master node. For simplicity, we omit the MAC address of a master node hereafter while representing its label, i.e., the master node with label $x_1x_2 \cdots x_{m-1}x_m$ actually has a label $(x_1x_2 \cdots x_{m-1}x_m, 000)$ if consistent labels with slave nodes are needed. Remember that, in a pseudo-balanced de Bruijn graph, any node has 2 in-neighbors and at most 4 out-neighbors, so MAC addresses 011 and 111 are always reserved for the two bridge slaves connecting to in-neighbors, MAC 010, 101, 001 and 110 are reserved for bridge slaves to connecting out-neighbors if they exist, and 100 is reserved for the 7th slave (it must be a pure slave) if it exists. Figure 3 illustrates all four possibilities for the piconet MAC address assignment according to the number of out-neighbors in scatternet backbone. In the figure, for simplicity, we use $y_1y_2 \cdots y_{m-1}y_m(y)$ to denote a node with label $y_1y_2 \cdots y_{m-1}y_m$ or $y_1y_2 \cdots y_{m-1}y_my$, whichever exists in the network. Notice that a master node in the constructed scatternet based on a pseudo-balanced de Bruijn graph $B(2, m)$ always has two incoming neighbors. For example, a master node $x_1x_2 \cdots x_m$ in level m can have incoming neighbor $0x_1x_2 \cdots x_{m-1}$ or $0x_1x_2 \cdots x_m$, but not both since the de Bruijn graph is built upon a universal

prefix set; similarly another incoming neighbor is $1x_1x_2\cdots x_{m-1}(x_m)$. Analogously, a master node $x_1x_2\cdots x_mx_{m+1}$ in level $m+1$ has incoming neighbors $0x_1x_2\cdots x_{m-1}(x_m)$ and $1x_1x_2\cdots x_{m-1}(x_m)$. On the other hand, the number of out-neighbors of a node in the pseudo-balanced de Bruijn graph $B(2,m)$ could be $1,2,3,4$. Notice that only the nodes at level m could have 3 or 4 out-neighbors and those at level $m+1$ could have 1 out-neighbor.

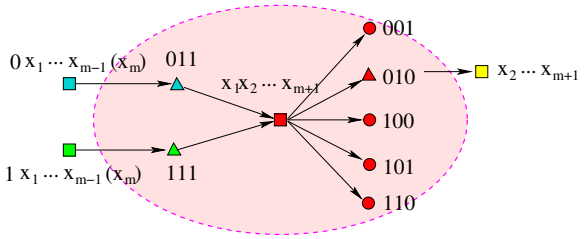
Table I summarizes the rule of assigning the MAC address to the bridge slave nodes in a piconet. Their MAC addresses can be calculated uniquely according to the label bit difference between current piconet and neighboring piconet IDs. For example, if the master u is labeled $x_1x_2\cdots x_s$ and its out-neighbor v is labeled $x_2\cdots x_sy_1y_2$, then the MAC addresses of their bridge slave is $y_1y_2\overline{y_2}$ assigned by u, and x_111 assigned by v. Remember that every bridge slave has two MAC addresses: one MAC address in each of the two piconets it resides.

TABLE I

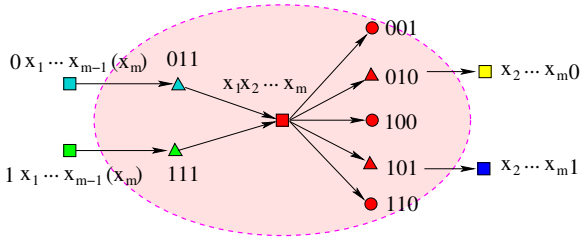
THE RULE TO ASSIGN MAC ADDRESS TO BRIDGE SLAVE NODES.

Node	In-Neighbor	Out-Neighbor		
	$yx_1\cdots x_r$	$x_2\cdots x_s$	$x_2\cdots x_sy_1$	$x_2\cdots x_sy_1y_2$
$x_1\cdots x_s$	$y11$	010	$y_1\overline{y_1}y_1$	$y_1y_2\overline{y_2}$

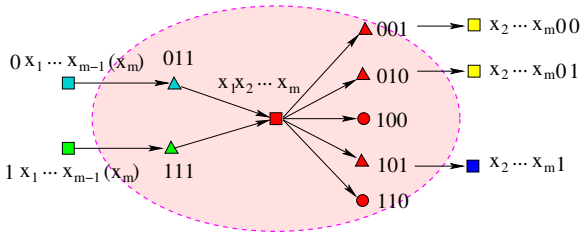
Notice that, in bluetooth scatternet, the bridge slave nodes have two independent piconet IDs and MAC addresses in two piconets respectively. However, since the routing mechanism in de Bruijn is directional, only their piconet ID and MAC address assigned by their in-master is public and meaningful for routing, saying *label* in the remaining paper, and the other one is only used for intra-piconet communication. Figure 4 illustrates one piconet in the scatternet. Here nodes u, I_1, I_2, O_1 and O_2 assume master roles and form the backbone for scatternet. These master nodes are connected in the de Bruijn graph by bridge slaves v_3, v_7, v_2 and v_5 respectively. Assume that node u has a label $x_1x_2\cdots x_{m-1}x_m$. Nodes I_1, I_2 denote the two incoming neighbors of node u, with label $0x_1x_2\cdots x_{m-1}$ and $1x_1x_2\cdots x_{m-1}$ respectively. Nodes O_1, O_2 denote the two outgoing neighbors of node u, with label $x_2\cdots x_{m-1}x_m0$ and $x_2\cdots x_{m-1}x_m1$ respectively. Nodes v_1, v_4, and v_6 are the pure slave nodes of u in the scatternet. The *label* of node v_i ($i\in\{1,2,4,5,6\}$) is $(x_1x_2\cdots x_{m-1}x_m,i)$, where i is the MAC address of node v_i in this piconet, and v_3 and v_7 has public *label* $(0x_1x_2\cdots x_{m-1},x_m\overline{x_m}x_m)$ and $(1x_1x_2\cdots x_{m-1},x_m\overline{x_m}x_m)$, respectively, which is consistent with the prefix of I_1 and I_2 respectively. Notice that the MACs of v_3 and v_7 in the piconet mastered by node u are 3 and 7 respectively, which are used only by nodes in this



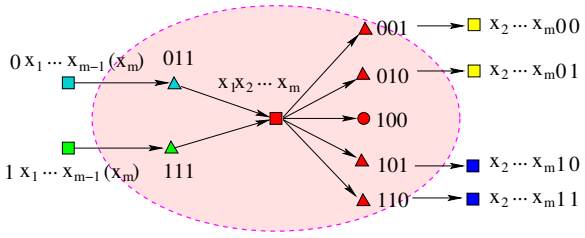
(a) One out-neighbor



(b) Two out-neighbors



(c) Three out-neighbors



(d) Four out-neighbors

Fig. 3. MAC address assignment for a piconet. Here a master node is denoted by a square, a pure slave is denoted by a circle, and a bridge slave is denoted by a triangle.

piconet and not broadcasted to the network.

