Journal of Science and Engineering Technology, Vol. 2, No. 3, pp. 1-7 (2006)

Bluetree#: An Extendable Bluetooth Scatternet Formation Using Only Slave/Slave Bridges

CHENG-MIN LIN[1], JEN-CHENG CHIU, TSU-WEN KO and CHYI-REN DOW

[1]*Department of Computer and Communication Engineering, Nan Kai Institute of Technology*

568 Jungjeng Rd., Tsautuen, Nan Tou, Taiwan

Department of Information Engineering and Computer Science, Feng Chia University

100 Wenhwa Rd., Seatwen, Taichung, Taiwan

Email: {lcm, renchen}@ nkc.edu.tw, m9208614@knight.fcu.edu.tw, crdow@fcu.edu.tw

ABSTRACT

With the vigorous development of wireless technology, the life style of human beings has been improved with more convenience and comfort. Bluetooth is a critical technology applied in a shorter distance wireless network. The roles of Bluetooth devices can be divided into three types: master, slave, and bridge. However, the master or bridge nodes become the bottleneck of a data traffic flow. A novel Bluetooth scatternet formation is required to obtain better QoS and lower power consumption. This study presents two novel algorithms. The first is Bluetree++ for increasing scatternet performance. The second is Bluetree# for increasing scatternet. The advantages of these two schemes are as follows. (1) A tree structure can be maintained. (2) No master is also a bridge so the load on the master is reduced. (3) The number of links in a bridge is the minimum possible, thus making the overhead in the bridge lighter.

***Key Words*: Bluetooth, wireless networks, piconet, scatternet, QoS, Bluetree++, Bluetree#**

Bluetree#：僅用從動橋接裝置之可延展的藍芽散射網路結構

林正敏 [1]　邱仁成　柯子文　竇其仁

[1] 南開技術學院電腦與通訊工程系

南投縣草屯鎮中正路 568 號

逢甲大學資訊工程系

台中市西屯區文華路 100 號

摘　要

由於無線網路持續地快速成長，人類的生活也隨之變得更方便及舒適，在短距離傳輸的無線網路世界裡，藍芽是關鍵的技術之一。藍芽裝置的角色可以區分為主動、從動和橋接裝置三種，然而在網路傳送資料時主動和橋接這兩種裝置是流量的瓶頸點，新的藍芽散射網路結構是需要以便獲得有較佳的頻寬品質及較低的電源損耗。此篇文章將提出兩種新的演算法，一是為了提升散射網路傳輸效能所提出的 Bluetree++，另一是提升散射網路的規模所提出的 Bluetree#，這兩種方案的優點如下所述：（1）維持樹的結構，（2）沒有主動橋接裝置，所以

主動裝置的負擔較少，（3）連接橋接裝置的總數量為最小，所以可使橋接節點造成的負載較輕。

關鍵詞：藍芽，無線網路，微網路，散射網路，頻寬品質，Bluetree++，Bluetree#

I. INTRODUCTION

Bluetooth is named from Harald Bluetooth, a Viking and former King of Denmark who was renowned for his ability to help people communicate with others. Most notably he united Denmark and Norway. Today Bluetooth is a technology that unites different types of electronic equipment from various manufacturers, enabling them to communicate with each other without the need for wires. Low-power consumption, low-tier and low-cost largely account for the success of Bluetooth applications [2]. Bluetooth wireless technology is used to communicate data and voice between any two nodes located beyond the effective communicating distance of a piconet or scatternet. Multiple piconets can co-exist in a common area because each piconet uses a different hopping sequence. Piconets can also be interconnected via bridge nodes to form a scatternet. Bridge nodes can timeshare between multiple piconets, receiving data from one piconet and forwarding it to another. Figure 1 illustrates that a bridge, B1 can act as a slave in both piconets (known as a S/S bridge) or a bridge, B2 can act as a master in one piconet and act as a slave in another (known as a M/S bridge). Too many bridges in the scatternet will waste a guard slot and increase the overhead associated with bridge switching among the participated piconets, increasing the probability of packet loss. Too many piconets in a communicative range will degrade the scatternet performance. Furthermore, unnecessary piconets also lengthen the routing path, delaying the transmission of packets from source to destination [3].

