Distributed, Robust and Self-Organizing Bluetooth Scatternet Formation

Navendu Jain*
*University of Texas at Austin
nav@cs.utexas.edu

Nitin Pabuwal†
†Amazon
npabuwal@amazon.com

B. N. Jain‡
‡Indian Institute of Technology, Delhi
bnj@cse.iitd.ernet.in

Abstract

Bluetooth is a promising low-cost low-power short-range radio technology for wireless personal area networks. The basic network unit in Bluetooth is a centralized master-slave topology named piconet. The piconets can be further interconnected to form a multi-hop ad hoc network known as a scatternet. The properties of scatternets pose new challenges for constructing an efficient topology.

In this paper, we present a novel distributed approach for scatternet formation which works efficiently even when all the devices are not in range of each other. We validate the efficiency of our approach through simulations demonstrating that the scatternet formed has fast connection establishment time and $O(\log n)$ diameter for a network of n devices. For the case of all-in-range devices, our protocol achieves close to the the minimum number of piconets.

Based on the implicit fault-tolerant nature of our protocol, we extend it with a self-healing mechanism to work well under mobile environments. Our simulation results indicate the robust nature of the formed scatternet when nodes join and leave arbitrarily. To the best of our knowledge, this is a first attempt that addresses scatternet formation for dynamic environments as well as when all the devices are not in communication range of each other.

I. Introduction

The ubiquitous use of information intensive consumer devices such as cell phones, personal digital assistants (PDAs) and laptop computers has called for a new networking paradigm to interconnect them. The goal is to create a personal area network (PAN) that accommodates seamless information transfer among devices without the need for manual configuration, cables, or wired infrastructure. In December, 1999, an industry consortium known as the Bluetooth Special Interest Group [15] standardized a short-range, low-power, radio frequency (RF) technology called Bluetooth. The main application scenarios of Bluetooth include ad hoc networking, data access points for Internet access, synchronizing data between PDAs, mobile phones and laptops and interactive multimedia conferences between wireless networked hosts.

Bluetooth uses two fundamental procedures known as *inquiry* for discovering other devices and *paging* to subsequently establish connections with them. During inquiry, a node collects information about the low-level state (native clocks) of its neighbors and uses it in paging to establish a bi-directional communication channel with them. The communication steps during the paging process are similar, except that the paging message is unicast to a selected listener. At the end of paging, a new piconet is formed with the sender and the listener assuming the roles of master and slave respectively. Later, if required, these roles can be swapped.

Each piconet comprises of one master and up-to seven active slaves. To facilitate high densities of communicating devices, a scatternet can be established by sharing slaves among piconets. Many existing scatternet formation algorithms are based on leader election process [6], [1], [8], [9], [7] and/or all-in-range devices [5], [6], [1], [8], [9]. In this paper, we address this issue by proposing a distributed protocol, BTSF (Bluetooth Scatternet Formation Protocol), which works efficiently even when all the devices are not in range of each other. The design of BTSF proposes and justifies a set of desired properties so that the resulting scatternet achieves fast connection establishment and close to the minimum number of piconets.

This paper is organized as follows. In Section II, we discuss the related work on scatternet formation algorithms. In Section III, we present the design of a new distributed protocol for forming Bluetooth scatternets. We evaluate the performance of our approach through simulations and compare it with the existing algorithms in Section IV. Finally, the concluding remarks are given in Section V.

II. Related Work

An ad hoc network based on Bluetooth presents special challenges. The distributed protocols for self-configuring networks [10] become infeasible because they assume a single broadcast channel. An approach towards understanding the topological structure of scatternets is studied in [13]. The authors showed that the space of all scatternet topologies is computationally infeasible to search efficiently without understanding its mathematical structure. They identified the mathematical properties of scatternets and described a technique for enumerating all feasible scatternet topologies. Miklos *et al.*[16] applied heuristics to form scatternets with desirable characteristics. They investigated the possible correlation between scatternet formation criteria and scatternet performance through simulations. Raman *et al.* discussed cross-layer optimizations in Bluetooth Scatternets.

Most of the existing algorithms are based on essentially two scatternet topologies:

- Tree Topology (TTP) (Figure 1)
  TTP assigns master/slave roles to nodes while connecting them in a tree structure. Every bridge (internal node) performs the dual function of a slave in its parent's piconet and as a master in its children's piconet.

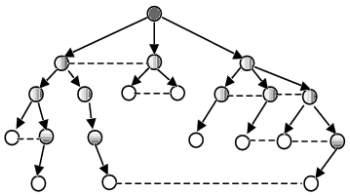

Fig. 1. Tree Scatternet: Hierarchical Network Formation

- Shared Slaves Topology SST (Figure 2)
  In shared slaves topology, the piconets share slaves, i.e. a slave frequently switches between member piconets and is active in only one of them at a time. The shared slave could either be a pure bridge (slave in all member piconets) or alternates being a master and slave in its respective piconets.
  A variant of SST based scatternet topology called *BlueRing* has been proposed in [8], [9] which connects piconets as a circular ring interleaved by bridges between piconets.

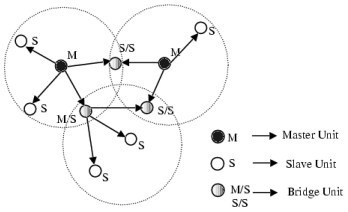

Fig. 2. Shared Master/Slave Scatternet: Flat Model

Both these topology formation schemes have their own pros and cons. TTP is good for networks with frequent broadcasts and avoiding routing loops. It selects the smallest possible number of links to form a connected scatternet. However, it's hierarchical structure lacks reliability since the loss of a parent separates all nodes in its subtree. Routing is inefficient as all the paths have to traverse the tree in upward and downward directions. As a result, parent nodes are likely to become communication "bottlenecks". In SST, the resulting scatternet maintains a balanced structure and is more robust to mobility. But one drawback is the potential formation of routing loops.