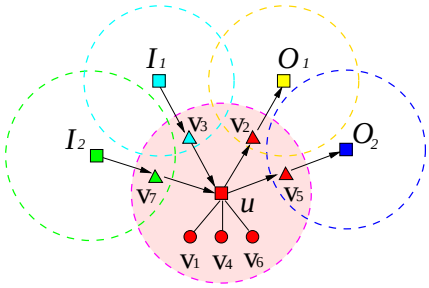


Fig. 4. An example of a piconet (with nodes inside the shaded region) formed by our method.

As will see in Section III, our labeling rule makes the updating of the scatternet topology and nodes' labels much easier when new nodes join the network or some existing nodes leave the network. For an incremental updating of the scatternet, there are two scenarios when a new node joins the network. The first case is that there is a master node who has a *free* slot for a pure slave. We then directly assign the newly joined node as the pure slave of that master node. The second case is that no master node has free slot for a pure slave. We then have to split some piconet and in turn create some free slots for pure slaves. The splitting of a piconet is performed such that the resulting backbone (formed by master nodes and bridge slaves) is still a pseudo-balanced de Bruijn graph. When a piconet is splitted (or two piconets are merged), the labels of some nodes have to be updated. While updating the topology, it is possible that some packets are already on their way to the destinations (via or toward this splitting piconet). Our labeling rule makes sure that the packets can still be routed without any interruption because the target can still be reached by the old label according to our prefix-based routing mechanism. Notice that only the local nodes are assigned new labels, and the re-labeling is also conducted locally. More details will be addressed in Section IV. Hence, our MAC assignment facilitates the self-routing mechanism and provides the resilience even during topology update.

Moreover, because each bridge node guarantees to be assigned different MACs in different piconets, the inter-piconet scheduling in dBBlue scatternet becomes easier especially when the Bluetooth system runs the synchronous services, i.e. the physical link is synchronous connection-oriented(*SCO*)[7]. In *SCO* mode, synchronizing the clocks of all *masters* nodes could solve the scheduling conflict problem on *bridges*, since here the MAC address of each slave is used as its time slot in the corresponding piconet.

C. Static Scatternet Construction

Given n nodes currently distributed in the network, this section gives an efficient algorithm to construct our de Bruijn based scatternet *dBBlue*, which has a low diameter and a bounded node degree property. In other words, we first study the construction of the scatternet for a static n-nodes network, which will serve as the base for our dynamic construction.

Our method will construct a balanced de Bruijn graph $B(2, m)$ as the initial backbone of the network. We will choose integer m such that $2^{m-1} < \lceil \frac{n}{6} \rceil \leq 2^m$. The choosing of m guarantees that there are enough bridge slave nodes, which implies that no master node serves as a bridge slave.

Our method does not consider the detail of the neighbor discovering process. We assume that every node already knows the existence of the other nodes.

Algorithm 1: Static DeBruijn-Based Scatternet

1. Assume that there is a leader already among these n nodes S. The leader could be the node with the smallest ID. We give the *token* to the leader and call it token node. The token node randomly selects 2^m nodes (including itself) into the master set M which assume the *master* roles in final scatternet topology, where $2^{m-1} < \lceil \frac{n}{6} \rceil \leq 2^m$ and n is the number of nodes in S. Let $r = n - 3 \cdot 2^m$, which is the total number of nodes that can be assigned as pure slaves.

2. The token node assigns itself a label 0^m, and each node in M a unique m bits label in the range from $0 \cdots 01$ to $1 \cdots 11$. The set of nodes M forms a de Bruijn graph $B(2, m)$ as the scatternet backbone.

3. The token node, with label $x_1 \cdots x_m$, selects 2 nodes [1] from the remaining nodes as its bridge slaves, and assigns them labels $(x_1 \cdots x_m, 010)$ and $(x_1 \cdots x_m, 101)$ respectively. Here 010, 101 will also serve as the Medium Access Code (MAC) for these two slaves in the piconet mastered by this token node.
The token node uses its bridge slave node $(x_1 \cdots x_m, 010)$ to connect with its out-neighbor $x_2 x_3 \cdots x_m 0$ and the bridge slave node $(x_1 \cdots x_m, 101)$ to connect the out-neighbor node $x_2 x_3 \cdots x_m 1$.

4. Then the token node selects $t = \min\{3, r\}$ nodes [2] from the remaining as its slaves and assigns them with labels $(x_1 \cdots x_m, 001)$, $(x_1 \cdots x_m, 100)$ and $(x_1 \cdots x_m, 110)$ in the order if they exist. Let $r = r - t$. After that, it passes the *token* to its successor.

5. Repeat the above steps (3) and (4) until all nodes in M are processed or no more nodes left. After all nodes have been processed, the current token node passes the *token* back to node 0^m again.

Once the initial topology construction is finished, the token node t will be responsible for the following

[1] There are two special nodes 0^m and 1^m, which only have 1 out-neighbor, we then just use one bridge slave node to connect with its out-neighbor.

[2] Node 0^m and 1^m may choose 5 nodes as its pure slaves since they only have one in-neighbor and one out-neighbor.

node joining and leaving issues. Master nodes form the backbone of bluetooth scatternet, and a piconet works like a node in de Bruijn graph.

