A Bluetooth Scatternet Formation Algorithm

Ching Law and Kai-Yeung Siu
Massachusetts Institute of Technology

Abstract— **A Bluetooth ad hoc network can be formed by interconnecting piconets into scatternets. The constraints and properties of Bluetooth scatternets present special challenges in forming an ad hoc network efficiently. In this paper, we present and analyse a new randomized distributed algorithm for Bluetooth scatternet formation.**

We prove that our algorithm achieves $O(\log n)$ time complexity and $O(n)$ message complexity. We show that, 1) in the scatternet formed by our algorithm, any device is a member of at most two piconets; and 2) the number of piconets is close to be minimal.

I. Introduction

Bluetooth [1], [2], [3], [4] is an emerging low cost and low power short-range radio technology. It has been projected that as many as 200 million Bluetooth devices will be shipped in year 2001 [3]. Thus, Bluetooth is likely to become another important platform for ad hoc networking. Ad hoc networking over Bluetooth can lead to many useful applications. For example, in a conference room, a special announcement can be broadcast to the Bluetooth-enabled mobile phones and hand-held computers through an ad hoc network. Bluetooth ad hoc networks can also be used for rapid deployment of EMID (electromagnetic identification) readers [5].

The area of ad hoc networking has gathered much research interests in the past years. Many studies have concentrated on the routing issues of ad hoc networks [6]. These studies usually assume that any two in-range nodes can communicate with each other. Therefore, an ad hoc network can be modeled as a graph such that the in-range nodes are adjacent. For example, simulation-based studies [7], [8] of ad hoc routing protocols have been conducted with a link-layer model based on or similar to the IEEE 802.11b standard.

A Bluetooth ad hoc network, however, brings new challenges. There are specific Bluetooth constraints not present in other wireless networks. A Bluetooth network is composed of *piconets*. Each piconet contains one master and up to seven slaves. Piconets can be connected into a *scatternet* by sharing slaves. As shown by Miklos *et al.* [9] and Zurbes [10], the configuration of a scatternet has great effects on the performance of the network. For instance, when a scatternet contains more piconets, the rate of packet collisions increases.

Before we can tap the enormous power of Bluetooth ad hoc networking, we must first devise an efficient protocol to form a scatternet from isolated Bluetooth devices.

In this paper, we study the problem of scatternet formation in the situation where the devices are in-range of one another. We adopt a two-layered approach for our investigation of this problem. First, we investigate how these devices can get organized into scatternets. We evaluate the performance of a new scatternet formation protocol. Second, as a subroutine of the protocol, we study how the devices can discover each other efficiently.

In Section II, we discuss the relevant research on Bluetooth scatternets. In Section III, we introduce the problem of scatternet formation. We present our algorithm in Section IV and analyse it in Section V. Section VI concludes with remarks on future work.

This work is supported in part by the Auto-ID Center <autoidcenter.org>.

II. Related Work

Aggarwal *et al.* [11] introduce a scatternet formation algorithm. Their algorithm first partitions the network into independent piconets, and then elects a 'super-master' that knows about all the nodes. However, the resulting network is not a scatternet, because the piconets are not inter-connected. Thus, another phase of re-organization is required.

Salonidis *et al.* [12] present a scatternet formation algorithm – BTCP (Bluetooth Topology Construction Protocol). BTCP has two phases: first, a leader is elected with a complete knowledge of all devices, and second, this leader will tell other devices how a scatternet should be formed. Our algorithm has only one phase: the scatternet is formed once a leader is elected. Since the topology is determined by a single device, BTCP has more flexibility in constructing the scatternet. A default scheme is presented in [12] for up to 36 devices.

III. Preliminaries

In this section we introduce some terminologies and performance measures for the scatternet formation problem.

Bluetooth devices share 79 channels of 1 Mhz bandwidth within the 2.45 GHz band. Frequency hopping is used to reduce interference and enhance security.

When two Bluetooth devices are connected, one of the devices acts as a *master* and the other device acts as a *slave*. Any Bluetooth device can perform the role of a master and/or a slave.

