

A Bluetooth Scatternet Formation and Healing Protocol for Ad-Hoc Group Collaboration

Dimitris N. Kalofonos
Nokia Research Center
5 Wayside Road
Burlington, MA 01803
dimitris.kalofonos@nokia.com

Somil Asthana
Department of Computer Science
State University of New York
Buffalo, NY 14226
asthana@cse.buffalo.edu

Abstract— In this paper, we present a new tree-based Bluetooth (BTH) scatternet formation and healing protocol called *GC-tree*. Our protocol enables group collaboration, e.g. wireless ad-hoc meetings, where individuals bring piconets with their devices in a scatternet to collaborate. We present experimental performance results from a prototype implementation using off-the-self BTH v1.1 hardware and Linux-based devices. Finally, we complement our experimental results with a realistic simulation model.

I. INTRODUCTION

In this paper we design a scatternet protocol, referred to as *GC-tree*, to work in dynamic environments where entire piconets or single nodes arrive and leave arbitrarily. The protocol incrementally builds a tree topology and heals partitions as they occur. We chose a tree topology because, despite its shortcomings, it is a loop-free topology.

Our protocol enables scenarios, such as wireless ad-hoc meetings, where individuals want to participate with their piconets of devices in a scatternet to collaborate, while maintaining sessions and existing security associations among their personal devices [1]. This would prevent devices to attach arbitrarily in a scatternet as previous proposals assume. Furthermore, unlike prior efforts which focus on simulation results using sometimes unrealistic assumptions, we attempt to bridge the gap between design and analysis of our protocol with simulations on one side, and the implementation and experimental performance evaluation on the other.

The rest of the paper is organized as follows: Section II reviews related work in the literature; Section III presents the scatternet formation and healing protocol; Section IV presents experimental and simulation results; finally, Section V presents our conclusions.

II. RELATED WORK

Scatternet protocols can be categorized as static (e.g. [2], [3], [4]) or dynamic (e.g. [5], [6]). Also, protocols can be characterized as centralized (e.g. [3]) or decentral-

ized (e.g. [2], [5]). Finally, protocols can be categorized based on the resulting topology. A mesh topology (e.g. [3], [7]) is more robust than a tree topology (e.g. [2], [5]) and avoids bottlenecks, but incurs a substantial routing and inter-piconet scheduling overhead. This is one of the main reasons why many approaches lead to a tree topology. BlueTree [2] is a static distributed tree topology protocol which requires each node to have prior knowledge of its neighbors. TSF [5] is a dynamic distributed tree topology protocol which allows nodes to join or leave arbitrarily. A similar approach was followed in [6], which added features to avoid loops and help healing.

III. PROTOCOL DESCRIPTION

A. Design Goals

In designing GC-tree we set the following design goals: scatternet formation starts with preconfigured piconets and restricts the nodes from connecting arbitrarily; new nodes can join the scatternet by invitation only; scatternet formation should involve minimal (if any) user interaction; partitions should heal automatically with a high probability and transparently to the users; unless there are network partitions, the nodes dedicate if possible all of their energy in communicating with each other; the protocol should allow for simple routing; finally, the protocol should be BTH standard compliant.

B. Scatternet Formation

Our design goals require user-piconets as the scatternet's building blocks. This prevents the use of a random scatternet formation strategy because nodes belonging to a piconet cannot connect arbitrarily. To address this issue our protocol allows only piconet masters, referred to as *Pico-Heads (PH)*, to participate in the scatternet formation. The piconet slaves are unaware of the scatternet, except in the cases of scatternet healing when their master leaves, as will be described in Section III-C. For simplicity, in the following description we assume that no partitions are created during formation; this is dealt with

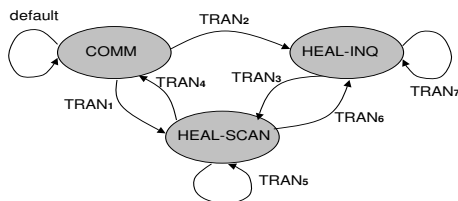


Fig. 1. Node state transition diagram.

by the healing mechanism described in Section III-C.