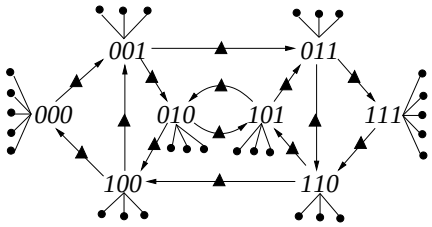


Fig. 5. dBBlue Bluetooth Scatternet.

Figure 5 illustrates a dBBlue scatternet containing 48 nodes based on $B(2,3)$ graph.

Theorem 1: In dBBlue scatternet, each master has no more than 7 slaves and each slave works as a bridge for at most 2 piconets. And the number of piconets is at most $\lfloor \frac{n}{3} \rfloor$ and at least $\lceil \frac{n}{6} \rceil$. Moreover, the computation cost is $O(n)$ for static construction.

Proof. From the topology construction, each master carries at most 5 same prefix slaves, and 2 different prefix slaves since each node in $B(2,m)$ graph has at most 2 in-neighbors, so each master has no more than 7 slaves. And, each slave exists as a free slave or as the bridge between its same prefix master u and one of u's out-neighbors, so the degree of a slave node is at most 2.

Let $n = 6a - b$, where $b \in [0,5]$ and 2^m is the number of masters. Then $2^{m-1} < \lceil \frac{n}{6} \rceil \leq 2^m$ implies $2^{m-1} + 1 \leq a \leq 2^m$. Thus, $n = 6a - b \geq 6(2^{m-1}+1) - b$ and $n = 6a - b \leq 6 \cdot 2^m - b$. Consequently, $3 \cdot 2^m + (6-b) \leq n \leq 6 \cdot 2^m - b$, which implies $\lceil \frac{n}{6} \rceil \leq 2^m \leq \lfloor \frac{n}{3} \rfloor$.

It is obvious that the total computation cost of constructing static dBBlue scatternet is $O(n)$.

In this paper we always assume that a bluetooth piconet consists of at most 7 slaves and 1 master. If future bluetooth technology allows a master to bring more slaves, say p, our scatternet construction method can adapt easily as follows. The scatternet backbone will be still based on $B(2,m)$ de Bruijn graph. However, m is chosen such that $2^{m-1} < \lceil \frac{n}{p-1} \rceil \leq 2^m$. In other words, every master node will carry $p-4$ pure slaves and 4 bridge slaves to connect to its two out-neighbors and two in-neighbors in the de Bruijn graph $B(2,m)$. It is not difficult to show that using de Bruijn graph $B(2,m)$ will create a scatternet with less piconets than using $B(d,m')$ for $d > 2$ since each master node will carry less pure slaves in the later case. On the other hand, the scatternet based on $B(d,m')$ for $d > 2$ does provide a better fault tolerance since the degree of each master node is increased to $2d$.

III. Dynamic Scatternet Updating

In this section we describe a vigorous method to *locally* update the scatternet topology dynamically when some node joins or leaves the network. Considering each piconet as an abstract node in the de Bruijn graph, our goal is to maintain a scalable pseudo-balanced de Bruijn graph.

A. Token Based Updating

First consider the case when a node wants to join the network. We have to assign a role for this newly joined node. There are several possible scenarios about the existing scatternet: (1) the existing scatternet has a master node that has free slave slots, then we can simply assign this newly joined node as the pure slave of this master node; (2) all master nodes in the existing scatternet already have 7 slaves, we then have to expand the backbone of the scatternet to incorporate this newly joined node. In other words, we have to split some piconet to two such that the two new piconets will have some free pure slave slots to hold this newly joined node.

Several methods can be used to implement the above scheme. For instance, when a node leaves, we do nothing if the backbone of the scatternet is untouched so that some update cost is reduced. However, this approach suffers a large cost when a node joins the network since we have to find where to put the newly joined node. One method is to use the broadcast method to travel the whole scatternet to find the master node with a free pure slave slot when a node joins. This may perform better if only a few of the existing piconets have free slots. The other method is to randomly select a master node and check if it has free slot. If it does not, we then select another random master node until one such master node is found. This approach performs better if the majority of the piconets have free slots.

To make the updating efficient, we should be able to quickly find the master node with an empty slot for a joining node and a pure slave to replace a leaving node if there is any. Our approach is to keep the current scatternet *compact* and assign a special node the *token* in a way such that all master nodes with label less than the token node do *not* have empty slot, and all master nodes with label larger than the token node do have empty slot. When a new node joins the network, we can simply assign it the empty pure slave slot and then update the token node if necessary. When a node leaves the network, we have to update the scatternet to keep the scatternet compact. Thus, we possibly have to move some nodes to fill the slot emptied by this left node.

Before we present the detail of our methods of updating the scatternet, we first study the possible status of the scatternet, which need be recorded in the token node.

When a new node requests joining the network, there are three possible scenarios to be discussed.

1. Current backbone is a balanced de Bruijn graph. Figure 6 illustrates an example. The token is held by the master node with the smallest label among all master nodes that have less than 5 same-prefix slaves. In this status, the master node with the token has some free slot for newly joined node and so do all master nodes with larger labels.

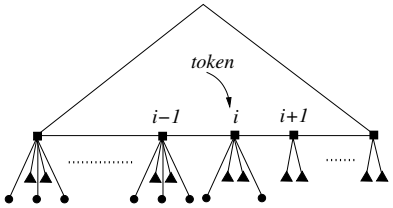


Fig. 6. Token in balanced de Bruijn graph.

2. Current backbone is pseudo-balanced de Bruijn graph $B(2, m)$ under *expanding* status, i.e., many nodes join the scatternet. Figure 7 illustrates an example. The token is held by the first master node with less than 5 same-prefix slaves in level $m + 1$ if it exists, otherwise the first master node in level m holds the token. In this status, all master nodes in level m and $m + 1$ do not have free slots except the last two master nodes in level $m + 1$. In other words, at most two master nodes have free slots.

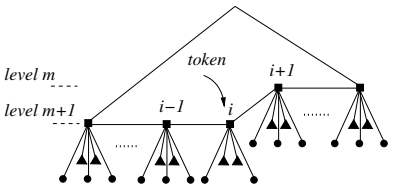


Fig. 7. Token in pseudo-balanced de Bruijn graph under expanding status.