A comparison [2] of these two schemes showed that the SST can carry far more communication traffic than TTP. Salonidis *et al.* [1] presented a symmetric link formation protocol - BTCP (Bluetooth Topology Construction Protocol). It has three phases: leader election, scatternet information transfer and scatternet establishment. The scheme assumes all nodes with-in range of each other and satisfies their defined properties up-to 36 devices. Since there is a single elected coordinator, BTCP has more flexibility in constructing the topology. However, the devices need to timeout to restart the election process in case the coordinator fails. BTCP is also unsuitable for dynamic

scenarios when devices join and leave arbitrarily. In Lakshmi *et. al.* [6] approach, the network is first partitioned into separate piconets and then a *super-master* is elected to form the inter-connections. Godfrey *et. al* [3] proposed a TTP based algorithm, TSF (Tree scatternet formation). Their simulations showed that the scatternet had low packet routing latency but a long network establishment time of 80 seconds for 60 nodes. In their work, only tree roots perform device discovery operations, so a scatternet might not be formed when roots can't hear each other. In comparison, our approach allows any node (except a bridge) to connect to all devices with-in its range. Ching *et al.* [5] gave a distributed randomized protocol for all-in-range devices, achieving close to the minimum number of piconets. In their work, devices pick seek (sender) and scan (listener) states randomly at periodic intervals (a round). This leads to long scatternet formation delays since in any round, the number of connections formed would be determined by the smaller of the number of devices running seek and scan. In contrast, in our work, all the devices do INQUIRY and INQUIRY SCAN alternately leading to faster network set-up.

Basagni *et al.* [7] presented a scheme which builds looped network topologies without having the requirement that all the nodes be with-in the transmission range of each other. Schemes given in [8], [9] propose a BlueRing scatternet where all nodes are arranged in a ring structure. In [8], each node participates in the ring acting as a Master-Slave (M-S) relay. In contrast, [9] forms a network in which the ring comprises of only the masters and their bridges with the neighboring piconets. The ring structure is unsuitable due to the large network diameter and interference arising out of the large number of formed piconets. In dynamic scenarios, the network partitions would persist for long durations until the ring structure is restored. Also, these protocols would be sensitive to the value of the time-out parameter in order to avoid forming sub-rings before termination.

Zaruba *et al.* [4] introduced "Bluetrees" with both centralized and distributed variants. The former builds a scatternet starting from some specified node called Blueroot, while the latter speeds up the scatternet formation process by selecting more than one root for tree formation and then merging the trees generated by each root. Bluenet [2], a SST based protocol, separates the phases of intra and inter-piconet connection formations. In contrast, they occur simultaneously in our algorithm leading to faster scatternet formation.

III. Bluetooth Scatternet Formation Protocol

In this section, we formulate a SST based scatternet formation algorithm. The usage models of our protocol include starting with isolated devices i.e. the *en masse* arrival and the *incremental* joining/leaving of the nodes within the existing core network.

Our protocol design is governed by the following properties that the resulting scatternet would satisfy:

1) No master-slave bridges are allowed i.e. only slave-slave bridges. There is a bandwidth loss since the devices can't communicate when their master is listening in the piconet for which it is a slave.
2) A bridge node connects only two piconets [1] to minimize switching overheads. Since a portable device has limited processing capabilities, a maximum bridge degree of two relieves a node of being overloaded (no bottlenecks). Routing becomes simplified as the bridge node receives packet from one master and sends to the other (if it is not the destination) with no computation overhead.
3) Two piconets share at most two bridges. Bi-connectivity makes the network more tolerant to mobility. As long as one of the links is active, the connectivity is maintained. This connectivity number between two piconets represents a trade-off between maintaining extra inter-piconet links on one hand and being fault-tolerant to mobility on the other. We chose *two* based on the simulation studies which are presented later in the paper.
4) The scatternet consists of minimum possible number of piconets [5], [1] so as to reduce inter-piconet interference. Since all piconets share the same set of 79 channels, there would be high collisions and burst failures as the number of piconets increases.
5) Balanced distribution of the network resources (e.g. power) within the scatternet.
6) Fault-tolerant under mobile environments.

A. Algorithm

The connectivity metric for the BTSF algorithm is shown in the link formation table I. Only the pairs with entries marked 1 are allowed to form connections. Due to all non-zero entries for a free device, it is guaranteed a faster connection establishment with the existing network. BTSF consists of a single phase with multiple rounds of connection formation. At the start of each round, the network comprises of components; each component is either

TABLE I
LINK FORMATION COMBINATION: ENTRIES WITH 1 ARE ALLOWED; BRIDGES DON'T FORM NEW CONNECTIONS

Device Status	Master	Slave	Free
Master	0	1	1
Slave	1	0	1
Free	1	1	1

a free device or a set of connected devices i.e. a piconet or a scatternet. During a round, new connections get established, existing piconets merge (if possible) and components interconnect with each other to form a bigger component, all these operations occurring in parallel.

Each device maintains a set of data structures corresponding to its *status* in the scatternet. A master node stores information about it's piconet members in *slaves* list. For a slave or bridge node, *master1* and *master2* denote the two masters in order with whom it is connected.

In BTSF (Figures 3, 4, 5, 6), each device (initially having zero knowledge of surroundings) performs INQUIRY and INQUIRY SCAN alternately with uniformly distributed state residence times similar to [1]. For every inquiry or response(s) (to it's own inquiry) received during INQUIRY SCAN, a node updates a *visibility graph* of devices within its radio range. This graph also stores the status of the neighborhood devices. When a connection gets formed, the two nodes exchange their status. We next, describe the protocol actions corresponding to the four non-zero (unique) pairs in the Table I.

A free node u uses inquiry and paging to connect to neighboring devices. If the discovered device v is also a free node, a new piconet is formed. In case v is a master having less than 7 slaves, u joins v's piconet. And if v is a slave, a temporary piconet gets formed with u as the master and v as the common slave (bridge). v then initiates a CONNECT procedure by sending the clock information of u to it's original master w. If w has less than 7 slaves and u lies within its range (visibility graph), then the temporary piconet breaks and u joins w's piconet. Otherwise, the piconet between u and v stays connected.