Few investigations have attempted to solve these issues in scatternet formation, including Bluetree [10], Bluenet [8-9] and BlueRing [5], but scatternet formation is still an important issue. In 2001, Zaruba *et al.* introduced "*Bluetrees*" as a protocol for forming connected scatternets [10]. Although Bluetree has a tree format structure, all master nodes except for the root nodes act as bridges. Hence, the master nodes have heavy traffic loads. To solve this problem, Wang *et al.* [9] in 2002 proposed a new scatternet formation algorithm, "Bluenet." Although the performance of Bluenet may be better than Bluetree, Bluenet loses the tree structure. Each node in the graph may have multiple predecessors as well as multiple successors. This characteristic may cause a lower performance and routing loop existence. Hence, Bluenet appears out of order and many master nodes also act as bridges. Lin and Tseng [5] proposed BlueRing in 2003. No master node is a bridge connecting two piconets; thus making it more efficient. Although BlueRing provides an effective scheduling scheme for efficient scatternet operation, the number devices is limited to less than 37 [6]. Additionally, two devices located in adjacent piconets spend more time transferring data to each other because of routing in the same direction (clockwise). Summarizing, Bluetree provides a structural tree, Bluenet enhances the performance of Bluetree, and no master node acts as a bridge in BlueRing. However, these schemes have some disadvantages such as, Bluetree performance is lower; Bluenet loses tree formation; BlueRing uses fewer devices. Therefore, this study proposes two novel schemes, Bluetree++ and Bluetree# to maintain the tree structure, retain good performance and ensure that no master node can act as a bridge.

The rest of this paper is organized as follows. Section II presents the Bluetree# family. Section III describes an efficient method, Bluetree#. Section IV presents numerical results and a performance comparison with other schemes. Section V states the conclusions.

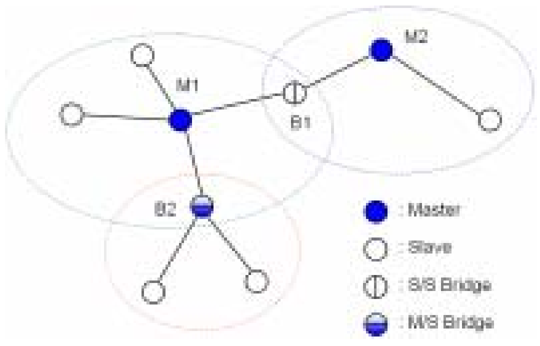

Fig. 1. Bluetooth scatternet

II. BLUETREE# FAMILY

Two Bluetooth units cannot exchange information freely unless a master-slave relationship has been set up between them. To avoid excessive delays in forming connections between potential communication pairs, a scatternet network must be formed in advance by collecting and storing the necessary routing information [9]. This section introduces a family of Bluetree#, three scatternet formation methods, including Bluetree, Bluetree++, and Bluetree#. An exception example using BlueRing is also presented because it is similar to Bluetree++ when a link between a bridge and a master is broken.

1. Bluetree

In [10], Zaruba *et al.* present "*Bluetrees*" as a feasible protocol for forming connected scatternets, which has two variations, namely, *Blueroot Grown Bluetree* and *Distributed Bluetree*. The first protocol starts by designating a node as the "root" of the tree to be formed. This node, called the "blueroot," then acquires its direct, one hop neighbors as slaves. Each slave then pages its unconnected one-hop neighbors and attempts to acquire them. This acquisition process continues with each subsequent level of the tree expanding by connecting to its immediate, unconnected neighbors until the entire tree has been built. The tree is then optimized by limiting each master to a maximum of seven slaves. Any master with more than seven slaves selects one or more pairs of slaves and instructs one node in each pair to become a master of the other. The new master is then disconnected from the piconet to form its own piconet. The second Bluetrees protocol speeds up scatternet formation by selecting several initial roots for the tree formation. It then merges the trees generated by each of the roots.

