Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 66 (2006) 1243–1258
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

BlueCube: Constructing a hypercube parallel computing and communication environment over Bluetooth radio systems

Chao-Tsun Chang[a], Chih-Yung Chang[b,*], Jang-Ping Sheu[a]

[a] *Department of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan*
[b] *Department of Computer Science and Information Engineering, Tamkang University, Tamsui, Taipei, Taiwan*

Received 17 November 2003; received in revised form 9 September 2005; accepted 17 April 2006
Available online 25 July 2006

Abstract

In parallel computing structures, Hypercubes [P. J. Wan, L. W. Liu, Y. Yang, Optimal routing based on the super-topology in Hypercube WDM networks, 1999, pp. 142–149] and [Y. R. Leu, S. Y. Kuo, A fault-tolerant tree communication scheme for hypercube systems, IEEE Trans. Comput. 45(6) (1996) 643–650] have many advantages: they support parallel computing, provide disjoint paths, and tolerate faults. If devices with computing capabilities can be linked as a Hypercube by taking advantage of Bluetooth radio's features, then an efficient communication and high-performance computing environment can be established by applying currently used algorithms. A Bluetooth device randomly searches for and connects with other devices, using time-consuming inquiry/inquiry scan and page/page scan operation and hence, results in an uncontrolled scatternet topology and inefficient communications. The present work proposes a three-stage distributed construction protocol for rapidly organizing a Hypercube computing environment that was constructed from Bluetooth devices. The proposed protocol governs the construction of links, the assigning of roles and the formation of the scatternet in order to efficiently construct a Hypercube structure. The constructed scatternet easily enables Bluetooth devices to establish a routing path, tolerate faults and create disjoint paths, and thus, achieves parallel and distributed computing in a Bluetooth wireless environment. Experimental results reveal that the proposed protocol can set up a scatternet that is appropriate for parallel computing and communications.

Keywords: Bluetooth; Hypercube; Role switch; Piconet; Scatternet; Parallel computing; Wireless communication

1. Introduction

The simultaneous use of the computing power of many machines to share several computations has been extensively investigated in recent years. Grid computing [4,20,3] has received much attention as an effective means of collecting possible computing resources over a wired network. Bluetooth [21] is a new wireless communication technology that has been gradually embedded in various machines such as desktop PCs, notebooks, Tablet PCs, workstations and others. Bluetooth connection technology dynamically links various machines more flexibly and easily than what could be achieved before, and also provides channels of communication between various types of machines. Thus, the use of Bluetooth technology to link various machines exploits the potential of wireless networks to be used in parallel and grid computing. Parallel architecture and programs designed for specific architectures have been widely discussed during the last decade. One of them is the Hypercube, a promising topology for which existing parallel algorithms have been designed with proven effectiveness. The present work proposes a connection algorithm which constructs a Hypercube topology for high-performance computing and communication via Bluetooth radio links.

In a Bluetooth network, a *piconet* is comprised of one master device and at most, seven active slave devices. A *scatternet* is a wireless network consisting of several piconets and bridge devices. The bridge services more than one piconet and is responsible for cross-piconet communication and service. A bridge can have different roles in different piconets. A slave/slave (S/S) bridge plays the slave roles in all participating piconets, whereas

* Corresponding author. Fax: +886 2 26209749.
E-mail addresses: cctas@tcts.seed.net.tw (C.-T. Chang), cychang@cs.tku.edu.tw, cychang@mail.tku.edu.tw (C.-Y. Chang), sheujp@csie.ncu.edu.tw (J.-P. Sheu).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.04.018

a master/slave (M/S) bridge plays the master role in one piconet and the slave roles in other participating piconets. However, a single device can never be a master for more than one piconet. A Bluetooth device randomly searches and connects with other devices using the time-consuming inquiry/inquiry scan and page/page scan operations that eventually results to a disconnected or uncontrolled scatternet.

Many studies [6,7,12,13,15,18] have investigated the Bluetooth connection problem to establish a connected scatternet. Connected scatternets have been formed [17] using Bernoulli trials to determine the role of each device in the Bluetooth discovery process. This method not only fulfills the scatternet connection requirement, but also minimizes the number of piconets in the scatternet. Protocols [19,24] have been proposed to generate connected scatternets and enable each master to manage its neighboring bridges in order to improve communication efficiency. In a connected scatternet, a cross-piconet routing path should be established when two Bluetooth devices in different piconets intend to communicate with each other. The protocols developed in [1] construct a routing path between two devices in different piconets. An efficient route reconstructing protocol [2] which applies role switching operations and a relay reduction policy, has been proposed to minimize the average single-path length. However, the route established by these protocols relies heavily on the existing topology of the scatternet. An improper scatternet topology typically leads to inefficient routes associated with problems of long routing paths and hence, increases power consumption and the rate of packet loss. The improper topology also creates unreliable communication environments in which the parallel computing process must tolerate faults.

Many studies [19,24] have sought to establish a connected scatternet using a preset scatternet topology in order to reduce the route length. However, the construction of the scatternet does not take into account the disjoint path or fault-tolerance [22,8,14]. Bluenet [23] has been proposed for constructing a connected scatternet based on the local knowledge of the node's neighbors. It affords multiple communication paths between source and destination devices rather than the unique path on the loop-free hierarchical structures of Bluetree [7] or Bluestars [16], although fault-tolerance is not addressed. BlueRing [9] is used to construct a ring structure with M/S bridges and protocol-level remedy, while recovery mechanisms are applied to reduce the packet delay and improve the network throughput. Moreover, single-point or multi-point failure is tolerated. Although routing on BlueRing scatternet is stateless, a long routing path and congestion result to inefficient communication. BlueMesh [10] has been presented to establish a connected mesh scatternet with a small network diameter, with disjoint paths between any pair of devices as well as with easier routing and scheduling. However, [6] noted that M/S bridges reduce the piconet performance since cross-piconet communication must be suspended when the master is participating in another piconet and hence, results in inefficient communication. Without a competent structure and role assignments, the controlled topology of a scatternet may not suffice to support parallel computing and optimize network performance.