Once a node becomes a piconet master, it needs to perform the dual functions of facilitating intra-piconet communication as well as connecting to new and existing components. In our algorithm, to simplify the protocol design, a master node performs inquiry and paging procedures after every cycle of polling all of its slaves. The idea behind this is that as the piconet size increases, the master should progressively spend more time communicating with it's slaves. At the same time, it should also work towards connecting new devices.

When u starts as a master and the found device v is a slave, v becomes a bridge node if there are less than two bridges between these two piconets. A MERGE procedure (CONNECT is a special case of MERGE) is initiated to merge the piconet having smaller number of slaves into the other governed by the visibility graph of the slaves in u and v's piconets and the maximum 7 (active) slaves per piconet constraints. The noteworthy point here is that the decision process for merge only executes at the two masters i.e. u and v's master (say w). (Assume w.l.o.g. that u's piconet has lesser number of slaves) The actual merge proceeds via u sending the network ids and clock information of itself and its slaves to w through v. Later, u sends k (determined by MOVE procedure called from MERGE) slaves into PAGE SCAN mode to listen to w's page messages (The actual migration of slaves from u's piconet to that of w is done using the MOVE procedure). This process alleviates the need for doing time-intensive inquiry (in the order of seconds) again compared to paging (in the order of milliseconds). During the decision process, if it turns out that both the piconets have equal number of slaves, the master having more of other piconet's slaves in its communication range (visibility graph) moves them into it's own piconet.

Whenever two masters (or slaves) join for the first time, each updates its visibility graph and terminates the connection. Upon receiving INQUIRY by the same device again, no reply is sent (looking at the visibility graph). Due to mobility, the status of a device may change as the network topology changes. Instead of using costly physical timers to purge the old entries, a counter is incremented based on number of query messages received from that device. When the counter overflows for an entry, it is deleted from the visibility graph so that the two devices could re-connect again. It might happen that the scatternet remains as a forest when the further connections are possible only between the two masters (slaves). This situation, though very unlikely, is a trade-off for not allowing master-

```
Algorithm BTSF(device u)
// The BTSF algorithm running at device u
    u performs device discovery using INQUIRY and PAGE procedures
    if a device v is found
     switch (u.status)
        case FREE:
          switch (v.status)
             case FREE:
               u.status⟵MASTER; v.status⟵SLAVE;
               u.slaves→add(v); v.master1⟵u; break;
             case MASTER:
               u.status⟵SLAVE; u.master1⟵v; v.slaves→add(u); break;
             case SLAVE:
               CONNECT(u,v); break;
          break;
        case MASTER:
          switch (v.status)
             case FREE:
               v.status⟵SLAVE; v.master1⟵u; u.slaves→add(v); break;
             case MASTER:
               u.vis_graph→add(v); end connection; break;
             case SLAVE:
               MERGE(u,v,v.master1); break;
          break;
        case SLAVE:
          switch (v.status)
             case FREE:
               CONNECT(v,u); break;
             case MASTER:
               MERGE(u.master1,u,v); break;
             case SLAVE:
               u.vis_graph→add(v); end connection; break;
          break;
```

Fig. 3. Pseudo code for the BTSF Algorithm

```
CONNECT(device u, device v)
//u  is a free node and v is a slave
   u ⟵MASTER; v ⟵BRIDGE; v.master2⟵u; u.slaves→add(v)
   v sends u's clock information to it's master1, say w
   if (w.vis_graph→lies(u) && w.slaves→num < 7)
     {
       v sends u to PAGE SCAN mode; the piconet of u and v breaks;
       w sends a PAGE to u and adds u to it's piconet;
     }
```

Fig. 4. Pseudo code for the CONNECT procedure

slave bridges for simplicity and performance. The above counter logic can be extended by allowing a temporary master-slave bridge on counter overflow, to be broken as soon as a slave-slave bridge is formed between their piconets.

B. Mobile Environments

Under dynamic environments, a scatternet formation protocol needs to be robust with respect to devices joining and leaving arbitrarily. Based on its implicit fault-tolerant nature, we have extended our protocol to work in mobile

```
MERGE(device u, device v, device w)
// u, w are masters and v is the common slave (bridge)
// Initially, both u and w exchange their slave list through v
   V1⟵ {x : x ∈ (({u} ⋃ u.slaves)\{v}) && w.vis_graph→lies(x)}
   V2⟵ {y : y ∈ (({w} ⋃ w.slaves)\{v}) && u.vis_graph→lies(y)}
   if (u.slaves→num == w.slaves→num)
     {
       if (|V1| ≥ |V2|) //move slaves from u to w
         MOVE(u, w,V1);
       else
         MOVE(w, u,V2);
     }
   else if (u.slaves→num < w.slaves→num)
      MOVE(u, w,V1);
   else
      MOVE(w, u,V2);
```

Fig. 5. Pseudo code for the MERGE procedure

```
MOVE(device u, device w, Set V)
// u, w are masters and V is the set of u's piconet members in communication range of w
   k ⟵ min(7 - w.slaves→num , |V|);
   if (k == u.slaves→num)
     u and all it's slaves join w's piconet;
   else // u needs to stay as the master
     w's piconet takes k (or 1 less) nodes from (V\{u});
```

Fig. 6. Pseudo code for the MOVE procedure

scenarios. BTSF, as described in the previous section, efficiently handles the events of new (free) devices joining by providing a fast connection establishment with the existing core network.

Due to mobility, slaves in a piconet can move out of hearing range of other masters while still being connected to their own master. The MERGE procedure needs to be modified to handle this particular case. During MERGE, if both the piconets can completely merge into a single piconet (say u, w are masters and u's piconet gets merged into w's), w would PAGE u only after it has previously connected to all of u's slaves. In case some slave(s) couldn't be contacted, w forms a temporary piconet with u and informs it about them. w then breaks the piconet and updates its visibility graph. Later, u sends a PAGE message to these remaining unconnected slaves and restarts its piconet.