Both protocols assume that each node knows whether it is in the blueroot, knows the identifiers of its one-hop neighbors, and knows whether those neighbors are already connected. They also assume that two nodes are connected if they are within a given physical distance of each other. In both protocols a master may be assigned more than seven slaves. This situation results in creating extra overhead for parking and reactivating slaves. The authors note that in an open, interference and obstacle-free environment, if a node has more than five neighbors, there are at least two nodes among the neighbors that are neighbors themselves. This observation is used to reconfigure the tree so that no master has more than five slaves. The tree branch reconfiguration is carried out throughout the network. However, Zaruba *et al*. [10] did not prove that this process will actually terminate. The authors did not detail how nodes discover each other and how they establish links using the existing Bluetooth protocols. Basagni *et al.* [1] proposed modifications to the Bluetree protocol that included adding "weights" to each node to help select the masters.

A tree topology may also result in inefficient routing because a message must travel up the tree and then back down to reach its destination. Sun *et al.* [7] suggested that the assumption that each node knows all of its neighbors is unrealistic considering the hopping frequency in Bluetooth devices. Based on the Bluetree rules, we can analyze the nodes in each layer and acquire simple formulas to calculate the master and bridge nodes as well as the total number of nodes in a full Bluetree. To compare the nodes with other protocols, s is defined as slave nodes at most in a piconet and presented as Bluetree(s). There is only one node located on the top of the tree and it grows as a binary tree when s is three. We can calculate the total number of nodes in each role in a full Bluetree using the following formulas:

$$M = \sum_{n=0}^{L-2} s^n \tag{1}$$

$$B = M - 1 \tag{2}$$

$$S = s^{L-1} \tag{3}$$

$$T = \sum_{n=0}^{L-1} s^n \tag{4}$$

where L is longer than one, M presents the number of master nodes as Eq.1, B shows the number of bridge nodes that should satisfy Eq.2, S indicates the total number of slave nodes, and T presents the total number of nodes as Eq.4 in a Bluetree.

2. Bluetree++

The specification clearly defines one Bluetooth master can take at most seven slaves to form a piconet. However, to increase the scatternet communication efficiency, BlueTree++(s, b) scheme [4], where s denotes as the number of slave nodes per piconet and b represents the number of bridge nodes per piconet, is implemented according to the following definition.

1. The first node of the tree is designated as the root.
2. Each parent node, located in odd layers, has s–1 child nodes or s slaves, where $1 \le s \le 7$. Meanwhile, the parent node has at most b–1 subtrees or b bridges, where $0 \le b \le s$.
3. Each nodes located in even layers has at most one child node.
4. The nodes that are not leaves and located in the odd layers are all masters. However, the other nodes that are not leaves and located in even layers are all bridges.

Note that each master node has at most 6 child nodes but not 7 because a predecessor, a bridge node of the master, is included in the piconet. Each master has (s–1) slave and (b–1) bridge linkages located in succeeding layer. The rooted master will have only (s–1) slaves and (b–1) bridge nodes. Figure 2 shows a general tree of Bluetree++ based on (s, b) = (5, 4). There are five slaves at most in each piconet. We observed that each piconet of Bluetree++ (s, b) consists of one master, at most s slaves, and at most b bridges. Because we limited the bridge play to only the slave/slave role in our algorithm, all master nodes never act as bridges. Each bridge is limited to two links as illustrated in Figures 2 to 4.

Figures 3 and 4 illustrate two Bluetree++ (4, 2) and Bluetree++ (4, 3) samples, respectively. There are only master nodes in the odd layers. Each master could have successors forming their own subtrees. Each master node, except for the root, has only one predecessor. Notice that this predecessor must be a bridge. Figure 3 shows that each master can have four slaves. Two of the four slaves in the topology will act as bridges to connect with another piconet. This scheme is called Bluetree++ (4, 2). As a result there are two bridges and two slaves in each piconet. Figure 4 reveals that each master node takes four slaves. We observed that there are four slaves at most in each piconet. At most three of these slaves will play a bridge role to expand the scatternet. Hence, this topology is called Bluetree++ (4, 3).