3. Current backbone is a pseudo-balanced de Bruijn graph $B(2, m)$ under *shrinking* status, i.e., many nodes leave the scatternet. Figure 8 illustrates an example. The token is held by the master node in level m with the smallest label. In this status, each master node in level $m + 1$ and level m has 4 and 2 same-prefix slave nodes respectively.

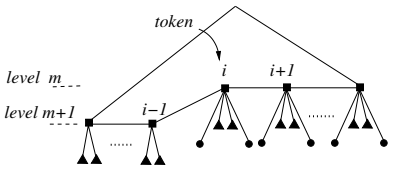


Fig. 8. Token in pseudo-balanced de Bruijn graph under shrinking status.

Those statuses *balanced*, *expanding*, *shrinking* will be recorded in the token data structure.

B. Node Joining

When a new node joins the network, there are three cases.

1. Token status is *balanced*, that is to say, the current backbone is a balanced de Bruijn graph. See Figure 6 for an illustration.

(a) The token node $x_1...x_m$ has less than 7 slaves. Then it simply adds the joining node into its slave set and assigns it a label $(x_1 \cdots x_m, y_1y_2y_3)$, where $y_1y_2y_3$ is one of the un-assigned MAC address in $\{001, 100, 110\}$. If the token node now has 7 slaves, then it passes the token to its successor.

(b) The token node is fully occupied by slaves. This could happen only when all master nodes in the scatternet have 7 slaves. Then the token is passed back to node 0^m if it is not at node 0^m. Change the token status to *expanding* and call Method 1 to split the current piconet mastered by the token node into two parts and add the joining node as a new pure slave with label $(x_1 \cdots x_m0, 001)$.

2. Token status is *expanding*, that is to say, current backbone is a pseudo-balanced de Bruijn graph under expanding status. See Figure 7 for an illustration.

(a) If the token node is in level $(m+1)$, i.e., with $(m+1)$-bits label $x_1...x_{m+1}$, the it must has less than 7 slaves. It simply adds the joining node into its slave set and assigns it a label $(x_1 \cdots x_{m+1}, y_1y_2y_3)$, where $y_1y_2y_3$ is one of the un-assigned labels in $\{001, 100, 110\}$. If the token node now has 7 slaves, then passes the token to its successor.

(b) If the token node is in level m, i.e., with m-bits label $x_1...x_m$. This could happen only when all master nodes in the scatternet has been fully occupied by 7 slaves. Call Method 1 to split the current piconet mastered by this token node into two piconets, and add the joining node as a new slave with label $(x_1 \cdots x_m0, 001)$.

3. Token status is *shrinking*, that is to say, current backbone is a pseudo-balanced de Bruijn graph under shrinking status. See Figure 8 for an illustration. In this case, the token node surely has exactly four slaves (see node leaving for more details). We first add the joining node as the slave of the token node and assign it one of the un-assigned MAC addresses in $\{001, 100, 110\}$. Call Method 1 to split current piconet into two piconets, and pass the token to the successor in level m. If the current token node is 1^m, then set token status to *balanced* and pass the token to master node 0^{m+1}. In other words, we basically undo the updating (piconets merging) caused by the previous node leaving event.

We then present our algorithm that splits one piconet mastered by node $x_1 \cdots x_m$ to two new piconets mastered by node $x_1 \cdots x_m0$ and node $x_1 \cdots x_m1$ respectively.

Method 1: Piconet split due to node joining

1. Token node $u = x_1 \cdots x_m$ promotes its slave node $v = (x_1 \cdots x_m, 100)$ as the master for a new piconet. We change the label $(x_1 \cdots x_m, y_1 y_2 y_3)$ of a pure slave node or an out-neighbor bridge slave node by simply appending y_2 in the MAC address, i.e., the new label is $(x_1 \cdots x_m y_1, y_2 y_3 y_2)$. Two new piconets have master nodes with labels $x_1 \cdots x_m 0$ and $x_1 \cdots x_m 1$ respectively. The detail of labelling and role updating is as follows:

(a) $(x_1 \cdots x_m, 000) \Rightarrow (x_1 \cdots x_m 0, 000)$, which assumes master role in first piconet.

(b) $(x_1 \cdots x_m, 001) \Rightarrow (x_1 \cdots x_m 0, 010)$, which assumes a bridge slave role in first piconet.

(c) $(x_1 \cdots x_m, 010) \Rightarrow (x_1 \cdots x_m 0, 101)$, which assumes a bridge slave role in first piconet.

(d) $(x_1 \cdots x_m, 100) \Rightarrow (x_1 \cdots x_m 1, 000)$, which assumes master role in second piconet.

(e) $(x_1 \cdots x_m, 101) \Rightarrow (x_1 \cdots x_m 1, 010)$, which assumes a bridge slave role in second piconet.

(f) $(x_1 \cdots x_m, 110) \Rightarrow (x_1 \cdots x_m 1, 101)$, which assumes a bridge slave role in second piconet.

Notice this label extension still preserves their prefix. Thus, after the piconet splitting, the message delivery will not be interrupted at all because old addresses are still reachable since the new label has the same prefix. In addition, the nodes with new labels for the corresponding MAC addresses will serve the bridge slave role in the two newly created piconets. Figure 9 illustrates the change while piconet splitting.

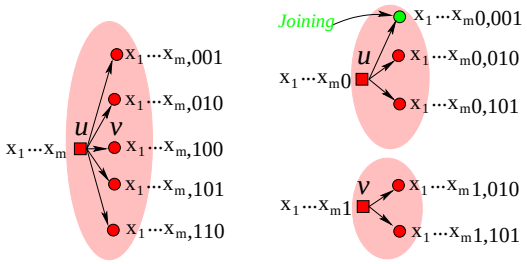


Fig. 9. Piconet splits due to node joining.