We next, describe the protocol's self-healing actions arising out of devices failing or leaving the scatternet.

- Bridge Failure:
  In BTSF, since any two piconets can share two common bridges, the connectivity is maintained as long as one of the links is active. This bi-connectivity allows packets to route through the active link without an increase in the hop-counts for routes between nodes. To restore this bi-connectivity after one of the bridge nodes fail, both the masters (of the failed bridge) allow their unshared slaves to connect with the other master.
- Master Failure:
  If the failure point is a master, all its unshared slaves would become disconnected from the network. For the case of all-in-range devices, a master can designate a secondary master among its pure slaves, who takes on the resposibility in case the original master fails. But, for the general case, we can't assume that the slaves can hear each other. To keep the protocol simple, the disconnected unshared slaves restart as free nodes to form connectivity with the existing network.
- Pure Slave Failure:
  No protocol action needs to be taken when an unshared slave fails. As described in the algorithm description, it's master would automatically start spending more time to look for new connections.

In a dynamic environment, where the total number of Bluetooth units may substantially vary, the above mechanism ensures that the changes affect only the local neighbourhood without propagating across the entire scatternet.

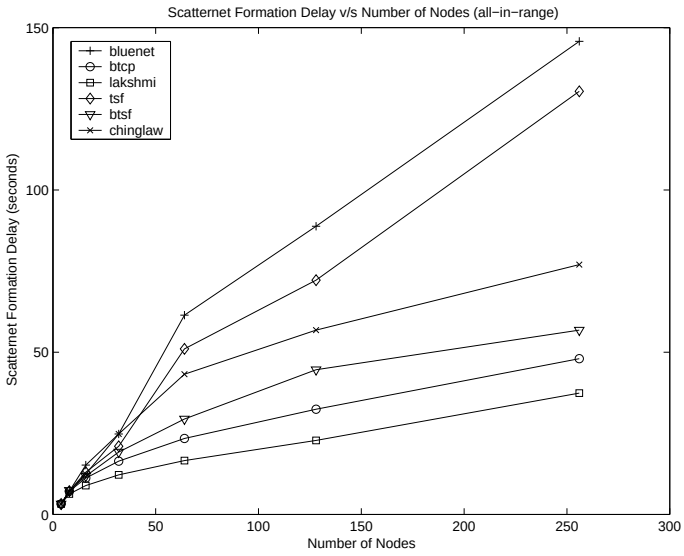

Fig. 7. Scatternet Formation Delay (all-in-range)

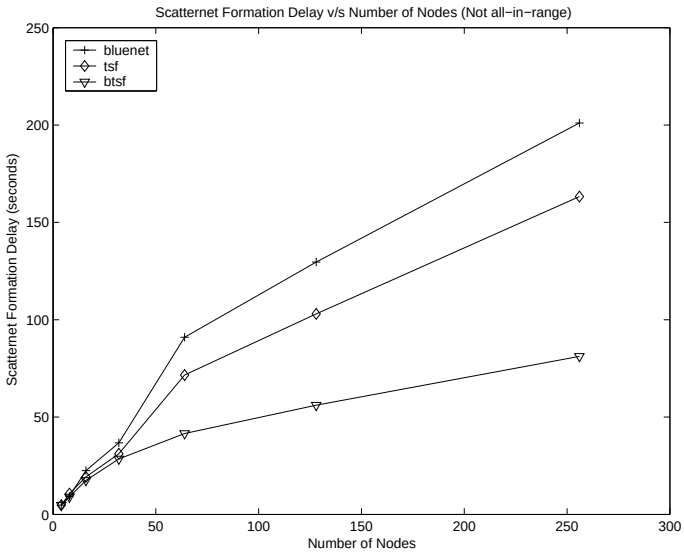

Fig. 8. Scatternet Formation Delay (not all-in-range)

IV. Results and Analysis

We have developed a matlab based simulator based on discrete event simulation of asynchronous processes (i.e. nodes are not simultaneously turned "on") for evaluating the performance of scatternet formation algorithms.

Each Bluetooth unit is implemented as a process consisting of the Baseband and the BTSF modules. The Baseband module does emulation of the inquiry and paging processes. The BTSF protocol runs at each node and establishes connectivity with the scatternet using Baseband as the underlying module.

The comparison analysis is based on the following metrics:

1) Scatternet formation latency is an important measure as users typically require fast connection establishment. Given a protocol, this delay corresponds to the time duration between the en-masse arrival of nodes till there exists atleast one path between any two nodes.
2) Network diameter is a measure of the longest path between any two nodes in the topology. This gives an estimation of the maximum routing delay in the network. For minimal delays, the diameter should grow logarithmically ($O(\log\ n)$) for n devices.
3) The number of piconets is an efficiency measure since the end-to-end communication delay for packets depends on the number of piconets traversed. It also determines the number of bridges required to maintain connectivity. For a scatternet of n all-in-range devices, the lower bound on the number of piconets is $\lceil(n$ - 1)/7$\rceil$ [5].
4) Node capacity, based on the metric given in [2], denotes the information carrying capability of the network. It gives the maximum rate at which packets can be pumped into the network, subject to the average bridge (inter-piconet) and the intra-piconet scheduling delay, perceived by a transmission. The bridge overhead corresponds to the bridge node's switching delay between member piconets. The intra piconet overhead is associated with the master node polling and coordinating among all the slaves in its piconet. For most cases, the former overhead is much larger compared to the latter.

   The capacity of node i is given by:

$$C_i = C_0 - n_s^i.\triangle B_1 - I_{bridge}^i.(n_p - 1).\triangle B_2 \quad (1)$$

   where $C_0 = 1000kbps$ is the link capacity, n_s^i is the number of slaves (if i is master, 0 otherwise), $\triangle B_1 = 10kbps$ gives the intra piconet overhead for each slave held by a master, I_{bridge}^i is 1 (if i is bridge else 0), $\triangle B_2 = 100kbps$ is the bridge overhead for joining piconets and n_p (> 1) is the number of piconets that a bridge node connects.

