Abstract—Bluetooth is a low-power, low-cost, short-range wireless communication system. In a Bluetooth ad hoc network, up to eight Bluetooth devices can communicate with each other in a special network called piconet. Scatternet can be formed by interconnecting piconets. In this paper, we present a distributed resource aware scalable scatternet formation algorithm. Though the proposed algorithm has two phases, it is not necessary that all the nodes be in the same phase at a given instant of time. The formed scatternet has the following properties: 1. All the devices may not be necessarily in each other’s transmission range. 2. Any device will be a member of at the most two piconets. 3. Higher resource-weighted devices are preferred over lower resource-weighted devices to perform the role of a master and/or a bridge. 4. Slave/Slave bridges are preferred over Master/Slave bridges. Extensive simulations have shown that the number of messages transmitted by low resource-weighted devices are significantly lesser than the number of messages exchanged by high resource-weighted devices, and the number of master/slave bridges are remarkably less as compared to the number of slave/slave bridges. The overall message complexity of the algorithm is $O(n)$. 

Index Terms—Ad Hoc Networks, Bluetooth, Scatternet, Topology construction.

I. INTRODUCTION

The need for a low cost, low power, short range radio technology has spurred the rapid development of Bluetooth [1][2]. A Bluetooth radio operates in the unlicensed, 2.4 GHz, Industrial-Scientific-Medical (ISM) band, sharing 79 channels each 1 MHz wide. It uses fast frequency hopping to enhance security and minimize interference. When two Bluetooth devices are connected to each other, one of the devices assumes the role of a master and the other becomes a slave. Each master may connect to at most seven active slaves, forming a network referred to as a piconet. The master mediates within the piconet and controls data transmission using a simple polling scheme.

The master employs the INQUIRY process to discover potential slaves. A 32-channel INQUIRY hopping sequence is used by the master, to hop 3200 times per second. Simultaneously, devices hoping to be discovered, perform an INQUIRY SCAN changing their listening frequency every 1.28 seconds, along the same sequence. The master learns the address and clock of the designated slave, when these frequency trains match. This match is followed by a PAGE process by the master, which uses the slave’s device address and estimated clock to determine the PAGE hopping sequence. Whereas, Slave doing PAGE SCAN hops for 1.28 seconds along the same sequence.

Once the master-slave device pair enters the CONNECTED state, subsequent communication takes place using a new hopping sequence determined by the master’s clock and address. This hopping sequence is spread over 79 channels and takes place at a rate of 1600 hops per second. All slaves belonging to a particular piconet use the same hopping sequence.

A Scatternet (Fig. 1) is a set of connected piconets. It is formed when two (or more) piconets share a common device, often referred to as a bridge. However since a device may be active in only one piconet at any instant of time, the bridge device must divide its time between the piconets. These nodes essentially act like switches relaying data from one piconet to another.

A bridge device may take on different roles in different piconets. It may be a master in one while being a slave in one or more piconets (master/slave type) or just a slave in all common piconets (slave/slave type). However, a bridge cannot be a master in more than one piconet. Bridges inherently have some additional responsibilities compared to a slave, as they are involved in a variety of tasks. Therefore they must have sufficient resources (both energy and computing) to carry out their gateway duties. This introduces potential issues in scatternet formation, as resource constrained devices should not be used as bridges. The same applies for masters, since they essentially control the piconet. Also, master/slave bridges tend to degrade the performance of the scatternet as compared to slave/slave bridges [12]. Therefore efficient scatternet formation and configuration is an important aspect of Bluetooth.

In this paper, we propose a novel, distributed 2-phase scatternet formation algorithm. The dynamic nature of our scheme allows efficient and connected topology construction.

† This work is supported by the Ohio Board of Reagents’ Doctoral Enhancement Funds and the National Science Foundation under grant CCR-0013361
at any time during the two phases or even after scatternet formation. It attempts to assign master and bridge roles to devices with higher resources, ensuring that the resulting scatternet is efficient and long lasting. Typical resource parameters include energy, computational power, mobility etc. In this paper we have prioritized forming bridges in the slave/slave configuration rather than the master/slave configuration, for the reasons mentioned earlier.