2. Then, both u and v need reselect the bridge slaves to connect with its in-neighbors and out-neighbors if needed. Simultaneously, both u and v's neighbors need reselect its same-prefix bridge slaves to connect with u and v. The selection still follows the rule described in Section II-B, Figure 3 illustrates all possible scenarios. Since the master nodes in the new piconets are in level $m+1$, each of them has at most 2 out-neighbors in the pseudo-balanced de Bruijn graph $B(2, m)$. Thus, we have enough bridge slave nodes for each new piconet. At the same time, their in-neighbor master nodes $yx_1x_2 \cdots x_{m-1}(x_m)$, where $y = 0$ or 1, of node u and v in the de Bruijn graph have to change one pure slave to bridge slave for connecting

node u or v. Notice this update is only restricted to local regions, so the update is totally localized.

3. Finally, the token is still kept by the master node $x_1...x_m0$, whose previous label is $x_1...x_m$.

C. Node Leaving

If a node leaves elegantly, it should first notify the token node before leaving. If a master/slave node leaves because unexpected reason such as power off, all of its neighborhood will detect it soon and notify the token node. Our method does not consider the detail of the exception detection process, we assume the token node can detect the node leaving in short time.

When the token node detects the node leaving, then there are three cases to be addressed again:

1. Token status is *balanced*, that is to say, current backbone is a balanced de Bruijn graph. Here two cases need be discussed:

(a) If the token node does have pure slave node, then the token node requests one pure slave to replace the position of the leaving node, including the label;

(b) If the token node u has no pure slave nodes, then it passes the token to its predecessor, say node v. There are two scenarios also, which as discussed as follows.

i. If node v has pure slaves, then it requests one pure slave to replace the position of the leaving node.

ii. If node v also has no pure slaves. This could happen only when $v = 1^m$, and all master nodes have only 2 slaves serving bridge slave role. Token node v changes the token status to *shrinking*, and call Method 2 to merge its corresponding critical pair, then ask one pure slave to replace the position of the leaving node.

2. Token status is *expanding*, that is to say, current backbone is a pseudo-balanced de Bruijn graph under expanding status.

(a) If the token node is in level m, i.e., with m-bits label $x_1...x_m$. This could happen only when all master nodes in the scatternet has been fully occupied by 7 slaves. The token need be passed the predecessor, which will ask one pure slave node to replace the position of the leaving node.

(b) If the token node is in level $(m+1)$, i.e., with $(m+1)$-bits label $x_1...x_{m+1}$. If the token node does have pure slave node, then the token node requests one pure slave to replace the position of the leaving node, otherwise two cases need be discussed here:

i. The least significant bit of the token node's label is 1. The token will be passed to be passed the predecessor, which will ask one pure slave node to replace the position of the leaving node.

ii. The least significant bit of the token node's label is 0. It first merges its corresponding critical pair by calling Method 2, then requests one pure slave to replace the position of the leaving node. Now if

the current token node is $0\cdots0$, then it changes the token status to *balanced* and passes the token to its predecessor $1\cdots1$.

3. Token status is *shrinking*, that is to say, current backbone is a pseudo-balanced de Bruijn graph under shrinking status.

(a) If the token node is not $0\cdots0$, then it passes the token to its second predecessor with least significant bit 0 in level $(m+1)$, which will call Method 2 to merge its critical pair piconet and ask one pure slave to replace the position of the leaving node.

(b) If the token node is $0\cdots0$, then it changes the token status to *balanced* and passes the token to node $1\cdots1$, which will ask one pure slave to replace the position of leaving node.

One special case is that the token node leaves. In this case, the token node will promote one of its pure slaves to replace it, i.e., to be the master node and the new token node. If no new pure slave exits, similarly, we have to ask some pure slave node from its predecessor to replace its role. When the token node did not leave elegantly, it is more complicated and we need fault tolerance about the token node, which is out of the scope of this paper.

We then describe our method to merge two piconets that are mastered by a critical pair.

Method 2: Piconet merge due to node leaving

1. Assume that the token node $u = x_1\cdots x_m0$ requests merging with its sibling master node $v = x_1\cdots x_m1$. The new piconet has master node with label $x_1\cdots x_m$. Notice that node u and node v each has at most 2 out-neighbors in the de Bruijn graph. The label change will be achieved by simply deleting the least significant bit as follows:

(a) $(x_1\cdots x_m0, 000) \Rightarrow (x_1\cdots x_m, 000)$, which is the master node in the new piconet.

(b) $(x_1\cdots x_m0, 010) \Rightarrow (x_1\cdots x_m, 001)$, which is a pure slave node or the bridge slave node to connect master node $x_1\cdots x_m00$ if it exists.

(c) $(x_1\cdots x_m0, 101) \Rightarrow (x_1\cdots x_m, 010)$, which is the bridge slave node to connect master node $x_1\cdots x_m0(1)$, whichever exists.

(d) $(x_1\cdots x_m1, 000)$ moves to replace the leaving node position.

(e) $(x_1\cdots x_m1, 010) \Rightarrow (x_1\cdots x_m, 101)$, which is the bridge slave node to connect master node $x_1\cdots x_m1(0)$, whichever exists.

(f) $(x_1\cdots x_m1, 101) \Rightarrow (x_1\cdots x_m, 110)$, which is a pure slave node or the bridge slave node to connect master node $x_1\cdots x_m11$ if it exists.

Notice this label shrink still preserves the label prefix. Thus, after the piconets merging, the message

delivery will not be affected at all because de Bruijn graph uses prefix based routing, old addresses are still reachable by the same prefix. The piconets mergence will not cause any routing problem although the node label shrink is not acknowledged by other nodes. At the same time, the sibling master node $v = x_1 \cdots x_m 1$ leaves to replace the position of leaving node. To continue the message delivery for node v, the new master node u will keep the new label of v for a period of time and forwards the message targeted to v accordingly. More detail is discussed in Section IV. Figure 10 illustrates the change of labels by merging piconets.

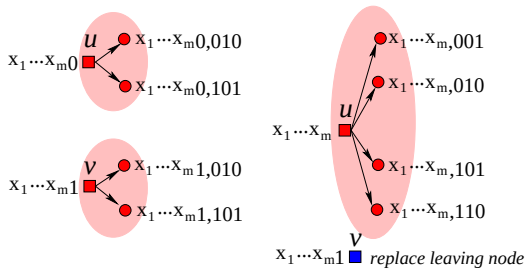


Fig. 10. Piconets merge due to node leaving.