Since current algorithms assume that all devices are in-range, the simulations were done under two scenarios, (1) all-in-range devices and (2) some devices are out of communication range of others. In the former case, n devices

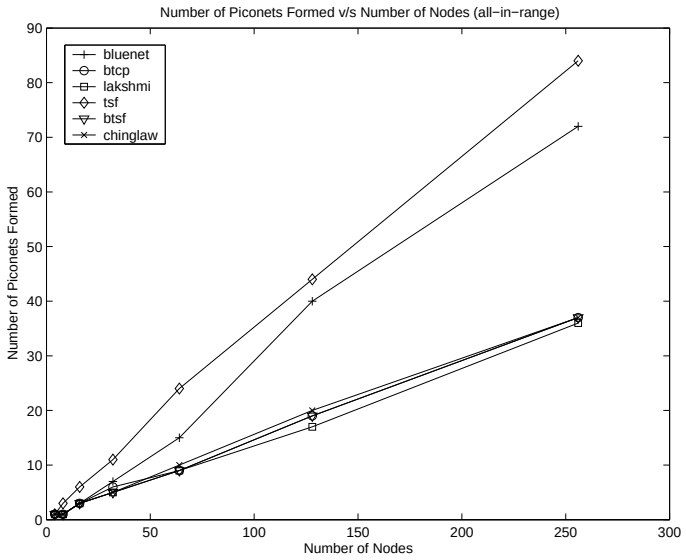

Fig. 9. Number of piconets (all-in-range)

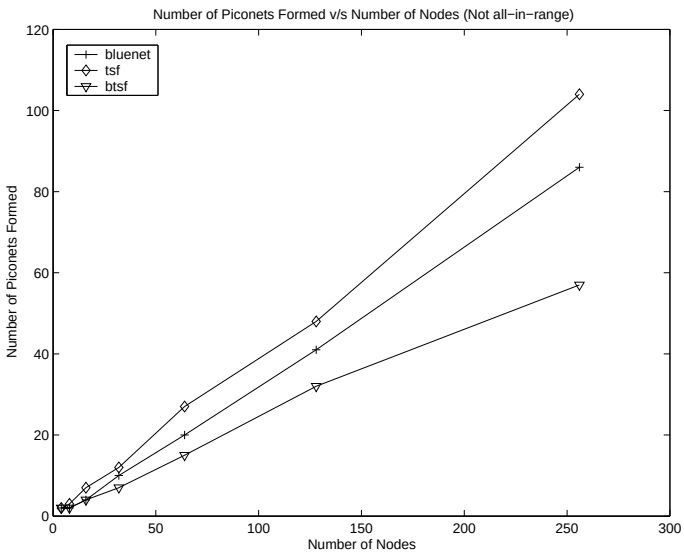

Fig. 10. Number of piconets (not all-in-range)

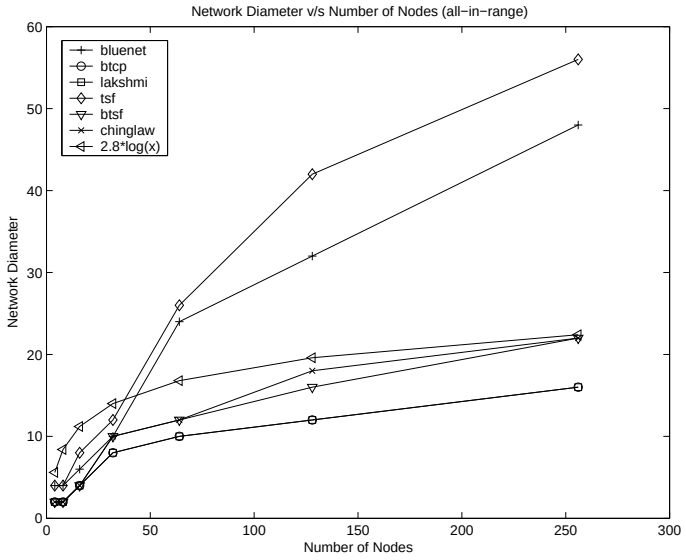

Fig. 11. Network Diameter (all-in-range)

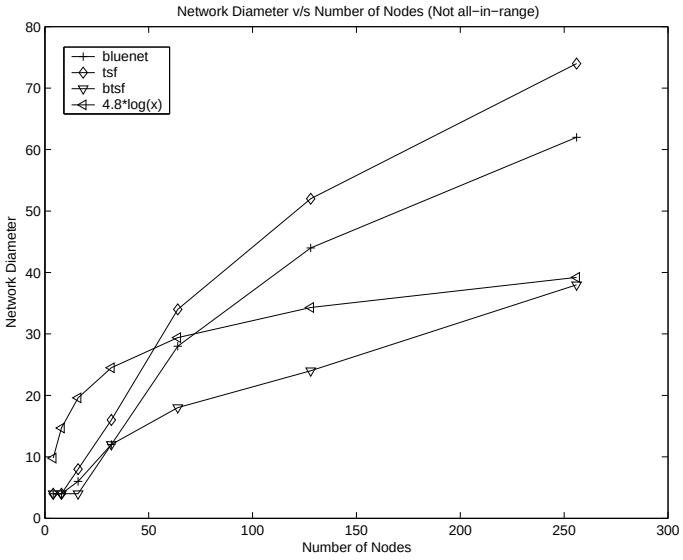

Fig. 12. Network Diameter (not all-in-range)

are randomly located within an area such that any two devices are with-in the Bluetooth radio coverage (assumed 10 mts). For the latter, a node is placed so as to lie with-in the range of at-least two and on an average, four other devices. For both these cases, the graphs have been plotted by taking the average of 50 independent experimental trials.

A. Simulation Results

The simulations showed that with a small trade-off in the scatternet formation delay, BTSF achieves the optimal network topology in terms of minimum number of piconets and the network diameter.