A Bluetooth device can discover other devices by the inquiry process. If the inquiry process succeeds, the master learns the address (which is unique for each Bluetooth device) and the clock of the slave. Then the master and the slave can be connected with the page process.

A *piconet* consists of 1 master and 1 to k slaves (k is 7 in Bluetooth 1.0b). All packets are exchanged between a master and its slaves within a piconet. There is no direct master-master or slave-slave communication. A device can be a slave in several piconets but be a master in only one

0-7803-7206-9/01/$17.00 © 2001 IEEE

piconet. The *degree* of a device is the number of piconets to which the device belongs. A device is *unshared* if its degree is 0 or 1. Otherwise, it is *shared*. A *scatternet* is a set of piconets connected through shared devices.

The problem of scatternet formation: How do a collection of isolated devices form a scatternet? The devices are isolated in the beginning, each device is not aware of the other devices. Therefore, the scatternet formation protocol must be distributed. We assume that the devices are in the communication range [1] of each other. This means that, potentially, any pair of devices can be connected directly.

A scatternet formation protocol has two important performance measures:

- time complexity — amount of time to form a scatternet. A scatternet must be formed as fast as possible to minimize the delay experienced by the users.
- message complexity — number of messages sent between the devices. This is important because Bluetooth devices usually operate with limited power. By reducing the number of messages sent, power consumption is conserved.

In addition, it is also crucial to have scatternets of good quality. It is not very useful to have scatternets that lead to poor network performance. Thus, we should aim to produce scatternet that facilitates inter-piconet communications. It is not easy to quantify the quality of a scatternet, but we believe the following quality measures are good indicators.

- number of piconets — a measurement of a scatternet's efficiency. Since all piconets share the same set of 79 channels, there will be more collisions when there are more piconets. As shown in [10], the burst failure rate increases with the number of piconets.
- maximum degree of the devices — the maximum number of piconets that any device belongs to. Since the piconets communicate through shared slaves, if a slave belongs to many piconets, then this slave could become the bottleneck of inter-piconet communications. A shared slave has to be time multiplexed between the piconets that it belongs to. Therefore, a shared slave of high degree could become overloaded.
- network diameter — maximum number of hops between any pair of devices. This will provide us with an estimation of the maximum routing delay of the scatternet.

A good balance among the quality measures is desirable. Consider, for example, a star topology: a single "central" slave is shared by all piconets. In such a scatternet of n devices with every piconet containing k slaves, there are $\lceil (n-1)/k \rceil$ piconets. Although the number of piconets is minimized (see Remark 1), this scatternet probably would not perform very well in practice because the shared slave will be overwhelmed, unless the network is small.

Remark 1: Let k be the maximum number of slaves in a piconet. A scatternet of n devices must contain at least $\lceil (n-1)/k \rceil$ piconets.

Proof: Omitted for brevity. ■

[1] 10m to 100m in Bluetooth 1.0b

IV. Algorithm

In this section, we present our scatternet formation algorithm. The development of this algorithm was inspired by our research on resource discovery algorithms in general networks [13].

Initially, we are given a set of isolated but in-range devices. During the execution of the algorithm, the devices are partitioned into *components*. A component is a set of interconnected devices. A component can be a single device, a piconet, or a scatternet. There is one *leader* in each component. For a single-device component, the only member is the leader. For a piconet, the master is the leader. For a scatternet, one of the masters is the leader. When a leader *retires*, it stops being a leader and will be inactive for the rest of the algorithm (unless it becomes a leader again). For any device v, let $\mathcal{S}(v)$ be the set of v's slaves. If v is not a master or has no slaves, then $\mathcal{S}(v) = \emptyset$. Let $k \geq 2$ be the maximum number of slaves in a piconet. Thus $|\mathcal{S}(V)| \leq k$ for any v.

In Lemma 2, we will prove the following invariances for the algorithm:

- Each leader either has no slave, or has at least one unshared slave in its piconet.
- Each leader has fewer than k slaves in its piconet, i.e., $|\mathcal{S}(u)| < k$ for any leader u.