If a well-known parallel computing environment like the Hypercube can be automatically established in a distributed manner, then the constructed scatternet will have the following characteristics.

(1) Many of the well-known algorithms have been developed for solving a set of popular problems in Hypercube. The developed algorithms can be efficiently utilized for parallel computing.
(2) Single and multiple disjoint routes can be established more easily in Hypercube than in any other arrangement. Hence, an efficient communication can be attained.
(3) Fault-tolerance has been widely addressed in relation to Hypercube. Thus, when the power is switched off, or a path is broken by a fault in a device, available results on fault-tolerance can be applied to solve communication and computing problems.
(4) Since disjoint routes are associated with every pair of devices, data can be transmitted on other routes even if the current route is a failure. This fact supports the construction of a backup route or the acceleration of data transmission using disjoint routes in communication.
(5) A Hypercube has no problem of disconnected scatternet or redundant bridges which can increase the propagation delay and consumption of power energy.
(6) Distributed computing can be performed in a wireless environment. Orderly transmission of data reduces the probability of packet collisions and transmission delays.

A Hypercube can be easily constructed conceptually: two nodes are linked together if their IDs differ by exactly one bit position. However, the core difficulties in constructing a Hypercube are those of rapidly assigning a unique binary code to each distributed device and bridging piconets with S/S devices in the constructed scatternet while randomly searching and connecting devices. The present work proposes a three-stage distributed protocol for constructing a Hypercube environment. By applying encoding techniques and role switching operations to the formation of a Bluetooth scatternet, the proposed protocol constructs the BlueCube which encompasses parallel computing and communicative environment features of the Hypercube. The resulting BlueCube meets all the above mentioned characteristics.

The rest of the present work is organized as follows. Section 2 introduces the background of Bluetooth as well as the challenge and basic idea behind the construction of a BlueCube scatternet. Section 3 describes the three-stage protocol for constructing a BlueCube, while Section 4 considers the performance of the proposed protocol. Section 5 draws the conclusions of the present work.

2. Background and basic concepts

This section introduces the process for establishing linkages. Some of the challenges for constructing a Hypercube scatternet using Bluetooth devices as well as the basic concepts related to the BlueCube construction protocol for overcoming these challenges are presented. The role switching operations used to

improve the performance of the BlueCube are also introduced. Furthermore, this work has the following assumptions regarding the scatternet environment. In the first place, all Bluetooth devices are within a range that enables them to be connected. Restated, any two devices can receive signals emitted by the other device. Secondly, all Bluetooth devices know how many devices are present in the environment. Similar assumptions were also made in [17,5]. Lastly, no link initially exists between any two Bluetooth devices.

2.1. Establishing links

The Bluetooth radio frequency (RF) module hops over 79 channels at 2.4 GHz in the unlicensed ISM band at 1600 times per second to prevent cochannel interference. Both the short packets and fast hopping capability increase the reliability of communication between two Bluetooth devices. Time division duplex (TDD) technology is used to achieve full duplex for a link connection [13]. The formation of a Bluetooth piconet primarily involves two important procedures namely, the inquiry/inquiry scan and the page/page scan. In the following illustration for establishing links, device A is the master while device B is the slave.

During the inquiry/inquiry scan, device A stays in the inquiry state to collect information about other devices to which it will be connected. On the other hand, the slave device B in its inquiry scan state, searches for the Bluetooth device in the inquiry state to which it will link. Initially, device A transmits an ID packet twice per time slot to quickly find a device in the inquiry scan state. On the other hand, device B, while still in the inquiry scan state, receives device A's ID packet. When the slave device receives the first ID packet, it must wait for a random backoff from 0 to 1023 time slots to prevent collision. Otherwise, it would cause a collision when they transmit back the FHS packet. After the backoff time, device B wakes up and stays in the inquiry scan state in order to receive the second ID packet transmitted by device A. Upon receiving the packet, device B enters the inquiry response state and returns to device A the FHS packet which contains its own clock, *BD_ADDR* and other relevant information. This information will assist device A in its page state in order to predict device B's hopping sequence and channel location in its page scan state. Device B enters the page scan state after transmitting back the FHS packet, while device A enters the page state upon receiving the FHS packet.

During the page/page scan procedure, the slave device, upon entering the page scan state, obtains the clock and *BD_ADDR* information about the master device that enters the page state. This enables the slave device to determine the hopping sequence of the master after linkage. Once device A enters the page state, it initially applies the slave's clock and *BD_ADDR* to predict device B's channel location, and thereafter sends an ID packet to device B to confirm that device B is indeed in the predicted channel. If device B is already in the page scan state and receives the ID packet transmitted by device A, then device B will retransmit the ID packet back for confirmation.

After device A receives the ID packet from device B, it in turn sends an FHS packet to device B. This packet contains information about device A's clock and *BD_ADDR*, as well as a 3-bit active member address which tells the hopping sequence of device A to device B after linkage. This 3-bit active member address (*AM_ADDR*) is used to identify the slave device which intends to communicate with the master in a piconet after linkage. On receiving the FHS packet from device A, device B sends an ID packet back to device A to confirm that the FHS packet was received. At this point, device B enters the connection state, while device A enters the connection state upon receiving the ID packet. After entering the connection state, device A initially transmits a POLL packet to device B to confirm the successful connection of the piconet. Upon receiving the POLL packet, device B, which knows that the predicted channel hopping sequence of device A is correct, sends a NULL packet to inform device A and officially joins the piconet established by device A. This completes the connection and then device A or device B begins to transmit data.