BTSF's scatternet formation latency is close to the minimal time taking algorithm (Figure 7) whose time intensive part is the leader election process. Since BTSF is completely distributed, there is a small overhead compared to the centralized algorithms. When not all devices are in-range, BTSF shows a significantly small topology formation delay compared to TSF and Bluenet (Figure 8). The number of piconets (Figure 9) formed by BTSF is close to the minimum number achieved by the centralized algorithms where the central coordinator determines the optimal topology. In the general case (Figure 10), the number of piconets is about half of Bluenet showing the clear

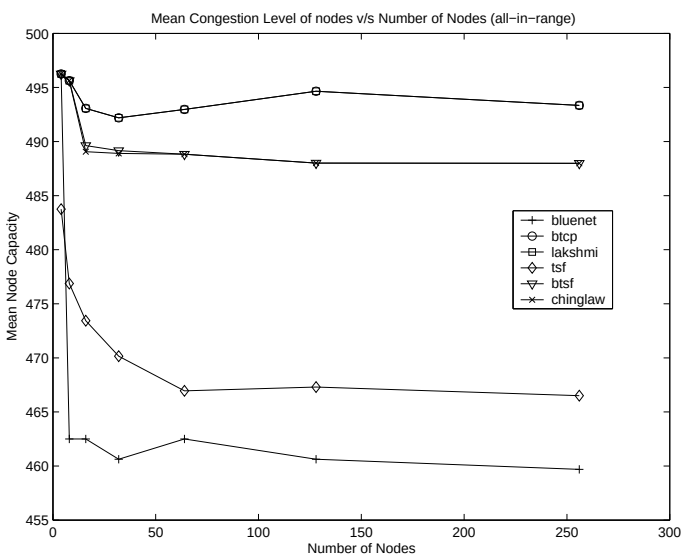

Fig. 13. Number of Nodes v/s Mean Node Capacity

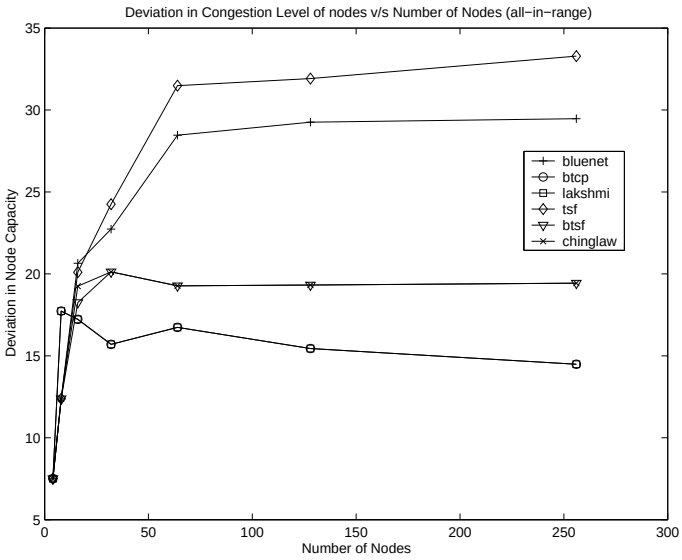

Fig. 14. Deviation in Node Capacity v/s Number of Nodes

advantage of the merging process in BTSF. The resulting network diameter (Figures 11, 12) of BTSF formed scatternet equals that of centralized algorithms, $O(\log\ n)$ for n devices.

The two parameters linked with the network capacity are the mean and the deviation of node capacities. Since BTSF tries to establish more than one connection with neighboring piconets, it uses more network resources but it turns out to be a small difference compared to the algorithm yielding the highest mean node capacity (Figure 13). This is a small overhead in return for the enhanced network connectivity and robustness against node and edge failures.

BTSF, being inherently distributed, uniformly distributes the network resources. Figure 14 shows that BTSF is second best to the centralized schemes in terms of capacity deviations where a coordinator does the allocation in the best way.

B. Mobile Scenarios

For dynamic environments, a scatternet network would pre-exist into which new nodes would be joining and existing nodes leaving at arbitrary times. The experimental set-up started with an exisiting scatternet of n devices. About $n/2$ nodes were randomly picked to leave and join the scatternet over a period of 120 seconds so that the total number of nodes in the network remains close to n.

We first explain the design rationale behind sharing two slaves between piconets. In order to determine a good value of connectivity number, we performed the experiment with values from 1 to 3. First, a 64 node scatternet was built taking each of the three values. We then randomly genearated and stored a pattern of node arrivals and departures as described above. The same pattern was input to all the four cases and the number of network partitions were computed after every sample of 4 seconds. The results in Figure 15 showed that the network is highly prone to being disconnected when only one slave is shared between any two piconets. When nodes are not all-in-range, a device would be connected to the network through its neighboring piconets. And when this single link fails, network partitions would occur frequently.

Figures 16, 17 show that bi-connectivity between piconets makes the network more robust in comparison to having a single link between them. Moreover, the performance is similar when more than two connections are maintained considering the cost of having redundant inter-piconet links. In a real setting, traffic spanning two piconets would be disrupted if a failed bridge lies on the current route and there is no alternative path available. Using the other active bridge, the traffic can be re-routed between the two piconets while the scatternet restores bi-connectivity.

We investigated the fault-tolerance of BTSF based on the resulting network diameter and number of network partitions under dynamic environments. The experiment was done taking n as 32, 64 and 128. In the case of

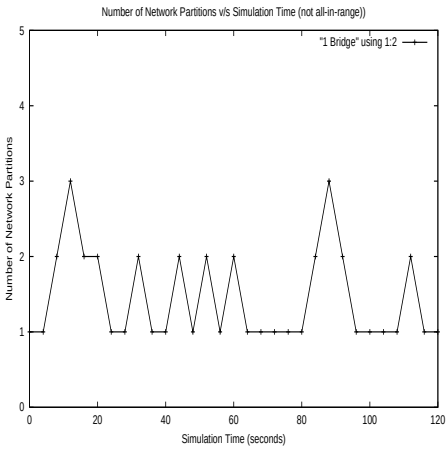

Fig. 15.