2. Then, node u need reselect the bridge slaves to connect with its in-neighbors and out-neighbors if needed. Simultaneously, the neighboring master nodes of u and v need reselect their same-prefix bridge slaves to connect with u. The selection still follows the same rule described in Section II-B, please see Figure 3 for an illustration for all possible scenarios. Notice this update is totally localized.
3. The token is now kept by the master node $x_1 \cdots x_m$.

It is not difficult to prove the following theorem.

Theorem 2: Our method *locally* updates the dBBlue scatternet using at most $O(\log n)$ communications when a node joins or leaves the network. In most cases, the cost of updating the scatternet is actually $O(1)$ since the node can leave and join without affecting the remaining scatternet. The number of nodes affected when a node leaves or joins the network is always bounded from above by a constant. Our method can construct the structure incrementally when nodes join the network one by one.

D. Bounded Network Size

The method described so far can incrementally construct the scatternet when the nodes join the network one by one and can update the scatternet structure efficiently when nodes leave or join the network even frequently without affecting the worst case properties of the scatternet. This method is efficient in most cases, however, it could generate lots of merging and splitting of piconets in the worst case: a node joins the scatternet which causes the splitting of a piconet, then a node leaves which in turn causes the merging

of two piconets, and repeat joining, leaving.

In most applications, the size of the bluetooth network is often stable, for example, within $[n, c \cdot n]$ for a small constant $c > 1$. If this is the case, we can apply the following approach to build the scatternet. First, we use Algorithm 1 to build a scatternet with n nodes. When a new node joins the network, we first try to find an empty pure slave slot for this node from the current token node. If no empty slot exists, we then pass the token to the successor of the current token node. When all master nodes in the scatternet have 7 slaves, we will start creating another piconet to connect to the current backbone. In other words, instead of having 3 pure slave nodes, a master node from the scatternet backbone will replace the pure slave nodes by 3 piconets (at maximum). We call such piconets *associated* with the master node of the backbone. Clearly, a backbone based on a balanced de Bruijn graph $B(2, m)$ could support from $3 \cdot 2^m$ nodes to $6 \cdot 2^m$ nodes without associating piconets. By associating piconets to the master nodes of backbone, the number of nodes it can support is increased to $27 \cdot 2^m$ since we can replace each pure slave node by a piconet of 8 nodes.

One disadvantage of associating piconets to master nodes is that every master node in the backbone will have to forward more messages than the scatternet created by the method described previously. The other disadvantage is that when the network size goes beyond its supported scope, the updating of the scatternet is more costly than before.

IV. Routing in Scatternet

We first describe the routing in the dBBlue scatternet with a balanced backbone. If both source and target nodes are masters, we assume the source master node u has a label $x_1x_2 \cdots x_m$ and the target master node v has a label $y_1 \cdots y_m$. According to the routing mechanism described in Section II-A, node u simply forwards the message to its neighbor master node $u_1 = x_2 \cdots x_m y_1$, relayed by their common bridge node $(x_1 \cdots x_m, 010)$ if $y_1 = 0$ or by $(x_1 \cdots x_m, 101)$ if $y_1 = 1$. Then u_1 forwards the message again to its neighbor master node accordingly. Clearly, the message is guaranteed to reach the target in at most $2m$ steps. If the source node is a slave, it first sends the messages to its master node. Notice that a pure slave node has only one master node and the bridge slave node has two master nodes. Then the bridge slave node just randomly picks one master node. Similarly if the target node is a slave, the message will be first forwarded to its master node. The procedure of routing message between these two master nodes is same as the previous description. Clearly, the routing path from one master node to another master node is at most $2m$ hops. The longest path between two nodes happens from a slave node to another slave node, which is at most $2m + 2$ hops. From $2^{m-1} < \lceil \frac{n}{6} \rceil$, we have $m < 1 + \log \lceil \frac{n}{6} \rceil$.

Thus, the diameter of the de Bruijn-based scatternet is $4 + 2\log\lceil\frac{n}{6}\rceil$.

Theorem 3: For any two nodes in dBBlue scatternet, there is a path with at most $4 + 2\log\lceil\frac{n}{6}\rceil$ hops and such path can be found locally by each intermediate node based on the label of the target.

Notice that, two assumptions are made in our routing scheme described above: (1) the source node knows the label of the target node, and (2) the backbone of the scatternet is based on a balanced de Bruijn graph. We will not try to resolve the first assumption in this paper, but discuss it briefly here. In the network communication, usually only IP address of the target is known. We can adopt a mechanism similar to that used in Peer-to-peer system D2B[6]: the *Label-IPAddr* pairs are distributively stored in *master* nodes on backbone in the way that the prefix of IP address matches the label of the host *master* node. Any node can query the backbone using the target IP address to get the target label and then sends message to the target node based on the target node's label. The *Label-IPAddr* pair of a node can also be broadcasted to the whole network if the nodes' leaving and joining is not frequent, i.e., the labels of nodes do not change frequently. Here, we discuss briefly how to perform broadcast in de Bruijn graph such that it guarantees to reach each node exactly once. We initiate the broadcast from node 0^m. Each node with label $0y_1 \cdots y_{m-1}$ continues forwarding the message to its out-neighbors. The forwarding is terminated when it reaches the nodes whose most significant bit is 1. Figure 11 illustrates such broadcast procedure. The broadcast basically works same as the breadth first search (BFS) in a binary tree. Clearly, a node will only forward the message to nodes with larger labels. Thus, a node receives the message exactly once. The communication cost of such broadcasting is exactly n messages.

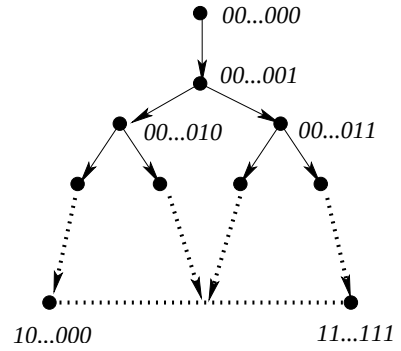


Fig. 11. Broadcast in de Bruijn graph.