As described above, a Bluetooth device stays in the inquiry/inquiry scan state for a long period and utilizes a large proportion of the connection time and power. The random search and connection also increase the difficulty of constructing a regular topology. If the connection is not controlled, then the constructed scatternet may have too many piconets and hence, increases the probability of packet collisions in the hopping sequence as well as the power consumption and the propagation delay. Furthermore, if the assignment of roles is not controlled, then the existence of the M/S bridge will reduce the inter-piconet performance as previously stated. The following subsection presents some of the challenges involved in constructing a Hypercube scatternet based on the aforementioned link construction procedure.

2.2. Challenges in constructing a hypercube scatternet and related basic concepts

This subsection initially addresses the challenges in constructing a Hypercube scatternet and these involve controlling the topology, role-playing and controlling the number of piconets. The basic concepts associated with the BlueCube construction protocol to overcome these challenges are also presented.

2.2.1. Challenges in topology control

Some challenges are encountered when constructing a Hypercube using Bluetooth devices. A Bluetooth device that acts as the master executes the inquiry procedure to connect randomly with another device which executes the inquiry scan procedure. However, a Hypercube has a regular topology which can be constructed by connecting the corresponding devices in two subcubes. In the example shown in Fig. 1(a), the two 1D subcubes are considered in order to construct a 2D Hypercube. A construction protocol without an elaborate control of the link connection yields an unexpected topology. Figs. 1(b) and (c) show examples of the unexpected topology in constructing a 2D Hypercube from two 1D subcubes. In Fig. 1(b), those devices which belong to the same subcube may connect with each other. In Fig. 1(c), two devices of a subcube connect to

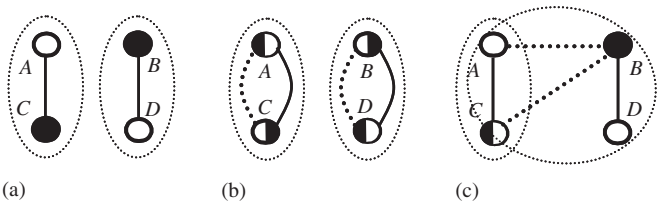

Fig. 1. (a) An example scatternet includes two 1D subcubes. (b) An unexpected topology. Devices that belong to the same subcube may connect with each other. (c) An unexpected topology. Two devices of a subcube connect to the same device of another subcube and hence, result in an unexpected structure.

a single device of another subcube and hence, result in an unexpected structure. A coding mechanism should be used to design a topology construction protocol that selects exactly one device to be connected to another device in a different subcube and hence, prevents the construction of an unexpected topology. Thus, a ring scatternet which connects all the Bluetooth devices will be obtained. Finally, a Hypercube topology can be constructed by connecting some devices in the ring scatternet.

2.2.2. Challenges in the control of role-playing and the number of piconets

Another challenge in constructing a Hypercube is the assignment of unexpected roles. Even though a Hypercube topology has been constructed, an improper role assignment will not support parallel computing and thus, fail to provide an optimal network performance. For example, an improper role assignment may create a large number of piconets and hence, increase transmission delay, power consumption and difficulty of scheduling. As shown in Fig. 2(a), the constructed Hypercube comprises four piconets and four bridges. Different piconets have different hopping sequences and clock standards. Thus, when device A receives data from device B, it will switch to another piconet, change its hopping sequence, and adjust its clock offset. Afterwards, the packet can be transmitted to device C. Besides, the presence of many piconets also raises the problem of collision in the hopping sequences. If device roles are properly assigned in the construction, the number of piconets and bridges can be simultaneously reduced from four to two, as shown in Fig. 2(b). Since devices A, B and C belong to the same piconet, device A may receive a data packet from device B, and then directly transmit it to device C in the next time slot. An improper role assignment may also lead to the formation of an inefficient bridge with the M/S role and hence, result in difficulties of scheduling and coordination among the devices. In Fig. 2(a), the constructed Hypercube has four M/S bridges. Proper role switching operations can be applied to change the devices' role from M/S to S/S as presented in Fig. 2(b) and thus, improve the performance of scheduling and coordination between devices.

The construction of a Hypercube without an efficient and competent connection control may cause some piconets to collect an excess of slave devices and hence, increase the traffic congestion rate and the packet loss rate. As shown in Fig. 3(a), master A connects three bridges and four slave devices

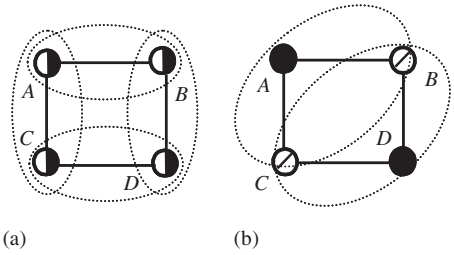

Fig. 2. (a) An example of a Hypercube scatternet constructed without a competent role assignment. (b) An example of a Hypercube scatternet constructed with a competent role assignment. Each bridge plays the S/S role and fewer piconets exist in the scatternet.

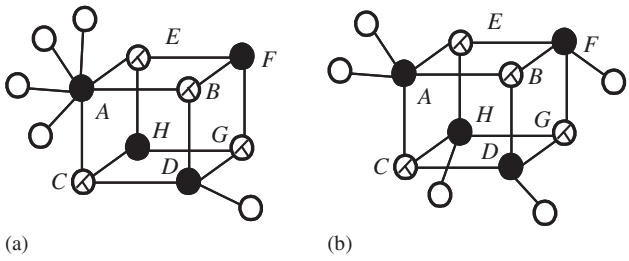

Fig. 3. (a) An example of a Hypercube scatternet constructed with an excess of slave devices collected in piconet A. (b) An example of a Hypercube scatternet constructed with a similar number of slave devices in each piconet.

and thus, becomes a traffic bottleneck in the Hypercube scatternet. Proper connection control should be designed to distribute slave devices so that the numbers of slave devices connected with devices A, D, F and H are approximately equal as shown in Fig. 3(b).