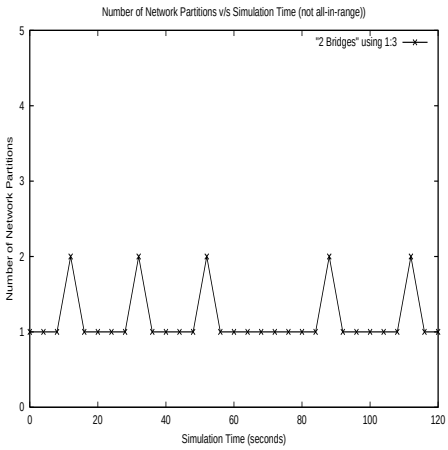

Fig. 16.

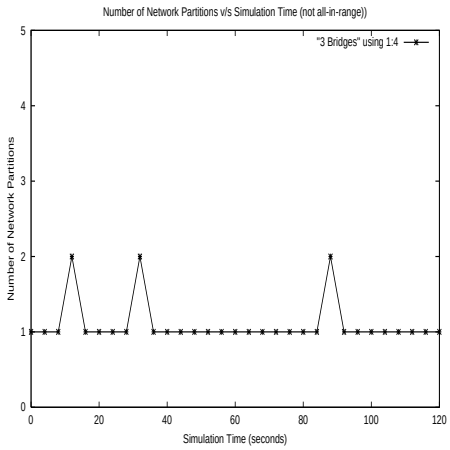

Fig. 17.

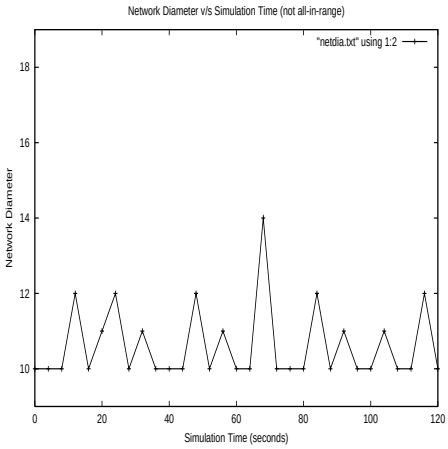

Fig. 18. 32 nodes

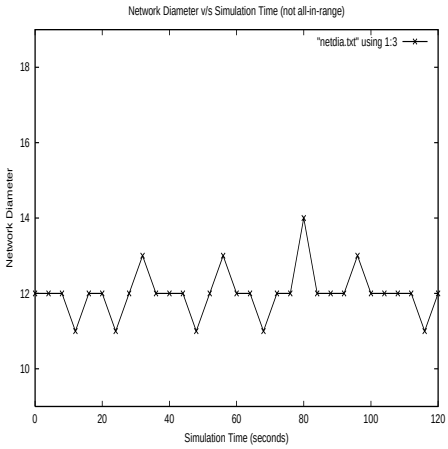

Fig. 19. 64 nodes

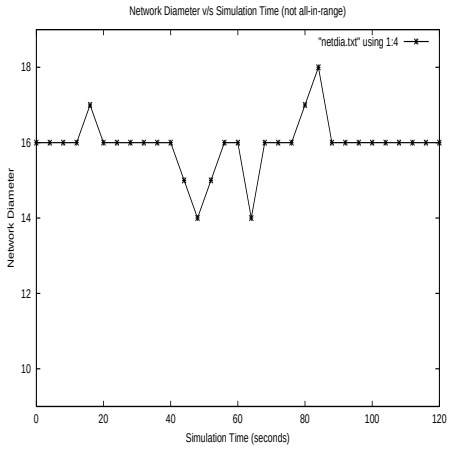

Fig. 20. 128 nodes

network diameter, we only consider a single component state of the network i.e. one connected scatternet since the diameter is undefined when there are partitions. Figures 18, 19, 20 illustrate that the network diameter changes by a small amount in spite of the dynamic network. This clearly shows the benefits of allowing two bridges between two piconets. For some cases, it even decreases when a shorter path is found. The experimental results for network partitions are given in Figures 21, 22, 23. Again, the bi-connectivity masks the changes arising out of mobility to maintain the entire network in a connected state.

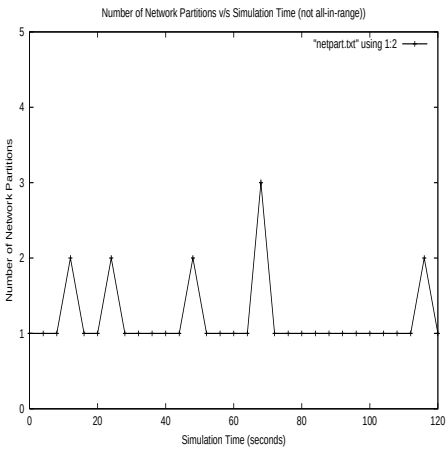

Fig. 21. 32 nodes

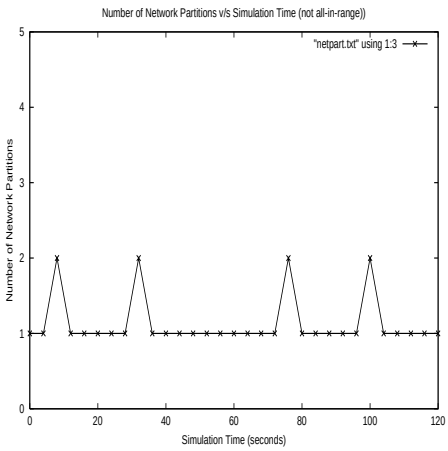

Fig. 22. 64 nodes

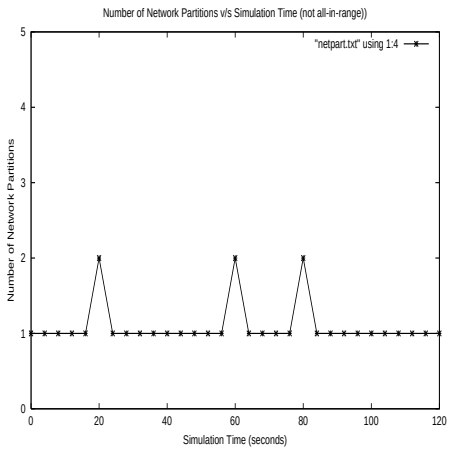

Fig. 23. 128 nodes

C. Architecture Framework and Implementation

Nitin *et. al.* designed an architectural approach [11] for modular design and seamless integration of various systems (routing, topology formation, APIs etc.) for proper functioning of applications over Bluetooth. The architecture is completely platform independent and supports portable and plug-n-play scatternet formation and routing modules, built over J2ME and JSR-82 (Java Specification Requests for Bluetooth). In our opinion, the development of a standard architecture is important so that new scatternet formation and routing algorithms could be developed independently and yet, lead to co-operating systems.