Every node just needs to know its own resources and computes a weighted resource value, which is used to decide its role allocation. The resulting scatternet has the following desirable properties:
1. All devices may not be in direct radio range of all other devices.
2. Any device will be a member of at most two piconets.
3. Due to a resource-aware role assignment, the scatternet lifetime is also increased.
4. There is no mandated hierarchical topology and eliminates redundant bridges between piconets.

The rest of the paper is organized as follows: In section II, we discuss related work. In section III, we present the distributed scatternet formation algorithm. Section IV evaluates the 2-phase algorithm, and we conclude in Section V with a look at the future work in this area.

II. RELATED WORK

In [3]-[5], all devices are assumed to be in each other’s radio range, so only single hop topologies are considered. The proposed solution in [4] looks at networks less than 36 nodes. [5] forms a tree-structured scatternet. It has the inherent deficiency of a hierarchical structure and is less reliable as it can easily be partitioned if a single node fails or moves out. [6] forms a scatternet among devices that are all not necessarily in range of each other. However it still generates a tree-structured scatternet, and relies on one device to start the scatternet formation process. In [7], the piconets generated may have more than 7 slaves, requiring slaves to use the Bluetooth low power PARK mode. This is not a workable solution in real traffic conditions. In [8], degree reduction techniques are applied, so that a piconet has less than 8 slaves, but requires current geographic location information at each device. In BlueNet [9] the number of slaves in a piconet is bounded (up to 7), but the resource-weight of each device and efficient role assignment (bridge or master) to devices is not taken into consideration. BlueMesh [10] algorithm guarantees a connected mesh, but the average number of slaves in each piconet is less than five. Also in [10], and [11] some nodes in the network may be a part of more than two piconets, which could result in reduced throughput performance.

III. SCATTERNET FORMATION ALGORITHM

The algorithm runs in two phases at each device. In the first phase, devices try to form new piconets, or align with existing piconets as bridges. At the end of this phase, master slave role assignments are made to all the nodes, with resource rich devices having a higher probability of becoming masters. The second phase which immediately follows the first phase, deals with bridge formation, redundant path detection and removal, and role optimizations.

Let K be the maximum number of slaves a master may have in a piconet (up to 7), and BRG be the number of bridges each piconet can have. Theoretically, BRG may vary from 1 to K, but too many bridges in a piconet may result in reduced piconet throughput. Therefore a reasonable value for BRG may be 2 or 3. The Resource-weight R, for each device is decided according to its energy, computational power, and its mobility. While the resource-weight is directly proportional to battery-power and computational power, it is inversely proportional to mobility. This computation does not in any way restrict the scope of R. It is a flexible value and may consist of any other parameters that the device may want to include. For any master device M, Let SL(M) be the set of M’s slaves, and B(M) be the set of M’s bridges.

In the beginning, all devices are considered to be isolated and execute procedure Phase1(), described below.

A. Phase1()

Each device selects a number N, inversely proportional to R. In Phase1(), each device executes procedure FindMaster() N times or until it becomes a slave. This provides a better opportunity for a low-resource-weighted node to become a slave, and a higher resource-weighted node, to become a master. If the device remains isolated after N attempts, it designates itself a master with 0 slaves and 0 bridges (i.e. set of slaves SL=Ø, and set of bridges B=Ø) and executes procedure FindSlave() until a Phase 1 timeout occurs or until the device has (K–BRG) number of slaves. This allows the master to form at least BRG number of bridges in the second phase. After becoming a master, device (u) initializes its path-set P = {u}. P denotes the masters (piconets); to which u have paths to, both direct and indirect.