All leaders execute procedure Main in the beginning of each round. We assume a constant ϕ, such that procedure Main and the procedures called by it can be completed in ϕ seconds. A good choice of ϕ can be found by simulations (see [14]) and by prototyping. We assume that all leaders will call procedure Main at time $t_0 + i\phi$, for $i = 0, 1, \ldots$, where t_0 is the start time.

In the beginning, all devices are leaders. In procedure Main, a leader calls Seek with probability p, $1/3 < p < 2/3$. Otherwise, the leader calls Scan or asks an unshared slave to call Scan. During each round, only one device in each component will call Seek or Scan.

```
Main(leader u)
1  x ← a random number in [0, 1)
2  if x < p   (1/3 < p < 2/3)
3     then Seek(u)
4     else if S(u) = ∅
5             then Scan(u)
6             else v ← an unshared slave of u
7                  Scan(v)
```

When a leader executes Seek, it tries to acquire a new slave (which is running Scan). However, the leader may not always succeed, because, in any given round, the number of devices running Scan can be fewer than the number of devices running Seek. Therefore, if a leader is not able to contact a slave after certain time, it should give up and run Main again in the next round. Similarly, Scan might also fail in any given round. Essentially, during each round, a matching is found between the Seek devices and Scan devices. The number of connections made (size of the matching) is the smaller of the number of Seek devices and the number of Scan devices.

```
SEEK(u)
1  u performs INQUIRY
2  if a slave v is found
3     then u connects to slave v by PAGE
4          // S(u) ← S(u) ∪ {v}
5          CONNECTED(u, v)
```

```
SCAN(v)
1  v performs INQUIRY SCAN
2  if v is contacted by a master u
3     then v waits for u in PAGE SCAN
```

We note that SEEK and SCAN devices will go into PAGE and PAGE SCAN modes respectively after all inquiries are completed. The amount of time required is investigated in [14]. In general, we make sure that each master is matched to only one slave, and vice versa. The details of this low-level protocol are discussed in [14]. When a leader u running SEEK connects to a slave v running SCAN, procedure CONNECTED(u, v) is called. If v's other master is w, the piconets of u and w will try to merge if possible. Essentially, if the piconets of u and w can be fit into a single piconet (with at most $k-1$ slaves), then w and the devices in $\mathcal{S}(w)$ become slaves of u. This is performed by the procedure MERGE. Otherwise, some slaves are moved from $\mathcal{S}(u)$ to $\mathcal{S}(w)$ by the procedure MIGRATE. There are other special cases. See Figures 1, 2, and 3 for illustrations of some cases in procedure CONNECTED.

```
CONNECTED(leader u, slave v)
 1  if v is a leader
 2     then // v was an isolated leader
 3          if |S(u)| < k
 4             then v retires
 5             else u retires
 6                  y ← an unshared slave of u
 7                  MOVE({y}, u, v)
 8     else w ← the other master of v
 9          w retires
10          switch
11             case |S(u) ∪ S(w)| + 1 < k :
12                  MERGE(u, v, w); return
13             case |S(u)| = 1 :
14                  MOVE({u}, NIL, w)
15                  v disconnects from w ; return
16             case |S(u) ∪ S(w)| + 1 = k :
17                  u retires
18                  y ← an unshared slave of u
19                  MERGE(u, v, w)
20                  MOVE({y}, u, v)
21                  v becomes a leader ; return
22             case default :
23                  MIGRATE(u, v, w); return
```

Communications between u and w in CONNECTED, MERGE, MIGRATE, and MOVE are via their common slave v.

Procedure MERGE(u, v, w) makes w and all its slaves become u's slaves.

```
MERGE(master u, slave v, master w)
1  v disconnects from w
2  MOVE(S(w) \ v, w, u)
3  MOVE({w}, NIL, u)
```

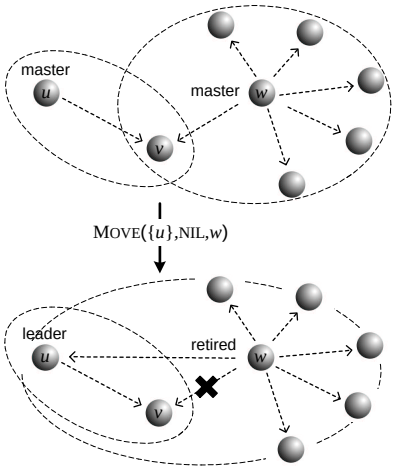

Fig. 1. Lines 13-15 ($|\mathcal{S}(u)| = 1$) in procedure CONNECTED for $k = 7$.