We have implemented BTSF as a pluggable scatternet formation algorithm module and have used it with a multimedia application built in complete accordance with this architecture. The BTSF module implements *Scatternet* and *RouterListener* interfaces, which are part of the proposed API [11] for interacting with the application and router modules respectively. The module also implements *DiscoveryListener* which is a JSR-82 interface for receiving device and service discovery events. When asked by an application module to join a scatternet, the module creates an instance of the mentioned router to be used, gets into alternate inquiry and inquiry scan mode and starts a thread to synchronously accept incoming connections. All new connections are handled precisely in the way defined by BTSF. The status of the device in the scatternet and the direct connections it has with other devices are stored in internal variables that are continously updated, depending on addtion of new connections or loss of old connections as notified by the router module. The application module may choose to cancel the joining process anytime before the device joins a scatternet. Once the device is in a connected state, the application may anytime choose to leave the scatternet, as a result of which the router is asked to shutdown and all internal state variables are purged and the module gets ready afresh to re-join a scatternet if so asked by an application module. Due to the unavailability of Bluetooth modules supporting piconet switching functionality (bridges), we used the Impronto simulator [14] developed by Rococo to emulate this functionality for the multimedia application. Though no experiments were done on the emulated set-up, our algorithm achieved good perceptual performance for data, audio and video streaming applications. We have also implemented BTSF over BlueZ [12], the official Bluetooth stack for Linux.

V. Conclusion

In this paper, we have presented a new scatternet formation algorithm which is completely distributed in nature as well as robust to mobility. Our simulation results show that BTSF achieves fast connection establishment and close to minimum (possible) number of piconets. In addition, the diameter of the resulting scatternet is $O(\log\ n)$. An important feature of our algorithm is that all the devices need not be in radio range of each other. Any device with-in the range of at-least one of the nodes (in the core network) gets connected quickly.

We are currently working towards theoretically proving the bounds as illustrated by the simulations. We have some preliminary proofs that validate our simulations. We would be incorporating the complete analysis in the final version of this paper.

References

[1] T. Salonidis, P. Bhagwat, L. Tassiulas and R. LaMaire, *Distributed Topology Construction of Bluetooth Personal Area Networks*, IEEE INFOCOM, 2001.

[2] Zhifang Wang, Robert J. Thomas and Zygmunt Haas, *Bluenet - a new scatternet formation scheme*, Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS), 2002.

[3] Godfrey Tan, Allen Miu, John Guttag and Hari Balakrishnan, *Forming Scatternets from Bluetooth Personal Area Networks*, MIT Technical Report, MIT-LCS-TR-826, 2001.

[4] Gergely V. Zaruba, Stefano Basagni and Imrich Chlamtac, *Bluetrees-Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks*, Proceedings of IEEE International Conference on Communications, pages 273-277, 2001.

[5] Ching Law and Kai-Yeung Siu, *A Bluetooth Scatternet Formation Algorithm*, Proceedings of the IEEE Symposium on Ad Hoc Wireless Networks 2001, San Antonio, Texas, USA, 2001.

[6] Lakshmi Ramachandran, Manika Kapoor, Abhinanda Sarkar and Alok Aggarwal, *Clustering Algorithms for Wireless Ad Hoc Networks*, Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Pages 54-63, Boston, MA, 2000.

[7] S. Basagni and C. Petrioli, *A Scatternet Formation Protocol for Ad hoc Networks of Bluetooth Devices*, IEEE Vehicular Technology Conference, Pages 424-428, 2002.

[8] C.C. Foo and K.C. Chua, *BlueRings - Bluetooth Scatternets with Ring Structures*, IASTED International Conference on Wireless and Optical Communication (WOC 2002), Banff, Canada, 2002

[9] T.Y. Lin, Y.C. Tseng and K.M. Chang, *Formation, Routing, and Maintenance Protocols for the BlueRing Scatternet of Bluetooths*, Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS), 2003.

[10] Nancy Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers 96.
[11] Nitin Pabuwal, Navendu Jain and B. N. Jain, *An Architectural Framework to deploy Scatternet-based Applications over Bluetooth*, Proceedings of IEEE International Conference on Communications (ICC), 2003.
[12] Maxim Krasnyansky, BlueZ, The official Bluetooth stack for Linux [Online] *http://bluez.sourceforge.net*
[13] Pravin Bhagwat and Srinivasa Rao, *On the characterization of Bluetooth scatternet topologies*, Submitted for Publication, 2001.
[14] Impronto Simulator 1.1, Rococo Software Ltd. [Online] *http://www.rococosoft.com/products/*
[15] Bluetooth Special Internet Group Document (SIG), *Specifications of the Bluetooth System - Version 1.1B*, http://www.bluetooth.com, 2001.
[16] G. Miklos, A. Racz, Z. Turanyi, A. Valko and P. Johansson, *Performance aspects of Bluetooth scatternet formation*, Proceedings of First Annual Workshop on Mobile and Ad Hoc Networking and Computing, 2000.
[17] Bhaskaran Raman, Pravin Bhagwat and Srinivasa Seshan, *Arguments for Cross-Layer Optimizations in Bluetooth scatternets*, Proceedings of Symposium on Applications and the Internet, pages 176-184, 2001.