2.2.3. Basic concepts for overcoming the challenges of constructing a hypercube scatternet

Each node should take a unique binary code to serve as an ID for linking the proper devices and overcome the problem of topological control. In the establishment of a link, a proper role assignment results into two devices intending to connect with each other in order to create a link with a predicted role and to form a controlled piconet. In addition, a constructor will be selected from each of the scatternets to connect with another scatternet's constructor under a proper role assignment. Two small scatternets grouped with 2^{k-1} devices are connected accordingly to form a big scatternet with 2^k devices. The two combined scatternets must have the same number of devices prior to the combination. Thus, the encoding operation in the protocol design facilitates the connection and role assignment of the Bluetooth devices.

Once the Hypercube topology is constructed in the connected scatternet, the number of bridges and piconets are reduced and the bridge devices with M/S role are changed. The role switching operations will be applied to reduce the numbers of bridges and piconets and change the role of each bridge from M/S to S/S. Finally, the devices which are not yet attached to the scatternet will be uniformly connected to existing piconets based on the coding information.

A three-phase distributed construction protocol is presented to implement the aforementioned ideas. The proposed protocol performs the encoding operation on *CK* (*connection key*) and *DOC* (*degree of connection*) values to facilitate the connection and role assignment of the Bluetooth devices and thus, construct the BlueCube efficiently. The constructed BlueCube is connected and has the shortest routing and multiple disjoint routes. In the present work, the *CK* value is used to determine whether the device enters the inquiry or the inquiry scan state. The information from *DOC* determines the connection with particular devices. The connection process consists of the following three phases.

Phase I. Ring construction phase (*RCP*): A coding mechanism is adopted to construct a ring scatternet and maximize the dimensions of a Hypercube structure. All devices in a ring scatternet form a Hypercube structure in the later phases.

Phase II. Scatternet construction phase (*SCP*): The *SCP* phase has two purposes. The first is to reduce the number of piconets and bridges, and the second is to connect the devices which have not yet been connected with the master devices in the ring scatternet. The role switching operations, piconet combination and piconet splitting, are applied to reduce the number of piconets and bridges.

Phase III. BlueCube construction phase (*BCP*): This phase restructures the scatternet from the two previous phases. The constructed *BlueCube computing environment* facilitates the building of a routing path, tolerates faults, and provides disjoint paths.

Section 3 presents the details of the three phases. The next subsection introduces the role switching operations which will be used in the proposed protocol.

2.3. Role switching operations

The performance of a Hypercube scatternet depends not only on the constructed topology but also on the role assignment of each device. The link connection procedure defined in the Bluetooth specifications may cause a device to have an improper role or yield too many piconets. Breaking radio links or changing the roles of some devices are simple ways to reorganize the topology, but doing so takes time because it involves the inquiry/inquiry scan and page/page scan operations. The proposed protocol uses the *role switching* operation to change the roles of the devices and improve the structure of the scatternet. The role switching operation [21] enables two devices to exchange roles very quickly instead of establishing a new link by executing the time-consuming inquiry and inquiry scanning procedures. In constructing the BlueCube, the two basic role switching operations, namely, *piconet splitting* and *piconet combination*, are utilized as basic functions for reducing the number of bridges and piconets and for changing the role from M/S bridge to S/S bridge.

2.3.1. Splitting piconet

A role switching operation can split a piconet into two piconets. As shown in Fig. 4(a), in piconet P, the slave device b intends to create a new piconet P_2 and acts as its master. Device b initiates a role switching request to device a. Fig. 4(b) shows that the device b creates a new piconet P_2 and plays a master role to P_2. Then, device a alternately participates in two piconets P_1 and P_2, with master and slave roles, respectively. Such role switching increases the number of piconets and bridges and is referred to as *piconet splitting*.

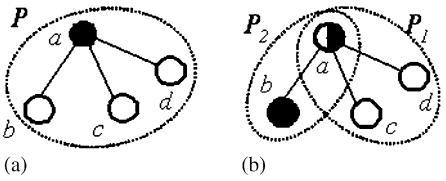

Fig. 4. (a) Topology before executing the *piconet splitting* operation. (b) Topology after executing the *piconet splitting* operation.

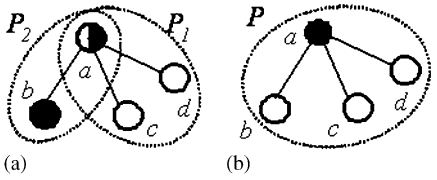

Fig. 5. (a) Topology before executing the *piconet combining* operation. (b) Topology after executing the *piconet combining* operation.

2.3.2. Combining piconets

As shown in Fig. 5(a), devices b and a play the master and slave roles, respectively, in piconet P_2. If device b intends to act as a slave, it sends a role switching request to exchange its role with that of device a. The role switching operation combines two piconets P_1 and P_2 into a single piconet P, as shown in Fig. 5(b). The role switching operation eliminates the bridge role of device a. This type of role switching reduces the number of bridges and piconets.

However, a proper role assignment and application of the role switching operation to the constructed scatternet do not guarantee the formation of a connected scatternet with a regular structure such as that of a Hypercube. Section 3 gives the protocol for constructing a BlueCube scatternet.

3. Protocol for constructing a BlueCube

This section details the BlueCube protocol for constructing a parallel computing and communication environment. Some terms are defined to enable the protocol to be described clearly and in detail.