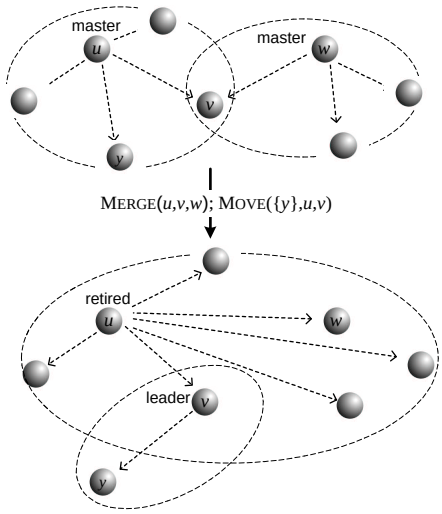

Fig. 2. Lines 16-21 ($|\mathcal{S}(u) \cup \mathcal{S}(w)|+1 = k$) in procedure CONNECTED for $k = 7$.

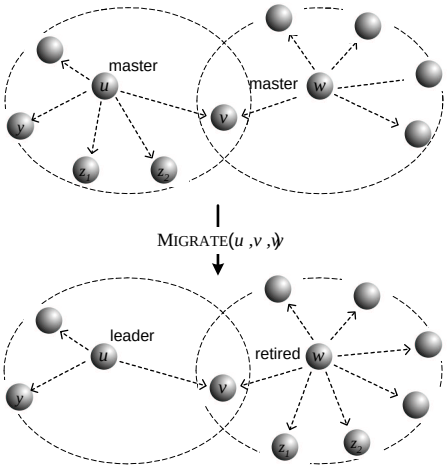

Fig. 3. Lines 22-23 (default) in procedure CONNECTED for $k = 7$.

Procedure MIGRATE(u, v, w) moves slaves from $\mathcal{S}(u)$ to $\mathcal{S}(w)$ until $\mathcal{S}(w)$ is full or when only two slaves are left in $\mathcal{S}(u)$.

```
MIGRATE(master u, slave v, master w)
1  i ← min(k − |S(w)|, |S(u)| − 2)
2   // i is the number of slaves to migrate
3  if i > 0
4     then y ← an unshared slave of u
5          Z ← { i slaves in S(u) \ {y, v} }
6          MOVE(Z, u, w)
```

Procedure MOVE is a subroutine called by CONNECTED, MERGE, and MIGRATE. All devices in set Z disconnect from u and become slaves of w.

```
MOVE(set Z, master u, master w)
1  devices in Z disconnect from u
2  devices in Z wait for w in PAGE SCAN
3  w connects to devices in Z by PAGE
```

Lemma 2: During the execution of the algorithm, the following invariances hold:

- Each leader has either no slave, or has at least one unshared slave.
- Each leader has fewer than k slaves.

Proof: We will prove the invariances by induction.

In the beginning, all devices are leaders without slaves. So our invariances are satisfied.

Assuming a state satisfying the claim, we only have to make sure that CONNECTED and its calls to MERGE, MIGRATE and MOVE preserve the invariances, because no piconet is formed or modified in MAIN, SEEK, SCAN.

Consider the procedure CONNECTED. Let $\mathcal{S}'(u)$ and $\mathcal{S}'(w)$ be the slaves of u and w after CONNECTED(u, v) is returned.

First, we consider the case v is a leader (lines 1-7). If v is a leader, it means that v does not have any slave. If $|\mathcal{S}(u)| < k$ (lines 3-4), v would become an unshared slave of u. If u has exactly k slaves (lines 5-7), then one unshared slave y is moved from $\mathcal{S}(u)$ to $\mathcal{S}(v)$. Thus, $\mathcal{S}'(v)$ contains an unshared slave. In this case, u is retired so that it does not need an unshared slave.