Phase1() (isolated u)
1 select N ≈1/R
2 FindMaster (u) for N times or until state(u) = slave
3 if state(u) = isolated
4 then state = master, SL(u) = Ø, P(u) = {u}, B(u)=Ø
5 else // state = slave
6 u waits in slave mode to be communicated by master.
7 while ([SL(u)] < (K – BRG) and until Phase 1 timeout)
8 do FindSlave(u)
9 Phase2(u)

The Phase 1 timeout value is directly proportional to the maximum time φ [3], during which a device performing INQUIRY will be able to find a device in INQUIRY SCAN state. The value of φ depends upon the number of devices within the transmission range of any one device. A device in the INQUIRY state is connected to at the most one other device in INQUIRY SCAN state. Henceforth, whenever it is mentioned that the devices (master M with slave S) are paired, it refers to a device doing INQUIRY (master) connecting to a
device doing INQUIRY SCAN (slave), and connection refers to the CONNECTED state of the master and slave devices after the completion of the PAGE and PAGE SCAN procedures.

**FindSlave (M)**
1. M performs INQUIRY
2. if M pairs to device S in ∅ and connects to it,
3. then SL(M) = SL(M) ∪ {S}
4. if S was already a slave with another master M',
5. /\ M sends address and path-set (P') of M'
6. then if P ∩ P' = ∅ and |B(M)| < BRG
7. then designate S as a bridge
8. /\ S bridge to masters in P'
9. P = P ∪ P'
10. B(M) = B(M) ∪ {S}
11. else M disconnects from S

When a master device executes FindSlave(), it tries to discover another isolated slave device running FindMaster(). A slave executing FindMaster() performs INQUIRY SCAN where as, device executing FindSlave() performs INQUIRY. The device (master) executing FindSlave() is paired and connected to the device (slave) executing FindMaster() (Fig. 2a). If the slave device was isolated before this connection, then the slave sends the master its resource-weight. If a master is not able to contact a slave in time φ, and if Phase1 timeout has not occurred it executes FindSlave() again. Otherwise, it proceeds to Phase 2. Similarly, if the isolated device is not able to contact a master in time φ, and the FindMaster() execution-count has not exceeded N, it executes FindMaster() again. Otherwise, it designates itself as a master and executes FindSlave() (see Phase1 for details).

It is possible, that a master (M) executing FindSlave() be paired and connected to a slave (S) executing FindBridgeMaster(). S, being in Phase 2, is already a slave in another piconet. The master now tries to form a bridge with the slave S. (Fig.2). Let M' be the master of S. M now checks if S can be a bridge with M'. (See Phase 2, FindBridgeSlave() procedure for more details). If S forms a bridge with M (Fig. 2b), the paths for M and M' are updated. If S cannot become a bridge (Fig. 2c, 4a), M disconnects S.

**FindMaster (isolated S)**
1. S performs INQUIRY SCAN
2. if S pairs to M in ∅ and connects to it
3. then if M was searching for bridge,
4. then S informs M about its isolated state
5. if M does not accept S as slave,
6. /\ this is true when |SL(M)| > (K – BRG).
7. then S disconnects from M
8. else S becomes M's slave
9. else M makes S its slave.

Similarly, it is possible that the isolated device (S) is paired and connected to a master (M) executing FindBridgeSlave(). M, being in Phase 2, could be searching for a bridge. In this case, M decides to make S a slave depending on number of slaves it already has (See Phase 2, FindBridgeSlave() procedure for more details).

At the end of Phase 1, each device becomes either a master, or a slave. Every slave device waits to be communicated by the master, while, every master device executes Phase2() until it succeeds to form BRG number of bridges, or Phase 2 timeout occurs. The Phase 2 timeout value is directly proportional to BRG and φ (maximum time to pair two devices).