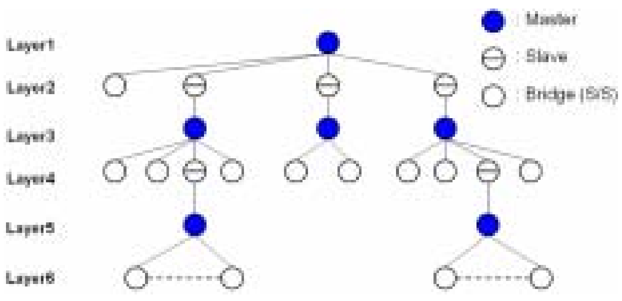

Fig. 2. General tree of Bluetree++ (5, 4)

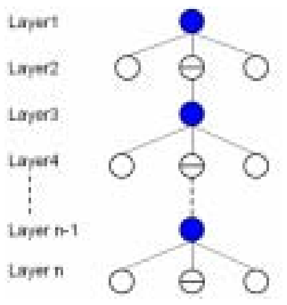

Fig. 3. A tree of Bluetree++ (4, 2)

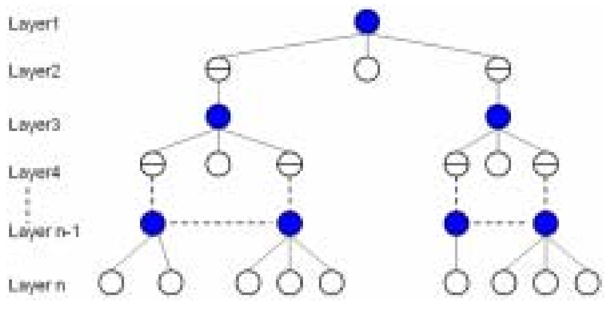

Fig. 4. A tree of Bluetree++ (4, 3)

3. Bluetree#

Each bridge node in the Bluetree++ scheme has only two links with two master nodes to form a connection between the two different piconets. This phenomenon could make a higher tree, and lengthen the communicating path, especially for two leaf nodes located at different sides from root node of the tree. The performance will decrease because of the longer path. To overcome the drawback of Bluetree++, we propose a novel scheme based on Bluetree++ named Bluetree#. Bluetree# will have more potential to shorten the layers and communicating path. A Bluetree# (s, b, l) tree, where l is the number of links in a bridge node, is formed according to the following rules.

1. The first node of the tree is designated the root.
2. Each node in the odd layers has s slaves including a parent node and s–1 child nodes ($1 \le s \le 7$). The parent node has at most (b–1) subtrees ($0 \le b \le s$). Notably each master node has at most 6 child nodes but not 7 because a parent node of bridge is included in the master node in a piconet.
3. Each nodes located in the even layers has at most one child node.
4. A child node that functions as a bridge role has (l–1) subtrees at most, where $l \ge 2$. Basically the l makes no difference with s and b.
5. The non-leaf nodes are located in odd layers are all masters. However, the other non-leaf nodes are located in the even layers and are all bridges.

For the same reasons as Bluetree++, the rooted master will only have at most (s–1) slaves and (b–1) bridge nodes.

Figure 5 shows that each master can have three children. Two of the three child nodes in the subtree will act as bridges as a predecessor of the next bottom subtree. Each bridge can have at most l–1 links to the next bottom subtree. This scheme is called Bluetree# (4, 3, 3). Obviously, Bluetree# will shorten the communicating path and layers.

4. BlueRing

A previous study [5] presented another scatternet formation scheme, "BlueRing." BlueRing is quite similar to Bluetree++ in application. BlueRing could be regarded as a

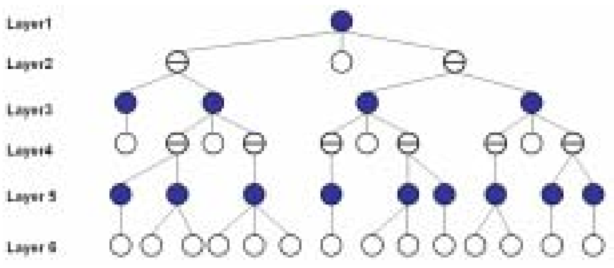

Fig. 5. An example of Bluetree# (4, 3, 3)

special case of Bluetree++ (*s*, 2). The following rules are applied to simplify BlueRing.