Definition (*Degree of connection (DOC)*). *DOC* is the degree (or number of dimensions) of a BlueCube established in a scatternet. When two scatternets are linked together, their *DOC* values are compared. They can link up only if their *DOC* values are identical. The initial value of *DOC* is zero. Also, 2^{DOC} is the number of devices in the connected scatternet.

Definition (*01* Sequence*). A 01* sequence is a regular expression of bits. The sequence is represented as a leading 0, followed by repeating 1's. Examples of 01* sequences are 0, 01, 011, 0111 and so on.

Definition (*Connection key (CK)*). *CK* value is *NULL* or a series of digits 0 and 1. The initial value of *CK* is NULL. Every device must maintain a certain *CK* value that is used to determine whether the device should perform an inquiry or an inquiry scan operation.

Definition (*Constructor*). A device in a scatternet whose *CK* value is NULL or matches a 01* sequence is known as a constructor. Symbols $Constructor(I)_d$ and $Constructor(IS)_d$ represent constructor d in the inquiry and inquiry scan states, respectively.

Notably, the proposed protocol guarantees that, at any particular time, only one constructor is present in a scatternet. This constructor will connect to another scatternet's constructor which has the same *DOC* value.

The proposed protocol consists of three main phases. In Phase I, every device maintains the information on *CK*. An n-degree BlueCube structure can be constructed by connecting two $(n-1)$-degree subcubes. In constructing the n-degree BlueCube, one constructor represents the connected $(n-1)$-degree subcube and is connected to another constructor of connected $(n-1)$-degree subcube. Every device in a connected subcube uses its *CK* value to determine which one is the constructor of this scatternet. The encoding of the *CK* value guarantees that exactly one constructor is present in the current scatternet. When two scatternets' constructors intend to connect with each other, their *DOC* values are compared. They can only form a larger scatternet if their *DOC* values are the same. If yes, they have the same degree and can establish a larger Hypercube. The two constructors may exchange *DOC* values through the two fields '*Undefined*' and '*AM_ADDR*' in the FHS packet. The general rules that must be followed by all devices to ensure that their *CK* and *DOC* values are maintained are stated below. Assume that the number of devices required to construct a BlueCube is n. The three-phase proposed protocol is also described below with some examples.

3.1. Ring construction phase (Phase I)

Initially, every device sets its *DOC* to zero and *CK* to NULL. The ring construction procedure includes the following steps and stops only when the number of devices in a ring scatternet exceeds half of the total number of devices.

3.1.1. Ring construction procedure (RCP)

Step 1: Every device attempts to construct a ring scatternet. The *DOC* is used to evaluate the number of devices, $k = 2^{DOC}$, in the connected scatternet.

Step 2: If $k \leqslant n/2$ then the device performs Step 3. Otherwise, it proceeds to Step 8 to complete *RCP*, and then enters Phase II.

Step 3: Every device uses its *CK* value to determine which one is a constructor of its scatternet. A device whose *CK* value is NULL or matches a 01* sequence is a constructor and proceeds to Step 4. Otherwise, it enters the waiting mode to wait for information from the constructor.

Step 4: The constructor randomly determines whether it should enter the inquiry or inquiry scan state, and then links with the constructor of other scatternets. $Constructor(I)_a$ sends out an ID packet. On receiving the ID packet from $Constructor(I)_a$, $Constructor(IS)_b$ sets the '*Undefined*' and '*AM_ADDR*' fields of the FHS packet to its own *DOC* value, and then transmits the packet back to $Constructor(I)_a$.

Step 5: On receiving the FHS packet, $Constructor(I)_a$ determines whether the received *DOC* is the same as its own *DOC*. If the *DOC* values are identical, then linking these two constructors, $Constructor(I)_a$ and $Constructor(IS)_b$, yields a BlueCube with a higher degree. Thus, device a enters the page state and sends out an ID packet to connect with device b which already entered the page scan state. If the *DOC* values are different, constructors a and b return to Step 4.

Step 6: After $Constructor(I)_a$ and $Constructor(IS)_b$ are connected, device a updates its *CK* value by expanding the bit value 0 at the most significant bit, and device b does the same using the value 1. Both constructors a and b increase their *DOC* values by one to indicate that a BlueCube with a higher degree can be constructed.

Step 7: Constructors a and b send the expanded bit to every device in the scatternet before linkage. Every other device in the waiting mode follows the constructor to update the *CK* value by expanding one bit at the most significant bit. Their *DOC* values are increased by one for yielding a value equal to that of the constructor. Every device, after its *DOC* and *CK* values are modified, repeats Step 1.

Step 8: Two devices whose *CK* values are 01* and 11* in the scatternet randomly enter the inquiry or inquiry scan state to connect with each other in order to construct a ring scatternet.

Initially, every device sets its *DOC* to zero and *CK* to NULL, as shown in Fig. 6(a). Devices *A* and *B* are used as examples. Each device executes Steps 1, 2 and 3. Initially, each device evaluates $2^{DOC} = 1$ and retains this value as the number of the device in a linked scatternet. Since the initial *CK* value of each device is NULL, each device plays the role of constructor and executes Steps 4 and 5. In Step 4, all devices randomly determine whether they should enter the inquiry state or the inquiry scan state. They try to connect with other devices of the same *DOC* value to form a piconet with twin devices as shown in Fig. 6(b). When the link was established, all the linked devices proceeded to Step 6. Device *B* already formed a link in the inquiry state and hence, its *CK* value was expanded to 0 and its *DOC* value was increased to one. Device *A* constructed a link in the inquiry scan state and thus, its *CK* value was expanded to 1 while its *DOC* value was increased to one. Afterwards, devices *A* and *B* proceeded to Step 7 to adjust the other devices' *CK* and *DOC* values, but no operation was done since devices *A* and *B* were not connected to any device.