We then discuss in detail how to route the packets when the scatternet backbone is pseudo-balanced. Assume the source master node u has a label $x_1x_2 \cdots x_{s-1}x_s$ and the target master node v has label $y_1y_2 \cdots y_{t-1}y_t$, where $s, t \in [k, k+1]$. Node u will forward the packet to its out-neighbor master node v with a label $x_2 \cdots x_{s-1}x_s$, or $x_2 \cdots x_{s-1}x_sy_1$, or $x_2 \cdots x_{s-1}x_sy_1y_2$. Notice that since the labels of all

nodes are a universal prefix set, we know that *exactly* one of these three labels does exist. Consequently, the diameter of pseudo-balanced de Bruijn graph is still $O(\log n)$. The bridge slave node from u to v could have a MAC address as follows (1) 010 if a master node with label $x_2 \cdots x_{s-1}x_s$ exists; or (2) $y_1\overline{y_1}y_1$ if a master node with label $x_2 \cdots x_{s-1}x_s y_1$ exists; or (3) $y_1 y_2 \overline{y_2}$ if a master node with label $x_2 \cdots x_{s-1}x_s y_1 y_2$ exists. Notice that we do not need any lookup table for relaying, since the MAC of a relaying bridge can be calculated easily. See Section II-B for more detail about the rules of labelling nodes and assigning MAC addresses in a piconet. A shorter route is obtained by looking for the longest sequence that is suffix of $x_1x_2 \cdots x_s$ and prefix of $y_1y_2 \cdots y_t$.

For the purpose of illustration, let's see how we route packets from a master node $x_1x_2x_3x_4 = 0010$ to a master node $y_1y_2y_3y_4 = 0001$ in the scatternet based on the de Bruijn graph illustrated in Figure 2. First, the master node 0010 checks the labels of all out-neighbor master nodes and finds that master node with label $x_2x_3x_4 = 010$ exists. Then it forwards the packet to master node 010 via the bridge slave node with MAC 010. Similarly, master node $x_2x_3x_4 = 010$ forwards the packet to master node with label $x_3x_4y_1 = 100$ via the bridge slave with MAC 010. Finally, the master node $x_3x_4y_1 = 100$ forwards the packet to node $y_1y_2y_3y_4 = 0001$ via the bridge slave with MAC 010. Notice that the last step it takes a shorter path other than via another master node $x_4y_1y_2y_3 = 0000$.

At last, we discuss how to route the messages while the scatternet is on updating due to nodes leaving or joining the network. When a node joins the network, the piconet mastered by the token node may be split into two piconets. Clearly, the message still can be routed since the labels of the two newly created piconets are the children of this token node. Similarly, when two piconets are merged to create a new piconet, the label-based routing still successfully route the packets. The remaining case is that when a node leaves, we may need find a pure slave node v from the current token node t to fill the space emptied by this left node. When a message targeted to node v reaches the piconet mastered by the token node t, node v has already been moved. To remedy this, we apply a mechanism similar to the mail-forwarding service provided by the post-office: the master node t will keep a record of the nodes moved to other piconets and its new label within a time window. When a message targeted for v reaches, the master node forwards the message to the new destination and also acknowledges the source node of the new label of v. The source node will then cache the label of node v if it is frequently used. To decrease messages forwarding, every master node could record the frequency that a slave node receives messages from other node. When a pure slave node is visited frequently by other nodes, then we switch its role with one of the bridge slaves with same prefix and broadcast the new labels of these two nodes to the network. When

we have to move a pure slave node to other piconet to make the scatternet compact, the pure slave node is the least frequently visited nodes among the current piconet.

V. Related Work

Zaruba, Basagni and Chlamtac [14] proposed two protocols for forming connected scatternet. In both cases, the resulting topology is termed a *bluetree*. The number of roles each node can assume is limited to two or three. The first protocol is initiated by a single node, called the *blueroot*, which will be the root of the bluetree. A rooted spanning tree is built as follows. The root will be assigned the role of master. Every one hop neighbor of the root will be its slave. The children of the root will be now assigned an additional master role, and all their neighbors that are not assigned any roles yet will become slaves of these newly created masters. This procedure is repeated recursively till all nodes are assigned. Each node is slave for only one master, the one that *paged* it first. Each internal node of the tree is a master on one piconet, and slave of another master (its parent in the initial tree). In order to limit the number of slaves, they [14] observed that if a node in unit disk graph has more than five neighbors, then at least two of them must be connected. This observation is used to re-configure the tree so that each master node has no more than 5 slaves. If a master node has more than 5 slaves, it selects its two slaves s_1 and s_2 that are connected and instructs s_2 to be master of s_1, and then disconnects s_2 from itself. Such branch reorganization is carried throughout the network. However, whether this approach will terminate is not proved in [14]. Tan *et al.* [13] proposed a similar method for single-hop network. In the second protocol [14], several roots are initially selected. Each of them then creates its own scatternet as in the first protocol. In the second phase, sub-tree scatternets are connected into one scatternet spanning the entire network. Notice that the tree topology suffers from a major drawback: the root is a communication bottleneck as it will be overloaded by communications between the different parts of the tree. Obviously, the root node in the tree-based scatternet is the bottleneck of the network and its congestion is $O(1)$, assuming that total traffic demand is a unit and is uniformly distributed. In addition, dynamic updating that preserves correct routing is not discussed in these protocols.

Law, Mehta and Siu [8] described an algorithm that creates connected degree bounded scatternet in single-hop networks. The final structure is a tree like scatternet, which limits efficiency and robustness. A single-hop Bluetooth scatternet formation scheme based on 1-factors is described in [1]. However, piconets are not degree limited in that scheme.

Salonidis *et al.* [12] proposed another topology construction algorithm recently. It first collects neighborhood information using an inquiry procedure, where senders search for receivers on randomly chosen

frequencies, and the detected receivers reply after random backoff delay. Leader is elected in the process, one for each connected component. Leader then collects the information about the whole network, decides the roles for each node, and distributes back the roles. In other words, basically, it is a centralized approach. Thus, the solution is not scalable, and not localized. Moreover, how to assign the roles is not elaborated in [12]. They also assume up to 36 nodes in the network. Another centralized solution for single-hop networks, where the traffic between any pair of nodes is known a priori, is described in [9].