Second, we consider the case that v is a slave of a leader w. Master w will no longer be a leader, so it does not have to satisfy the invariances. There are four cases:

$(|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 < k)$: All devices in $\mathcal{S}(w)$ become slaves of u. Device v was an unshared slave in $\mathcal{S}(w)$. After the merge, u is the only master of v, so v becomes an unshared slave of u. Also, we note that $|\mathcal{S}'(u)| = |\mathcal{S}(u) \cup \mathcal{S}(w)| + 1$ is smaller than k by assumption.

$(|\mathcal{S}(u)| = 1)$: Leader u will have v as its only slave. v is unshared because it was w's unshared slave.

$(|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 = k)$: In this case, u retires so it does not need to satisfy the invariances. When v becomes a leader, it has an unshared slave y, obtained from u. Moreover, since v was a slave itself, so y is v's only slave.

default: Slaves in $\mathcal{S}(u)$ are migrated to $\mathcal{S}'(w)$ until $|\mathcal{S}'(w)|$ is k, or when $\mathcal{S}'(u)$ has only two slaves left (one of them is v). We note that MIGRATE will always reserve an unshared slave y for $\mathcal{S}'(u)$. By assumption, w had at most $k-1$ slaves before CONNECTED is called. Therefore, we can still always at least move one slave from $\mathcal{S}(u)$ to $\mathcal{S}'(w)$. Therefore, $|\mathcal{S}'(u)|$ is at most $k-1$, because at least one slave is removed after u has obtained slave v. ■

The last leader will keep calling MAIN even after the scatternet is formed. It is because the leader cannot be certain that all devices are already connected unless it knows the total number of devices. In practice, we can let the leader stop after it has failed to find any device for certain number of rounds. In particular, the probability that n leaders fail to make any connections for l rounds is $(p^n + (1-p)^n)^l$, which is less than $(5/9)^l$ for $n \geq 2$ and $1/3 < p < 2/3$.

V. ANALYSIS

A. Scatternet Properties

We show that the scatternet formed possesses two useful properties: small degrees for shared devices and small number of piconets. In [14], our simulations show that the diameter of the scatternet formed is $O(\log n)$ on average.

Lemma 3: At most one piconet in the scatternet formed by the algorithm contains fewer than $k-1$ slaves.

Proof: When a scatternet is formed, there is only one component. Thus there is only one leader left because each component has exactly one leader. Therefore, except for one piconet, the masters of the other piconets are not leaders. Thus we only need to show that when a leader is retired and if it is a master, it has at least $k-1$ slaves.

There are only four places in CONNECTED that a leader is retired.

line 4: v becomes a slave.

line 5: The test $|\mathcal{S}(u)| < k$ is false. Thus u had at least k slaves before line 7: MOVE$(\{y\}, u, v)$, which reduces the number of u's slaves by 1. Thus u would have $k-1$ slaves.

line 9: We must show that for all the four cases in lines 11-23, w will have at least $k-1$ slaves when CONNECTED returns.

In the first and the third cases, w loses all its slaves in the procedure MERGE and becomes a slave.

In the second case (lines 13-15), we have $|\mathcal{S}(u)| = 1$ but $|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 \geq k$ because the condition of the first case is not satisfied. u and w have one shared slave v. This implies that $|\mathcal{S}(u)| + |\mathcal{S}(w)| - 1 + 1 \geq k$, and thus $\mathcal{S}(w) \geq k-1$ before MOVE is called. Master w loses slave v, but gains a new slave u, so w still has at least $k-1$ when procedure CONNECTED returns.

In the last case (lines 22-23), we have $|\mathcal{S}(u) \cup \mathcal{S}(w)| + 1 \geq k+1$, and MIGRATE will move all devices in the piconet of u to w until w has k slaves or u has only 2 slaves left. In the latter case, only one slave in $\mathcal{S}(u) \cup \mathcal{S}(w)$ will not become a slave of w. Thus w would have at least $k-1$ slaves after the MIGRATE operation.

line 17: All slaves of w and w itself become slaves of u in line 19. We note that u and w had $k-1$ slaves in total (line 16), thus u should have k slaves after MERGE (line 19). MOVE (line 20) would remove one slave from u. Therefore, u still has $k-1$ slaves when CONNECTED returns. ■

Lemma 4: The algorithm forms a scatternet with $m-1$ devices of degree 2 and $n-m+1$ devices of degree 1, where n is the number of devices and m is the number of piconets.