As shown in Fig. 7(a), the *CK* and *DOC* values of devices *B* and *C*, respectively, are all 0's and 1. Hence, they become

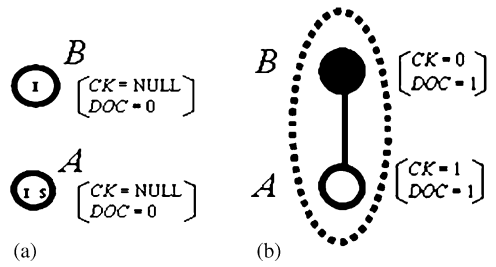

Fig. 6. Initial connection of devices A and B: (a) initial state; (b) piconet formed by twin devices.

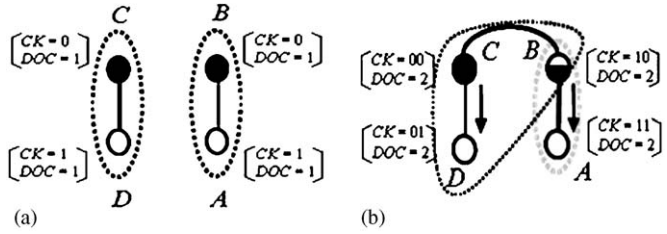

Fig. 7. Connected linkage for the second time in Phase I: (a) ring scatternet formation at $CK = 0$ and $DOC = 1$; (b) connection of devices B and C and modification of CK and DOC values.

constructors of their scatternets. In Step 4, devices B and C, which are now masters, randomly determine whether to enter the inquiry or the inquiry scan state, and attempt to link to other devices. The device whose CK value is inconsistent with the 01* sequence stays in the waiting mode. In Fig. 7(b), devices B and C are taken as examples to explain the connection procedure. After executing Step 5, devices C and B enter the inquiry and inquiry scan states, respectively. Devices B and C establish a connection with each other since their DOC values are equal. Then they proceed to Step 6. Device C is in the inquiry state during the connection and hence, its CK value is expanded to 00 while the DOC value is increased by one. Device B constructs the link in the inquiry scan state and hence, its CK value is expanded to 10 while its DOC value is increased by one. In Fig. 7(b) the arrows signify that devices B and C perform Step 7 to inform all devices in the scatternet to change the CK and DOC values. Each device in the waiting mode follows the original constructor while expanding its CK value by the same bit and increasing the DOC value by one. The resulting DOC values of all devices in the waiting state are now the same as those of the constructors.

Fig. 8(a) shows the formation of a ring scatternet at $CK = 01$ and $DOC = 2$ according to the same RCP. Devices E and D play the role of constructors in their scatternets and link up because they have the same DOC value. However, unlike in the previous step, both constructors D and E act as slaves before they are connected. Thus, device E adopts the role of master and constructs a new piconet after being connected to device D and yields the results plotted as black dotted lines in Fig. 8(b). Afterwards, device E expands its CK value to 001 and increases its DOC to three. Similarly, device D expands its CK value to 101 and increases its DOC value to three.

Fig. 9(a) shows the formation of a ring scatternet at $CK = 011$ and $DOC = 3$, following the same RCP. Devices H and I are constructors of their scatternets and link up because they have the same DOC value. Fig. 9(b) uses the devices H and I as examples to illustrate the linking procedure. They implement Step 6. During the connection, device H is in the inquiry state and expands its CK value to 0011 and increases its DOC value to four. Similarly, device I is in the inquiry scan state, and expands its CK value to 1011 and increases its DOC value to four. The arrows in Fig. 9(b) show that, after executing Step 7, both devices H and I transmit the new CK and DOC values to all devices in the scatternet in which they were previously included. Each device in the waiting mode modifies its own CK and DOC values according to the newly received CK and DOC values.

Each device in the constructed chain scatternet now executes Steps 1 and 2. Then it knows that the number of connected devices is 16, or 2^{DOC}, which exceeds half of the total number of devices. Fig. 10 shows that every device performs Step 8 to form a ring scatternet. Device A, which has a CK value of 0111, will link to device P, whose CK value is 1111. Thus, the beginning of the chain can be connected to its end. Fig. 11 shows the resultant ring scatternet. The other constructors, which are not linked with the ring scatternet, enter Phase II after a timeout.

3.2. Scatternet construction phase (Phase II)

In a Bluetooth network, an excess of piconets can easily cause collision in the hopping sequence and hence, increase the packet loss rate [11,25]. In addition, a study [2] depicts that the small number of bridges in a route increases the success rate of data transmission and saves the guard time. The probability of packet collisions increases with the number of piconets in the constructed BlueCube. Furthermore, the guard time cost decreases with the number of bridges in the BlueCube. Executing role switch operations in Phase II reduces the number of piconets and bridges and thus, increases the success rate and reduces the guard time cost. Another aim of Phase II is that the devices which have not been connected to the ring scatternet will construct a link to a particular piconet of the ring. This Phase comprises two main procedures.

(1) *Role switching procedure* (*RSP*): This procedure applies role switching operations to reduce the number of piconets and bridges. It is implemented by combining piconets, splitting piconets and switching the roles of devices.
(2) *Remaining device connection procedure* (*RDCP*): This procedure enables the devices which have not yet participated in the ring scatternet to connect to the ring such that the number of newly constructed links in all the piconets is approximately the same.