Sun, Chang and Lai [10] described a self-routing topology for single-hop Bluetooth networks. Nodes are organized and maintained in a search tree structure, with Bluetooth ID's as keys (these keys are also used for routing). It relies on a sophisticated scatternet merge procedure with significant communication overhead for creation and maintenance. Bluerings as scatternets are proposed in [4]. Ring structure for Bluetooth has simplicity and easy creation as advantage, but it suffers large diameter (i.e., the maximum number of hops between any two devices) and large number of piconets.

The works are most related to our dBBlue scatternet construction method is [2] and [6].

Barriere, Fraigniaud, Narajanan, and Opatrny [2] described a connected degree limited and distributed scatternet formation solution based on projective geometry for single-hop networks. They assume that only slave nodes can act as bridges. They described procedures for adding and deleting nodes from the networks and claimed that its communication cost is $O(\log^4 n \log^4 \log n)$ and the computation cost is $O(\log^2 n \log^2 \log n)$, where n is the number of nodes in the network. The degree of the scatternet can be fixed to any $q+1$, where q is a power of a prime number. However, in their method, every node need hold information of the projective plane and the master node who has the "token" needs to know the information of the projective scatternet (which label should be used for the new coming master and which existing nodes need to be connected to it). However, the authors did not discuss in detail how to compute the labels for the new master and its slaves, and what will happen when the number of nodes reaches the number of nodes of a complete projective scatternets.

Notice that our dBBlue scatternet can be easily transformed to support a Bluetooth network in which a piconet has any number $p \geq 6$ of slaves, while the method in [2] can only support the piconet with $q+1$ slaves where q is a power of a prime number. Moreover, the dynamic updating cost of dBBlue is at most $O(\log n)$.

The construction of dBBlue scatternet is inspired by the method proposed by Fraigniaud and Gauron [6] for constructing a network topology for P2P environment based on de Bruijn graph. When a node u joins the P2P network, it [6] randomly selects a node v in the de Bruijn graph and then creates two

children nodes of v: one for v and one for u. This random selection of node v cannot be applied to Bluetooth scatternet since it may create a de Bruijn graph with node whose degree is large than 7. It is not difficult to show that for Bluetooth scatternet, we can only afford the de Bruijn graph whose node label lengths differ by at most 1. In this paper, we proposed a novel method for assigning MAC addresses to nodes such that a self-routing is still possible during the updating procedures when node leaves or joins the network. The de Bruijn graph is used as backbone of the scatternet in our dBBlue structure.

VI. Conclusion

In this paper, we addressed the problem of scatternet formation for single-hop Bluetooth based ad hoc networks, with minimal communication overhead. We adopted the well-known structure *de Bruijn graph* to form the backbone of the *dBBlue* scatternet. The diameter of the scatternet *dBBlue* is $O(\log n)$ and we can find a path with at most $O(\log n)$ hops between every pair of nodes without using any routing table. A useful MAC assignment mechanism is integrated in dBBlue protocol to support efficient and resilient self-routing, and hence facilitates the inter-piconet scheduling especially when Bluetooth system runs in *SCO* mode. We discussed in detail the method to *locally* update the structure *dBBlue* using at most $O(\log n)$ communications when a node joins or leaves the network. In most cases, the cost of updating the scatternet is actually $O(1)$. Our method can construct the structure *dBBlue* incrementally when the nodes join the network one by one. In addition, the structure formed by our method can sustain the faults of 2 nodes and the network is still guaranteed to be connected. If a node detects a fault of some neighboring master node or bridge slave node, it can dynamically re-route the packets and the path traveled by the packet is still at most $O(\log n)$ hops. Previously no method can guarantee all these properties although some methods can achieve some of the properties.

References

[1] S. Baatz, S. Bieschke, M. Frank, P. Martini, C. Scholz, and C. Kuhl. Building efficient bluetooth scatternet topologies from 1-factors. In *Proc. IASTED Wireless and Optical Communications WOC*, 2002.

[2] L. Barriere, P Fraigniaud, L. Narajanan, and J. Opatrny. Dynamic construction of bluetooth scatternets of fixed degree and low diameter. In *14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 781–790, 2003.

[3] S. Basagni, R. Bruno, and C. Petrioli. Device discovery in bluetooth networks: A scatternet perspective. In *Proc. IFIP-TC6 Networking Conference, Networking 2002*, 2002.

[4] F. Cgun-Choong and C. Kee-Chaing. Bluerings - bluetooth scatternets with ring structure. In *Proc. IASTED Wireless and Optical Communications WOC*, 2002.

[5] N. de Bruijn. A combinatorial problem. In *Koninklijke Nederlandse Academie van Wetenschappen*, 49, pages 758–764, 1946.

[6] Pierre Fraigniaud and Philippe Gauron. The content-addressable network d2b. Technical Report Technical Report TR-LRI-1349 (also appeared in 22nd ACM Symp. on Principles of Distributed Computing (PODC)), 2003.

[7] Jaap C. Haartsen. The bluetooth radio system. *IEEE Personal Communications*, 7:28–36, 2000.

[8] C. Law, A.K. Mehta, and K.Y. Siu. Performance of a new bluetooth scatternet formation protocol. In *Proc. ACM Symposium on Mobile Ad Hoc Networking and Computing MobiHoc*, pages 183–192, 2001.

[9] D. Miorandi and A. Zanella. On the optimal topology of bluetooth piconets: Roles swapping algorithms. In *Proc. Mediterranean Conference on Ad Hoc Networks MedHoc*, 2002.

[10] C.K. Chang M.T. Sun and T.H. Lai. A self-routing topology for bluetooth scatternets. In *2002 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '02)*, 2002.

[11] C. Petrioli and S. Basagni. Degree-constrained multihop scatternet formation for bluetooth networks. In *Proc. IEEE GLOBECOM*, 2002.

[12] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed topology construction of bluetooth personal area networks. In *Proc. IEEE INFOCOM*, 2001.

[13] G. Tan, A. Miu, J. Guttag, and H. Balakrishnan. Forming scatternets from bluetooth personal area networks. Technical Report MIT-LCS-TR-826, MIT, 2001.

[14] G.V. Zaruba, S. Basagni, and I. Chlamtac. Bluetrees - scatternet formation to enable bluetooth based ad hoc networks. In *Proc. IEEE International Conference on Communications(ICC)*, 2001.