Proof: Omitted for brevity. ■

Theorem 5: The scatternet formed by the algorithm contains at most $\lfloor (n-2)/(k-1) \rfloor + 1$ piconets.

Proof: Consider a scatternet produced by the algorithm. Let n be the number of devices and m be the number of piconets. By Lemma 3, at most one piconet has size less than k. (A piconet has size less than k if and only if it has fewer than $k-1$ nodes.) The remaining piconet has size at least 2. By Lemma 4, $m-1$ devices have degree 2 and the rest of the devices have degree 1. Therefore, we can conclude that the scatternet contains at least $k(m-1)+2-(m-1) = (k-1)(m-1)+2$ devices. Thus, $n \geq (k-1)(m-1)+2$. Since m is an integer, $m \leq \lfloor (n-2)/(k-1) \rfloor + 1$. ■

Comparing Theorem 5's upper bound $\lfloor (n-2)/(k-1) \rfloor + 1$ with Remark 1's lower bound $\lceil (n-1)/k \rceil$, we note that our bound is very close to be optimal. For example, when $n = 100$ and $k = 7$, our algorithm forms a scatternet containing at most 17 piconets, comparing with the lower bound of 15 piconets.

B. Asymptotic Complexities

We first derive the algorithm's time complexity and then its message complexity.

Lemma 6: During a round with at least 2 leaders, the number of leaders is reduced by a constant fraction with a constant probability.

Proof: Let $m \geq 2$ be the number of leaders. Let p be the probability that a leader chooses to run SEEK in any round. The algorithm specifies that $1/3 < p < 2/3$. We will assume $p \leq 1/2$, because if $p > 1/2$, we can switch the roles of SEEK and SCAN and the proof follows similarly.

During each round, we have a matching between the SEEK devices and the SCAN devices. Let random variable X be the number of SEEK devices in a given round. Since X is distributed binomially with parameter p, we have $\mathrm{E}[X] = pm$ and $\mathrm{Var}[X] = mp(1-p)$.

Let α be a real number between 0 and 1. If $(1-\alpha)pm \leq X \leq (1+\alpha)pm$, then at least $(1-\alpha)pm$ connections are made between the SEEK devices and SCAN devices because: 1) there are at least $(1-\alpha)pm$ SEEK devices; and 2) there are at least

$$m - (1+\alpha)pm = (1-p-\alpha p)m \geq (1-\alpha)pm$$

SCAN devices since $(1-p) \geq p$ if $p \leq 1/2$.

Thus, the probability of having at least $(1-\alpha)pm$ connections (matches between SEEK devices and SCAN devices) is

$$\begin{aligned}
&\Pr\{\text{at least } (1-\alpha)pm \text{ connections}\} \\
&= \Pr\{(1-\alpha)pm \leq X \leq (1+\alpha)pm\} \\
&= \Pr\{|X - pm| \leq \alpha pm\} \\
&= 1 - \Pr\{|X - pm| > \alpha pm\}.
\end{aligned}$$

The Chebyshev's inequality states that

$$\Pr\{|X - \mathrm{E}[X]| > t\} < t^{-2}\mathrm{Var}[X].$$

By setting $t = \alpha pm$, $\mathrm{E}[X] = pm$, and $\mathrm{Var}[X] = mp(1-p)$, we have

$$\Pr\{|X - pm| > \alpha pm\} < \frac{mp(1-p)}{(\alpha pm)^2} = \frac{1-p}{m\alpha^2 p}.$$

Since $m \geq 2$ and $p > 1/3$, we have $(1-p)/pm < 1$. Thus we can pick α so that $\alpha^2 > (1-p)/2p \geq (1-p)/mp$. Then $c = (1-p)/(2\alpha^2 p)$ is a constant smaller than 1. Therefore,

$$\Pr\{\text{at least } (1-\alpha)pm \text{ connections}\} > 1 - c.$$

Each connection reduces the number of leaders by 1. Therefore, with probability at least $1-c$, the number of leaders is reduced by a fraction $(1-\alpha)p$. ■

Theorem 7: The algorithm forms a scatternet in $O(\log n)$ rounds with probability at least $1 - 1/n^{\Theta(1)}$,

Proof: Omitted for brevity. ■

Theorem 8: The message complexity of the algorithm is $O(kn)$.