**FindBridgeMaster()**
1. While ([B(M)] < BRG, and until phase 2 timeout )
2. do p = random number in [0, 1]
3. if (0 ≤ p < 0.5 or state = M/S bridge)
4. then FindBridgeSlave(M)
5. else find ‘S’ such that , S ∉ SL(M), ∉ B(M), BrgCnt(S) < z, and R(S) = max( u, \( \forall u \in SL(M) \))
6. if S exists,
7. then send path-set P to S
8. inform S to execute FindBridgeMaster
9. if S becomes bridge
10. /\ P is the path-set of M'(other master of S)
11. then P = P ∪ P'
12. B(M) = B(M) ∪ {S}
13. else increment BrgCnt(S)
14. if S gets bridge notification
15. then state = M/S Bridge
16. P = P ∪ P'
17. B(M) = B(M) ∪ {M}

The master (M) may fail to find such a slave because of two possible reasons: (1) All the non-bridge slaves of the master have exceeded Bridge-Scan-Count. (2) Master has 0 slaves. In such a situation, the master executes FindBridgeMaster() to perform INQUIRY SCAN.

**Fig. 2:** (a): Device doing FindSlave() finds a device executing FindMaster(). (b), (c): Device doing FindSlave() finds a device executing FindBridgeMaster(). Bridge is formed in part (b), but not in part (c).
Before asking a slave (S) to execute `FindBridgeMaster()`, the master (M) sends its path-set P (list of masters to which M already has a direct/indirect path) to S (Fig. 3). Then the master waits to listen from the slave for an outcome. If S returns a failure to be a bridge, (Fig. 4a) M increments Bridge-Scan-Count associated with S. If M' designates S as a bridge, M' sends its path set P' to S, which in turn is relayed to M. M then updates its path-set \( P = P \cup P' \).

```
FindBridgeMaster(S, set P)
//M is previous master of S
1 S performs INQUIRY SCAN
2 if S pairs to M' in φ and connects to it
3    then S sends address and path-set (P) of M to M'
4    if M' doesn’t designate S as a bridge,(P \( \cup \) P' \( \neq \) \( \emptyset \))
5    then S disconnects from M’ and returns
6    else S becomes bridge between M and M'
7 S sends address and path set(P') of M' to M
```

A device executing `FindBridgeMaster()` performs INQUIRY SCAN. This device (S) may be a slave asked by its master to be a prospective slave/slave bridge, or a master deciding to become a prospective master/slave bridge. Suppose a master M' (performing INQUIRY) and S are paired and connected, S sends address and path set (P) of its master (M). Note: If S is a prospective master/slave bridge, it sends its own address and path-set. If M' agrees S to be a bridge, (Fig. 3) it sends its path set P' to S, which is relayed to its previous master M. If not, S disconnects from M', (Fig. 4a).

The master (M') executing `FindBridgeSlave()` performs INQUIRY. It pairs and connects to a slave(S). If S was a prospective master/slave bridge (S executing `FindBridgeMaster()`), it sends address and path set (P) of its previous master M to M'. M' compares its path-set P' with set P. If \( P \cap P' = \emptyset \) (Fig. 3), (this implies M' doesn’t have a known existing path to any of the piconets having path to M), then M' accepts S as a bridge with M to reach all masters in path-set P. This check ensures elimination of most of the redundant bridge formations. M' now sends its own path-set P' to S, and

```
FindBridgeSlave(Master M')
1 M' performs INQUIRY. P' is its path-set.
2 if M' pairs to S in φ and connects to it
3    then SL(M') = SL(M') \( \cup \) \{S\}
4    if S sends address and path set (P) of M
5    //M is master of S, S in FindBridgeMaster()
6    then if \( P \cap P' = \emptyset \)
7        then make S as a bridge
8            //S bridge between M and M'
9        then P' = P' \( \cup \) P
10       B(M')=B(M') \( \cup \) \{S\}
11    else S isolated, searching for master
12       if |SL(M)| \( \leq \) (K – BRG)
13          then accept S as a slave.
14          else M disconnects from S.
```

At the end of Phase 2, it may also happen that some Master/Slave bridges are Masters with 0 slaves. In that case, it changes its state to a pure slave with the master it has been a bridge with. Masters having less than K slaves execute Phase2() procedure periodically to accommodate new devices joining the scatternet thus making the algorithm scalable. Also, the masters periodically exchange their path-sets to eliminate redundant bridge formations with newly joined devices.
IV. SIMULATION RESULTS

In this section, we evaluate performance of our scatternet formation algorithm. We use simjava [13], a discrete event simulation package for Java to develop a Bluetooth Simulation.