Our scatternet formation protocol assumes that a user takes an action (e.g. with a GUI) to initiate the scatternet formation. The PH of that user’s piconet becomes the scatternet’s ROOT and initiates the construction of the tree topology by starting scanning. Apart from bootstrapping the process the ROOT has no other special role. Other users wanting to participate also need to take an action. The PHs of those users’ piconets start inquiring and look for a scanning node to attach. On successful inquiry a PH pages and connects to a scanning node in the scatternet and performs a role switch to become the slave in the newly formed connection. Upon attachment to the scatternet each PH starts scanning and becomes a possible attachment point for other free PHs trying to connect. The ever increasing number of possible attachment points accelerates the scatternet formation. This process is incremental and involves a growing *main tree* that contains the ROOT and a number of free PHs trying to attach to it. We assume that each node stops scanning after a sufficiently long timer T_F expires.

C. Scatternet Self-Healing

Unless a partition occurs, all PHs are in the *COMM* state and dedicate all their time-slots in communication with other nodes. When a partition occurs, PHs detect broken links and trigger the healing mechanism. Link-level broadcast messages are used to bring any PHs that remain connected in one of two healing states (*HEAL-SCAN* and *HEAL-INQ*), during which they take part in the healing mechanism by setting a sufficiently long timer T_H . The state transition diagram among the three states is depicted in Figure 1 and explained below.

The general idea of the healing mechanism is that PHs which are part of the main scatternet tree (containing the ROOT) are in the *HEAL-SCAN* state and perform scanning to become possible attachment points. On the other hand, PHs which are part of partitioned sub-trees trying to attach back to the main tree are in the *HEAL-INQ* state and never perform scanning; only the root of each such sub-tree performs inquiry looking for scanning PHs in the main tree to attach. In the rest of this section we describe this idea in detail. Note that the ROOT PH is treated as a separate case below so, unless explicitly mentioned, departing nodes don’t include the ROOT.

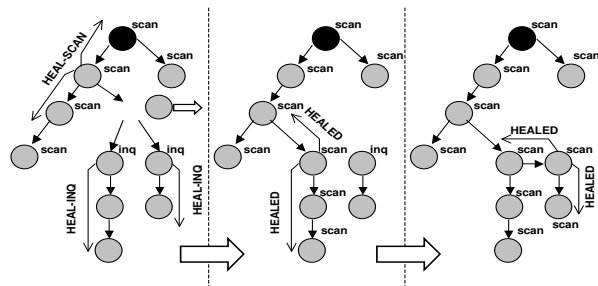


Fig. 2. A partition occurs when the scatternet is fully connected.

We first consider the case where the partition occurs as a one-time event because a piconet (i.e. user) leaves while the scatternet is fully connected (all nodes in *COMM* state); no further partitions occur until healing completes. When a piconet leaves, the event is detected by the parent of the departed PH and all its PH children (referred to as *orphan PHs*). The parent node broadcasts a *HEAL-SCAN* message, changes to the *HEAL-SCAN* state ($TRAN_1$)¹ and starts scanning. All PHs receiving the *HEAL-SCAN* message do the same except rebroadcasting the message. The orphan PHs, on the other hand, are temporarily the roots of smaller disconnected subtrees which try to connect back to the scatternet. Each orphan PH broadcasts a *HEAL-INQ* message, changes to the *HEAL-INQ* state ($TRAN_2$) and starts inquiring. Any other PH receiving a *HEAL-INQ* message does the same, but does not start inquiry. With all the nodes of the main tree scanning, the orphan PH connects back to the tree with high probability within a period much shorter than T_H . Upon connecting back to the main tree the ex-orphan PH broadcasts a *HEALED* message, changes to the *HEAL-SCAN* state ($TRAN_3$), and starts scanning too. Upon receiving a *HEALED* message, the PHs of the main tree which were in the *HEAL-SCAN* state ignore it, while the PHs which were part of the partitioned sub-tree in the *HEAL-INQ* state, change to the *HEAL-SCAN* state ($TRAN_3$) and start scanning too. Eventually, when the timer of each node in the *HEAL-SCAN* state expires it changes into the *COMM* state ($TRAN_4$). The case described above is illustrated in Figure 2. Note that late arrivals of users joining an already formed scatternet fall in the case described above. The new user has to take an action to join the scatternet, which makes that PH act as if it had become an orphan PH. For the main tree, based on our design goals, we chose to have any of the users already in the group take an “update” action. That PH then acts as if it had just lost one of its children.