Proof: We first consider the message complexity of each invocation of the procedures. We note that each of the procedures MAIN, SCAN, SEEK, CONNECTED, MERGE, MIGRATE sends $O(1)$ messages. Procedure MOVE moves at most k devices. Thus it sends $O(k)$ messages.

To analyze the message complexity of MAIN, SEEK, and SCAN, it is sufficient to find the expected number of times that MAIN is called, because each call to MAIN leads to a call to SEEK or a call to SCAN.

First, we argue that we can assume that when a leader w chooses SCAN so that if it or its slave is contacted by another leader u, then w will retire. This is true except that if u has k slaves, then u will retire instead. See lines 5-7 in procedure CONNECTED. However, for simplicity of the analysis, we can assume that w retires instead of u. In other words, we can assume that w and u swap their identities whenever we are in this case. This will not affect our result because we only care about the total number of messages sent by these leaders. The high-level algorithm does not rely on an identifier of the device. (Device address is used by low-level Bluetooth INQUIRY and PAGE. But we note that these processes are independent between rounds in the algorithm.)

During any round, each leader chooses SCAN with probability $1-p$. Assume that a leader w has chosen SCAN. Leader w or w's unshared slave will definitely be contacted by another leader if the total number of SCAN devices is not more than the number of SEEK devices.

Let X_i be the random variable of the number of SCAN devices over i components. Thus, $\mathrm{E}[X_i] = (1-p)i$. Let $m \geq 2$ be the number of components.

We now assume that $p \geq 1/2$ because if otherwise, we can switch the roles of SCAN and SEEK. Assume w has chosen SCAN.

$$\begin{aligned}
&\Pr\{w \text{ or } w\text{'s slave is contacted by a leader}\} \\
&= \Pr\{X_{m-1} \leq m/2 - 1\}.
\end{aligned}$$

By Markov inequality, we have

$$\begin{aligned}\Pr\{X_{m-1} > m/2-1\} &= \Pr\{X_{m-1} \geq m/2-1/2\} \\ &\leq \mathrm{E}[X_{m-1}]/(m/2-1/2) \\ &= 2(1-p).\end{aligned}$$

Thus,

$$\Pr\{X_{m-1} \leq m/2-1\} \geq 1-2(1-p) = 2p-1.$$

Thus, each leader retires with probability at least $(1-p)(2p-1)$, which is positive except when $p = 1/2$.

We now consider the case where $p = 1/2$. In the proof of Lemma 6, we have

$$\Pr\{\text{at least } (1-\alpha)pm \text{ connections}\} > 1-(1-p)/(\alpha^2 pm).$$

Let $p = 1/2$, $\alpha = 1/2$, and $m \geq 5$, then

$$\Pr\{\text{at least } m/4 \text{ connections}\} > 1/5.$$

Therefore, with probability at least $1/5$, at least $m/4$ connections are made. And when that happens, each device has a probability of at least $1/4$ to be the slave of the a connection being made. This proves our argument for $m \geq 5$. The cases where $m = 2, 3, 4$ can be easily verified.

Therefore, any leader w has a constant probability of retiring during each round. This means that each leader is active for $O(1)$ rounds. Thus MAIN is called $O(n)$ times in total, and the overall message complexity for procedures MAIN, SEEK, SCAN is $O(n)$.