During the execution of the *RSP*, only devices with the role of master (M/S bridge or pure master) can initiate a role switch request. This prevents the master and slave devices from simultaneously initiating a role switch request that would lead to an awkward situation wherein both are waiting for each other to

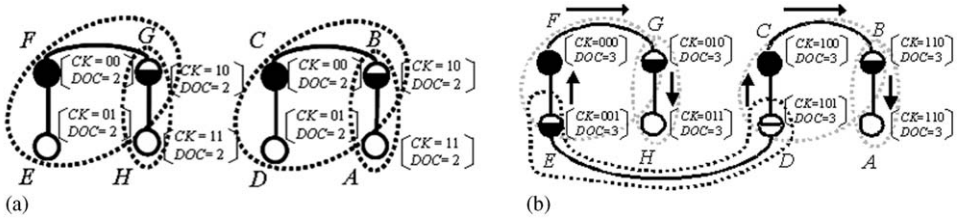

Fig. 8. The third connection built in Phase I: (a) ring scatternet formation at $CK = 01$ and $DOC = 2$; (b) connection of devices D and E and modification of CK and DOC values.

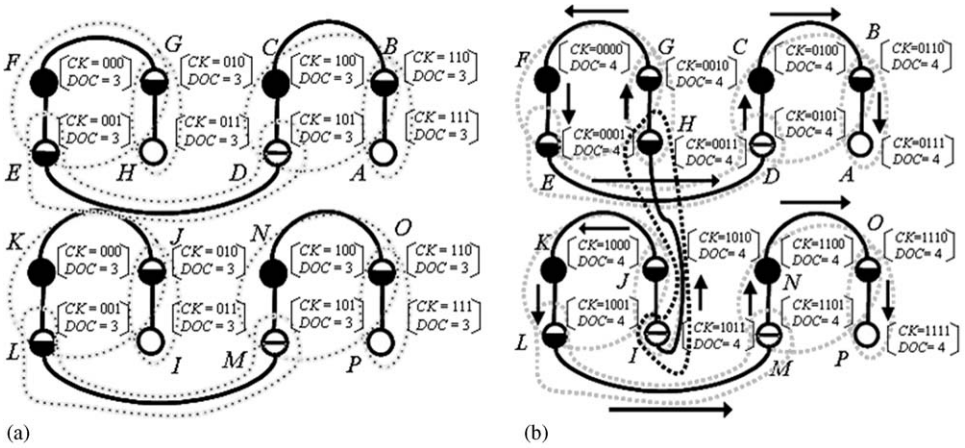

Fig. 9. The fourth connection built in Phase I: (a) ring scatternet formation at $CK = 011$ and $DOC = 3$; (b) connection of devices H and I and modification of CK and DOC values.

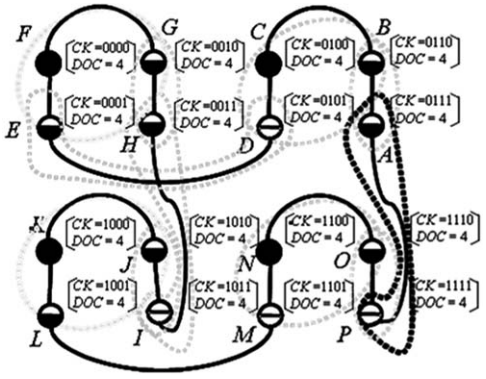

Fig. 10. Devices A and P established a connection to construct a ring scatternet.

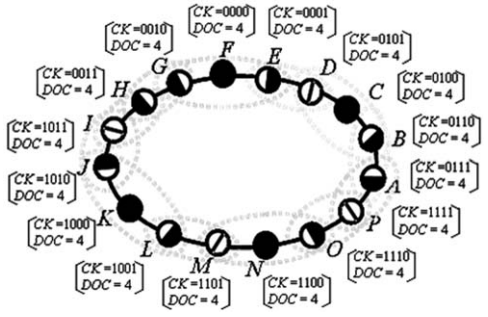

Fig. 11. The resulting ring scatternet constructed in Phase I.

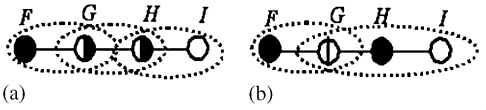

Fig. 12. An example of role switch operations applied on devices G and H in the ring scatternet: (a) the role playing of devices in the scatternet; (b) the role playing after executing the piconet combination.

reply. As shown in Fig. 11, a device that acts as an M/S bridge can take the role of a pure master or S/S bridge such that the role of each device in the ring scatternet eventually alternates between master and S/S bridge. Fig. 12(a) shows devices G and H as examples to illustrate role switching. Devices G and H originally act as M/S bridges and connect to devices F and I, respectively. Three piconets are constructed from four devices in the original ring scatternet. Here, devices G and H change their roles to S/S bridge and master, respectively, and hence, yield two piconets in the ring scatternet. The number of piconets and bridge devices can then be reduced by one, as shown in Fig. 12(b). The term *Subsequence of CK value* is defined to illustrate the operation of Phase II.

Definition (*Subsequence of CK value*). Let the m-bit CK value S be $S_1S_2 \ldots S_m$, where $S_i = 0$ or 1. The subsequence of

S is defined as $S' = S_i, S_{i+1}, \ldots, S_m$, for all $2 \leqslant i \leqslant m$. For example, the *subsequence* of $CK = 1101$ is {101, 01, 1}.

The *RSP* is described below. The results of the example in Phase I will be used to illustrate Phase II.

3.2.1. Role switching procedure

Step 1: A device that plays the role of either the master or M/S bridge applies the following *Role Determination* (*RD*) formula to determine which of the two roles it should adopt.

$$RD(CK) = \begin{cases} \text{S/S bridge} & \text{if number of 1's in } CK \text{ is odd,} \\ \text{Master} & \text{if number of 1's in } CK \text{ is even.} \end{cases}$$

Step 2: A master device that intends to act as the S/S bridge will initially send an LMP_Switch_req message to its slave with smaller *CK* value in order to request a role switching. After the master exchanges its role with that of the slave with smaller *CK* value, it will request another role switching with the other slave.