We then consider the case when either a node belonging to the main tree or a node in a partitioned sub-tree depart, while healing is still taking place. If a PH of

¹ $TRAN_i$ $i = 1, \dots, 7$, denote state transitions shown in Figure 1.

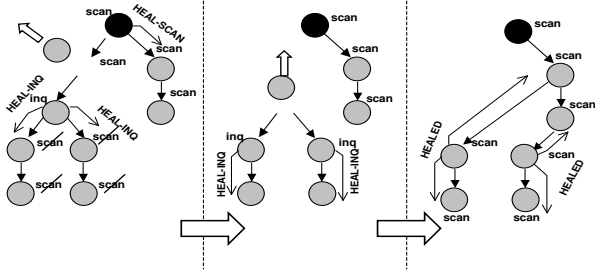


Fig. 3. New partition occurs while scatternet is still healing.

the main tree in the HEAL-SCAN state loses any child, it broadcasts a HEAL-SCAN message, remains in the HEAL-SCAN state ($TRAN_5$) and continues scanning. Any other PH in the main tree receiving a HEAL-SCAN message while in the HEAL-SCAN state does the same except rebroadcasting the message. If a PH while in the HEAL-SCAN state becomes an orphan PH, it broadcasts a HEAL-INQ message, changes its state to HEAL-INQ ($TRAN_6$), stops scanning and begins inquiry. Any other PH in the HEAL-SCAN state receiving a HEAL-INQ message does the same but does not begin inquiry. Moving our attention to the partitioned sub-trees, if a PH in the HEAL-INQ state becomes orphan, it broadcasts a HEAL-INQ message, remains in the HEAL-INQ state ($TRAN_7$) and starts inquiry. Any other PH receiving a HEAL-INQ message while in the HEAL-INQ state remains in the HEAL-INQ state. Finally, note that a PH losing any of its children while in the HEAL-INQ state is not an event that triggers any action. The scenario described above is illustrated in Figure 3.

An important case happens when the piconet leaving contains the ROOT. During the scatternet formation phase the ROOT selects one of its children PHs as its *successor* in case it leaves abruptly. The ROOT makes sure to keep a successor named at all times, as any of its children may leave. When the ROOT leaves, its successor takes over as ROOT and acts as if it were a parent which lost its child as described above. To deal with the unlikely case that both the ROOT and its successor leave at exactly the same instant, one could extend this idea to an ordered successor list communicated to all the children of the ROOT.

So far we have considered partitions caused by entire user-piconets leaving, not individual nodes. If a node leaving is a slave there is no need for any healing. If the master (PH) of a piconet leaves, though, the piconet will be destroyed. To make its destruction temporary, we introduce a simple mechanism for *piconet healing* based on the idea of a piconet successor which takes over as master when the old master goes. Since the piconet is temporarily dissolved, the rest of the scatternet heals as described in the previous cases. Once the piconet heals, the new

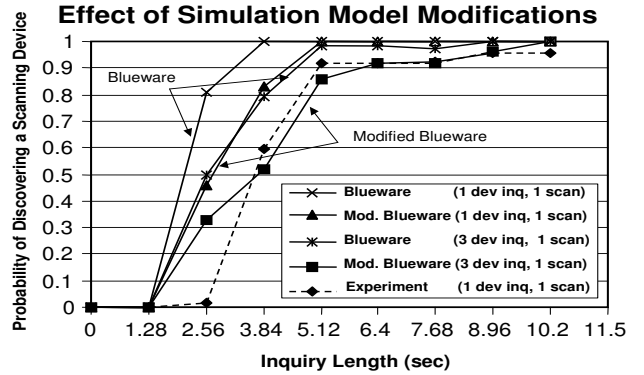


Fig. 4. Effect of modifying the Blueware baseband ns-simulator.

master PH acts as a PH orphan as described above and the piconet attaches to the rest of the scatternet.