1. The rule of Bluetree++ (*s*, 2) must be followed, where *s* is larger than one.
2. The button layer must be an even layer, and its child node must be the only node located in the top layer.

III. BLUETREE# ANALYSIS

We define a full Bluetree# (*s, b, l*) as satisfying the four conditions listed as follows:

1. All masters have *s* salves and *b* bridges from the slaves.
2. Each bridge node is connected to *l* links.
3. A tree grows top down, layer by layer.
4. Unless the limit of Bluetooth is specified, 256 piconets exist at most in a scatternet and the tree must be symmetrical.

Let *L* be the layers in a full Bluetree# (*s*, *b*, *l*). If *L* is equal to one or two, the result is as same as Bluetree++. We analyze the other conditions and divide them into two parts. Equations 5 to 7 and 8 to 10 describe each condition allocated in the odd and even layers to evaluate the extending scale of each role for nodes in a full Bluetree #.

When *L* is odd, each role of nodes will be the equations as follows.

$$M = \begin{cases} 1 & L = 1 \\ \sum_{i=0}^{\frac{L-3}{2}} (b-1)^i \times (l-1)^i + (b-1)^{\frac{L-1}{2}} \times (l-1)^{\frac{L-1}{2}} & L > 2 \end{cases}$$

and *L* is odd, (5)

$$B = \begin{cases} 0 & L = 1 \\ \sum_{i=1}^{\frac{L-1}{2}} (b-1)^i \times l^{i-1} & L > 2 \end{cases} \quad \text{and } L \text{ is odd,} \qquad (6)$$

$$S = \begin{cases} 0 & L = 1 \\ \sum_{i=0}^{\frac{L-3}{2}} (b-1)^i \times (l-1)^i \times (s-1) - B \times 2 & L > 2 \end{cases}$$

and *L* is odd. (7)

As *L* is even, each role of nodes will be the equations as follows

$$M = \begin{cases} 1 & L = 2 \\ \sum_{i=0}^{\frac{L-2}{2}} (b-1)^i \times (l-1)^i & L > 3 \end{cases} \quad \text{and } L \text{ is even,} \qquad (8)$$

$$B = \begin{cases} 0 & L = 2 \\ \sum_{i=1}^{\frac{L-2}{2}} (b-1)^i \times (l-1)^{i-1} & L > 3 \end{cases} \quad \text{and } L \text{ is even,} \qquad (9)$$

$$S = \begin{cases} s & L = 2 \\ M \times (s-1) - B \times 3 & L > 3 \end{cases} \quad \text{and } L \text{ is even.} \qquad (10)$$

where

s: Max number of slaves in a piconet,
b: Max number of bridges in a piconet and $b \le s$,
l: Max linkage number of bridge node,
L: the layers of full Bluetree#,
M: Total number of masters in the full Bluetree#,
B: Total number of S/S bridge nodes in the full Bluetree#,
S: Total number of slave in the full Bluetree#.

Figures 6 and 7 are two examples to illustrate the different policies with *s*, *b*, and *l* in a full Bluetree# protocol. Both have same number of master and bridge nodes. There are twenty-one master and ten bridge nodes in each full Bluetree#. There are fifty-four slave nodes in Bluetree# (4, 3, 3) and seventy-five ones in Bluetree# (5, 3, 3).

We also observed that the Bluetree# rules could apply to Bluetree++. There are three parameters in Bluetree# named *s*, *b*, and *l*. Two of the three parameters, *s* and *b*, are applied to Bluetree++. In our observation, there are only two links to both a successor and a predecessor, respectively, for each bridge node in the Bluetree++ protocol. Therefore, Bluetree# (*s*, *b*, 2) is equal to Bluetree++ (*s*, *b*).

Theorem 1. Bluetree++ (*s*, *b*) is the same as Bluetree# (*s*, *b*, *l*) when *l* is two.

Proven. As defined in the previous section, *l* is the number of links with a bridge node that acts only in the slave/slave role.