Procedure CONNECTED is called exactly $n-1$ times, thus the message complexity of CONNECTED is $O(n)$. Each call to CONNECTED could result in at most 1 call to MERGE, at most 1 call to MIGRATE, and at most 3 calls to MOVE. Thus the overall message complexity of MERGE and MIGRATE is $O(n)$, and the overall message complexity of MOVE is $O(kn)$. ■

Corollary 9: If k is a constant, then the message complexity of the algorithm is $O(n)$.

We note that $O(n)$ is the optimal asymptotic message complexity because each device needs to send at least one message to form a scatternet.

VI. Concluding Remarks

In this paper, we introduced a Bluetooth scatternet formation algorithm with $O(\log n)$ time complexity and $O(n)$ message complexity.

We have shown that the algorithm produces scatternet with some desirable properties: small number of piconets for minimizing inter-piconet interference, and low device degrees for avoiding network bottlenecks. Simulation results of our algorithm are presented in [15], [14].

Our protocol can be easily extended to work with dynamic environments (with devices joining and leaving the scatternet) and to support fault tolerance. Our current protocol already handles the events of devices joining. The new devices can simply start as leaders and will discover or be discovered by other devices. Additional work is required to deal with the case of devices leaving or failing. We can give an outline of a possible solution:

- If a master fails (or leaves the network), then a new master can be elected from the slaves. If the failed master was shared, then the new master should become a leader and merge with the rest of the scatternet by the protocol.
- If a shared slave fails, its master should become a leader again and then it will be connected to the rest of the scatternet by the protocol.
- Nothing needs to be done when an unshared slave fails, unless it is the only unshared slave of an active leader.
- In general, if we end up with a leader u with no unshared slave, then this leader has to disconnect from its shared slaves. Other masters connected to this leader u through the shared slaves should now become leaders again. This will allow the protocol to proceed as usual. Fortunately, this expensive reorganization should be a rare event.

References

[1] "The Bluetooth Special Interest Group," `http://www.bluetooth.com`.

[2] Jaap Haartsen, "Bluetooth - the universal radio interface for ad hoc, wireless connectivity," *Ericsson Review*, , no. 3, pp. 110–117, 1998.

[3] Jennifer Bray and Charles F. Sturman, *Bluetooth: Connect Without Cables*, Prentice Hall, 2001.

[4] Brent A. Miller and Chatschik Bisdikian, *Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications*, Prentice Hall, 2000.

[5] "Auto-ID Center," `http://autoidcenter.org/`.

[6] Charles E. Perkins, *Ad Hoc Networking*, Addison-Wesley, 2001.

[7] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," *Mobile Computing and Networking*, pp. 85–97, 1998.

[8] Samir R. Das, Robert Casta neda, and Jiangtao Yan, "Simulation-based performance evaluation of routing protocols for mobile ad hoc networks," *Mobile Networks and Applications*, vol. 5, pp. 179–189, 2000.

[9] Gy. Miklos, A. Racz, Z. Turanyi, A. Valko, and P. Johansson, "Performance aspects of Bluetooth scatternet formation," in *Proceedings of The First Annual Workshop on Mobile Ad Hoc Networking and Computing*, 2000.

[10] Stefan Zurbes, "Considerations on link and system throughput of Bluetooth networks," in *Proceedings of the 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2000, vol. 2, pp. 1315–1319.

[11] Alok Aggarwal, Manika Kapoor, Lakshmi Ramachandran, and Abhinanda Sarkar, "Clustering algorithms for wireless ad hoc networks," in *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Boston, MA, Aug. 2000, pp. 54–63.

[12] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire, "Distributed topology construction of Bluetooth personal area networks," in *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, 2001.

[13] Ching Law and Kai-Yeung Siu, "An O(log n) randomized resource discovery algorithm," in *Brief Announcements of the 14th International Symposium on Distributed Computing, Technical Report, Technical University of Madrid*, Oct. 2000, pp. 5–8.

[14] Ching Law, Amar K. Mehta, and Kai-Yeung Siu, "Performance of a new Bluetooth scatternet formation protocol," in *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing 2001*, Long Beach, California, USA, Oct. 2001.

[15] Amar K. Mehta, "Ad-hoc network formation using Bluetooth scatternets," M.S. thesis, Massachusetts Institute of Technology, June 2001.