IV. PERFORMANCE RESULTS

A. Experimental and Simulation Setup

We implemented our scatternet formation protocol using BTH v1.1 compliant hardware. We used devices running Linux kernel 2.4.18 with the BlueZ stack v2.2. We equipped nodes with dual-radios, since no off-the-shelf BTH hardware supported master/slave (or slave/slave) scatternet operation. We used hold-mode to emulate the effect of having only a single-radio.

For our simulation results we modified the Blueware ns-simulator [8], which was in turn based on BlueHoc [9]. Our changes were based on our experience with real BTH hardware and can be summarized as follows: we implemented a periodic page scan mode with period of 1.28 sec and window 11.25 msec, as opposed to nodes entering the page scan state only after responding to an inquiry; we increased the page timeout value from 1.28 sec to 6.4 sec; we chose randomly between Train A and Train B at the starting point of inquiries; we allowed for fewer than seven BTH links per node to reflect hardware limitations²; we randomized the starting times of nodes inquiring to avoid artificial synchronizations; we increased the number of inquiry responses parameter from 1 to 10 and allowed for fixed-time inquiries in multiples of 1.28 sec. We believe that these changes make our simulation model depict better realistic BTH hardware behavior, as also illustrated in Figure 4. Our modified Blueware simulator can be found in [10].

B. Results and Comments

We first evaluated the delay in scatternet formation. The results when all nodes arrive simultaneously (en-mass) are depicted in Figure 5. For direct comparison with [5] we show the median delays and we evaluate

²Unless otherwise stated, the number of BTH links per node in GC-tree results is set to four to reflect the used hardware.

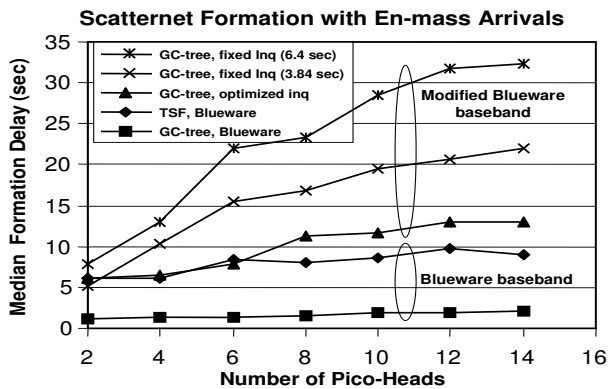


Fig. 5. Formation delay for en-mass arrival.

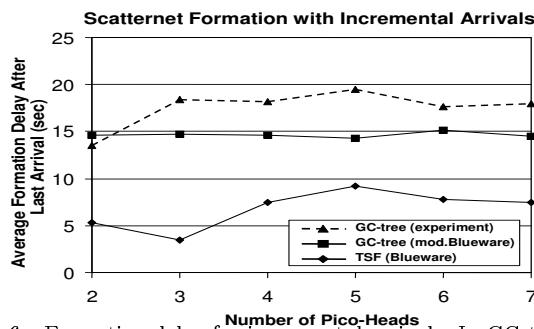


Fig. 6. Formation delay for incremental arrivals. In GC-tree simulation and experiments inquiry length is 12.8 sec.

the GC-tree performance using the Blueware ns-model as well. The reason GC-tree performs so better is because in our approach user-actions dictate which nodes will initially perform inquiry and scanning, while in TSF nodes have to go through a period of alternating between inquiry and scanning. From Figure 5 we note that the best GC-tree performance is achieved with an optimized inquiry, which stops with the first response from a scanning device. Furthermore, Figure 6 presents experimental and simulation results when piconets arrive incrementally every 10 sec. The delays were measured as the interval between the last arrival and the time the scatternet got fully formed.