When *l* is 2, one of the two links will act with a predecessor master node. The other one will link with a child node in its subtree. If the two links are linked from its predecessor master nodes, and no link is left to connect with its

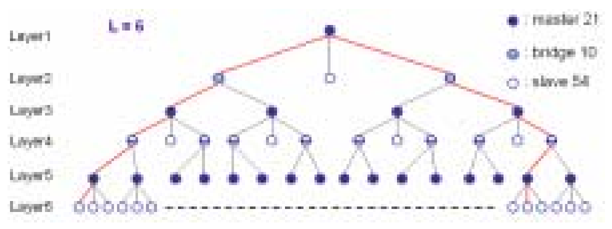

Fig. 6. A Bluetree# (4, 3, 3) with six layers full tree

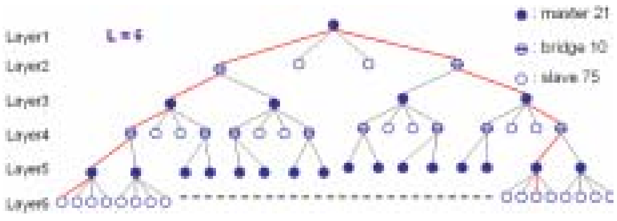

Fig. 7. A Bluetree# (5, 3, 3) with six layers full tree

subtree, the tree structure will break off. Because the Bluetree++ topology is also a tree structure, each bridge node in Bluetree++ has a link from its predecessor master node connected with a master node from its subtree. These two conditions in Bluetree++ (*s*, *b*) and Bluetree# (*s*, *b*, *l*) are equal and maintain the same tree structure.

When *l* is 3, besides the two basic links that come from predecessor and link with its subtree, the left link should make connection with a master from another piconet. This structure does not match a basic definition of Bluetree++ when *l* is greater than or equal three. Hence, we can prove that Bluetree++ (*s*, *b*) is the same as Bluetree# (*s*, *b*, 2). ■

IV. EXPERIMENTAL RESULTS

Experimental results based on Bluetree# algorithm will be demonstrated in this section. There are three parameters, named *s*, *b*, and *l*. The *s* is defined as the maximum number of slave nodes in a piconet. The *b* indicates the maximum number of bridge nodes in a piconet and the *l* shows the maximum number of links for a bridge node. They are used to represent a Bluetree# family tree formation. We will discuss and compare the experimental results in each formation.

Figure 8 shows the bridge ratio for three formations when *s* is 5, *b* is 4 and *l* is 3. Here, the bridge ratio is defined as the number of bridge nodes divided by the total number of nodes. Bluetree# has lowest bridge ratio in the same layer for three formations. The ratio is reduced by 15% more than Bluetree and 10% more than Bluetree++. The bridge nodes of Bluetree++ and Bluetree# only use the slave/slave configuration with lower traffic than the master/slave configuration in Bluetree. Therefore, Bluetree++ enhances the tree based Bluetooth scatternet performance becase of the number of bridge links for Bluetree++ is only two when that for Bluetree# is *l* and that for Bluenet is *s*+1.

Furthermore, Figure 9 shows the slave ratio of three formations when *s* is 5, *b* is 4 and *l* is 3. It indicates the