We vary the total number of nodes from 4 to 200, and assign resource-weight (R) to each node randomly within the range (0, 1]. Though we have considered corresponding values of N (N ∝ 1/R) to be within the range 0-30, the value is flexible and may change with local policies. The number of nodes in any of the node’s transmission range is decided randomly.

We evaluate the performance by comparing the number of Master/Slave bridges formed with the number of Slave/Slave bridges formed. The role-assignments are checked against the resource-weight of the nodes. The number of messages that sent by nodes in different roles have been compared. Average number of slaves per piconet and average number of bridges per piconet formed in the scatternet have been plotted. O(n) message complexity is verified. The results are average of 15 trials runs for varying topologies with same number of nodes.

In this section, we evaluate performance of our scatternet formation algorithm. We use simjava [13], a discrete event simulation package for Java to develop a Bluetooth Simulation.

We vary the total number of nodes from 4 to 200, and assign resource-weight (R) to each node randomly within the range (0, 1]. Though we have considered corresponding values of N (N ∝ 1/R) to be within the range 0-30, the value is flexible and may change with local policies. The number of nodes in any of the node’s transmission range is decided randomly.

We evaluate the performance by comparing the number of Master/Slave bridges formed with the number of Slave/Slave bridges formed. The role-assignments are checked against the resource-weight of the nodes. The number of messages that sent by nodes in different roles have been compared. Average number of slaves per piconet and average number of bridges per piconet formed in the scatternet have been plotted. O(n) message complexity is verified. The results are average of 15 trials runs for varying topologies with same number of nodes.

In Fig. 5, role assignment of the nodes is plotted against their resource-weight. The role assignment indicates that most of the low resource-weighted devices are assigned the role of a slave and most of the higher resource-weighted devices are assigned the role of a master and/or a bridge.

Fig. 5. Nodes distribution according to their roles performed in scatternet and their resource weights (n = total number of nodes = 64)

Fig. 6 shows the number of devices assuming different roles (master, slave, slave/slave bridge, master/slave bridge). Clearly, the number of master/slave bridges formed is remarkably less as compared to the number of slave/slave bridges.

For low-power devices, it is important to minimize the number of messages sent in order to conserve energy. Fig. 7 shows the comparison between the average number of messages sent by a slave device, a slave/slave bridge device, and the average number of messages sent by a master or a Master/Slave bridge device. It is evident from the graph that the slaves (low-resource-weighted devices) send the lowest number of messages while forming the scatternet. Also, the number of messages sent by a slave/slave are fewer as compared to that by master and master/slave bridge device. Time taken to form the scatternet can be deduced from the number of messages sent by the master having the maximum message count. From Fig. 7, it is evident that our proposed algorithm scales very well as the number of nodes increases.

Fig. 6. Number of slaves, masters, S/S bridges, M/S bridges formed in the scatternet vs. number of nodes.

Fig. 7. Avg. number of messages sent by a slave, S/S bridge, and master (or M/S bridge) vs. number of nodes in different roles.

Figure 8 shows the per-piconet connections in the scatternet, by plotting the average number of slaves and bridges per piconet. Each piconet has on average, 5-6 slaves and 2-3 bridges. This indicates the efficiency of our algorithm in forming a connected scatternet.
The total number of messages in scatternet formation is presented in Fig. 9. The graph clarifies that the number of messages is linear against the number of devices. This agrees with the $O(n)$ message complexity.

![Fig. 9. Total number of messages to form scatternet for different number of nodes.](image)

VI. REFERENCES