BTH hardware usually limits the number of available

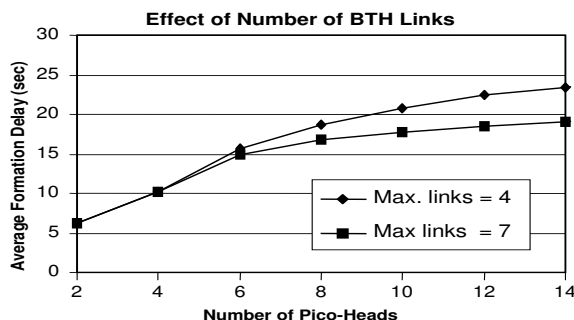


Fig. 7. Effect of maximum number of BTH links/node. PHs with random no. of slaves 0-3 (1-4 user devices). Inq. length is 3.84 sec.

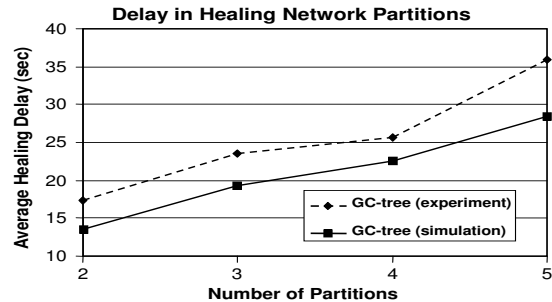


Fig. 8. Effect of no. of partitions in healing. Inq. length 12.8 sec.

BTH links to below the theoretical seven. Figure 7 shows the impact of the max number of links on the performance of GC-tree, obtained with modified-Blueware simulations. Fewer available links translate to fewer possible attachment points in the already formed scatternet and this increases the scatternet formation delay.

Finally, we assessed the performance of the GC-tree healing mechanism with both simulations (modified-Blueware) and experiments. Results shown in Figure 8.

V. CONCLUSIONS

In this paper, we described a new scatternet formation and healing protocol that enables group collaboration. The protocol considers users' piconets as the building blocks of the scatternet; it is dynamic, allowing arrivals and departures of nodes or entire piconets; and creates a tree topology which allows for low routing and piconet switching overhead. Finally, we presented experimental performance results from a prototype implementation and simulation results using a more realistic model.

REFERENCES

- [1] S. Asthana and D. Kalofonos. Enabling secure ad-hoc group collaboration over bluetooth scatternets. In *IEEE ASWN workshop*, 2004.
- [2] G. Zaruba, S. Basagni, and I. Chlamtac. Bluetrees - scatternet formation to enable Bluetooth-based ad hoc networks. In *IEEE Int. Conf. on Comm. (ICC'01)*, 2001.
- [3] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaira. Distributed topology construction of bluetooth personal area networks. In *IEEE INFOCOM*, 2001.
- [4] Hongyuan Chen, T.V.L.N Sivakumar, Leping Huang, and Tsuyoshi Kashima. Topology-controllable scatternet formation method and its implementation. In *International Workshop On Wireless Ad-hoc Networks (IWWAN) '04*, May 2004.
- [5] G. Tan, A. Miu, J. Guttag, and H. Balakrishnan. An efficient scatternet formation algorithm for dynamic environments. In *IATED Comm. and Comp. Networks (CCN'02)*, 2002.
- [6] F. Cuomo, G. Di Bacco, and T. Melodia. SHAPER: a self-healing algorithm producing multi-hop Bluetooth scatternets. In *IEEE Globecom*, 2003.
- [7] C. Petrioli and S. Basagni. Degree-constraint multihop scatternet formation for Bluetooth networks. In *IEEE Globecom*, 2002.
- [8] G. Tan. Blueware: Bluetooth simulation for ns. <http://nms.lcs.mit.edu>.
- [9] IBM Research. Bluetooth extension for ns. <http://oss.software.ibm.com/developerworks/opensource/bluehoc/>.
- [10] *Project Dynamic Proximity Networking, SUNY at Buffalo*. <http://www.cse.buffalo.edu/~qiao/Research/proximity/>.