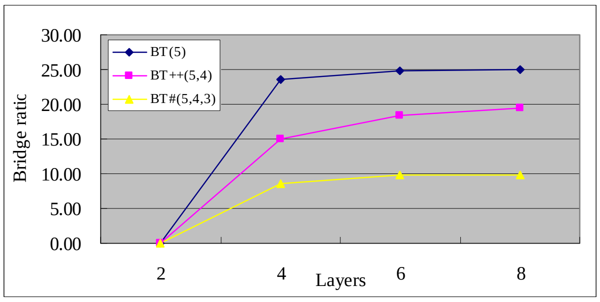

Fig. 8. Bridge ratio of three formations

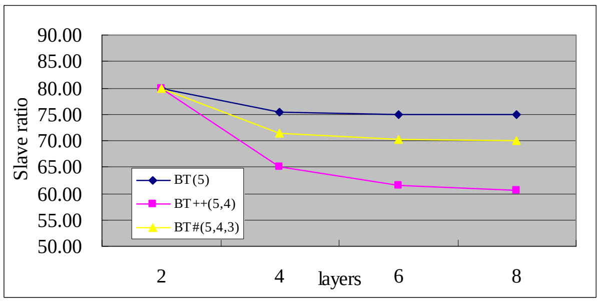

Fig. 9. Slave ratio of three formations

scalability of the tree formations. First of all, we define a slave ratio as the number of slave nodes divided by the total number of nodes. Although, Bluetree has a higher slave ratio than other formation, the performance is lower than Bluetree#. Bluetree# has higher scalability than Bluetree++.

V. CONCLUSIONS

A well structured scatternet with the appropriate number of piconets and bridges for a specific traffic pattern will increase the performance of a Bluetooth network. This paper proposed two efficient methods to form a scatternet for typical personal wireless communication applications. The two algorithms were applied in a distributed formation for ad hoc networking using Bluetooth technology. Because a smaller piconet is created in which fewer bridge nodes are needed, the proposed schemes, Bluetree++ and Bluetree# more easily conform to the piconet link QoS requirement. The advantages of these schemes are as follows. (1) A tree structure can be maintained. (2) No master is also a bridge, so the master load is reduced. (3) The number of links in a bridge is the minimum possible, thus making the bridge overhead lighter.

Because the tree-based topology will cause a bottleneck at master nodes, basing the system on a tree-based topology by adding simple rules to improve the load problem is an interesting issue. The mobility and join or lost connections for each node will be investigated in the future.

ACKNOWLEDGEMENT

The authors would like to thank the National Science Council of R.O.C. for financially supporting this research under Contract NSC-93-2213-E-252-005.

REFERENCES

1. Basagni, S., R. Bruno and C. Petrioli (2003) A performance comparison of scatternet formation protocols for networks of Bluetooth devices. Proceedings of the First IEEE International Conference on Pervasive Computing

and Communications, Dallas, Texas.

2. Bluetooth Special Interest Group (2003) Specification of the Bluetooth System. version 1.2, http://www.bluetooth.com.
3. Chang, C. Y., K. P. Shih, S. C. Lee and C. H. Tseng (2004) Adaptive role switching protocols for improving scatternet performance in Bluetooth radio networks. Proceedings of 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Barcelona, Spain.
4. Lin, C. M., C. R. Dow, T. W. Ko, C. M. Chiu and K. L. Liao (2005) BlueTree++: A novel scatternet formation scheme. Proceedings of the 23rd IASTED International Conference on Parallel and Distributed Computing and Networks, Innsbruck, Austria.
5. Lin, T. Y. and Y. C. Tseng (2003) A new BlueRing scatternet topology for Bluetooth with its formation, routing, and maintenance protocol. *Wireless Communications and Mobile Computing*, 3(4), 517-537.
6. Salonidis, T., P. Bhagwat, L. Tassiulas and R. LaMaire (2001) Distributed topology construction of Bluetooth personal area networks. Proceedings of IEEE INFOCOM, Anchorage, Alaska.
7. Sun, M. T., C. K. Chang and T. H. Lai (2002) A self-routing topology for Bluetooth scatternets. Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks, Manila, Philippines.
8. Wang, Z., Z. Haas and R. J. Thomas (2003) Bluenet II -- A detailed realization of algorithm and performance analysis. Proceedings of the 36th Hawaii International Conference on System Science (HICSS-36), Big Island, Hawaii.
9. Wang, Z., R. J. Thomas and Z. Haas (2002) Bluenet -- A new scatternet formation scheme. Proceedings of the 35th Hawaii International Conference on System Science (HICSS-35), Big Island, Hawaii.
10. Zaruba, G. V., S. Basagni and I. Chlamtac (2001) Bluetrees - Scatternet Formation to Enable Bluetooth-Based Ad Hoc networks. Proceedings of IEEE International Conference on Communications (ICC 2001), Helsinki, Finland.

Received: Jan. 24, 2006 Revised: May 9, 2006
Accepted: Jul. 30, 2006