Step 3: An M/S bridge that intends to act as a S/S bridge will switch role with its slave device in piconet.

Step 4: Once the devices have completed role switching, the master devices initiate *RDCP* to link with other devices which have not yet been connected into the ring. Devices that have become S/S bridges remain in the waiting mode to wait for the master to complete Phase II.

The example presented in the previous section is now used to clarify the *RSP*. As shown in Fig. 11, device C intends to act as a S/S bridge. According to Step 2 of *RSP*, device C initially switches its role with device D and changes its role from master to M/S bridge. As a result, piconet $P1$ contains devices C and D and a new piconet $P4$ containing devices C and B is constructed as shown in Fig. 13. In piconet $P1$, devices C and D become M/S bridges. Device C then switches its role again with device B. When device B accepts the request sent by device C, piconets $P3$ and $P4$ are combined into the new piconet $P3$ as plotted by the black dotted lines in Fig. 14 and device C becomes an S/S bridge. Since device B has changed its role from master to S/S bridge, it skips Step 3 and enters waiting mode as described in Step 4. Then device E will intend to change its role from M/S bridge to S/S bridge. According to Step 3 of *RSP*, device E switches its role with device D and hence, piconets $P1$ and $P2$ are combined into a new piconet as shown in Fig. 15. Similarly, the other devices in the ring scatternet follow the operations of the *RSP* to change their device roles. The *RSP* operations cause the role of each device in the ring scatternet to eventually alternate between master and S/S bridge. Thus, the number of piconets and bridges are decreased from 12 to eight as shown in Fig. 15.

When the *RSP* is complete, the master devices and the devices which have not yet been connected to the ring will implement the *RDCP*. The master devices will begin linking up with the devices that have not been connected to the ring scatternet. The unconnected devices can connect to the masters in the ring according to their *CK* values in such a way that the new links are distributed equally in the existing piconets. The steps of

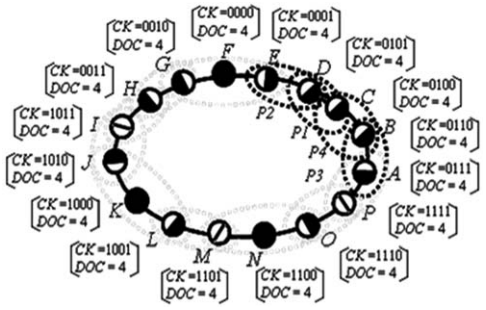

Fig. 13. Piconet splitting in the ring scatternet.

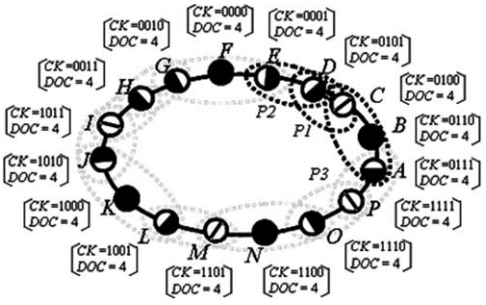

Fig. 14. Piconet combination in the ring scatternet.

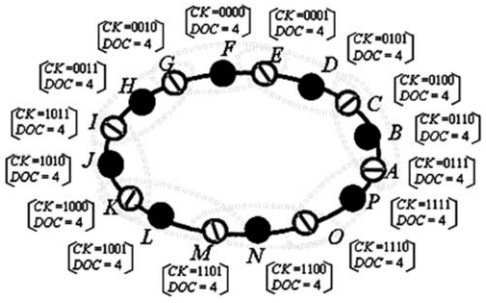

Fig. 15. The role of each device after executing the *RSP*.

RDCP are described below and explained with reference to the example in Phase I.

3.2.2. Procedure for connecting the remaining devices

Step 1: The formula below, which is in terms of *DOC* and n, yields the maximum number called the collection number $CN(n, DOC)$ of devices that the master can collect.

$$CN(n, DOC) = \left\lceil \frac{n - 2^{DOC}}{2^{DOC-1}} \right\rceil.$$

Step 2: The master device checks whether the number of collected devices has reached *CN*. If it does, it proceeds to Step 8. Phase III is then implemented. Otherwise, it proceeds to Step 3.

Step 3: The master sends an ID packet to search for devices that have not yet been connected to the ring scatternet.

Step 4: Devices that have not yet been connected to the ring scatternet are still in Phase I, and may have formed a scatternet

or remained isolated. The master device in the ring scatternet, upon receiving the FHS packet sent as a reply from the constructor or the isolated point, can then check the constructor's *DOC* value against its own value. The difference between the values indicates that the constructor cannot still be connected to the ring scatternet. Subsequently, one party enters the page state and the other enters the page scan state, to establish a connection. However, if the master has not received any reply from the slave after the timeout, then all devices have been linked to the ring scatternet, and the master proceeds to Step 8 and then enters Phase III.

Step 5: After the link has been established, the master sends a subsequence of its *CK* value to the newly connected device as the new *CK* value of that device. Notably, the sent subsequence element will be marked to enable the master to send an unmarked subsequence to a newly connected device later on. Thereafter, the master device repeats Step 2. Devices that are newly linked to the ring scatternet proceed to Step 6.

Step 6: A device that is linked to the ring scatternet sends an inquiry scan request packet to all devices in the ring scatternet to which it originally belonged in order to inform all devices to enter the inquiry scan state.

Step 7: A device that receives an inquiry scan request packet will break its current link and become an isolated point. Then, it will enter the inquiry scan state and wait in order to accept the ID packet sent out by the master in the ring scatternet.

Step 8: The device completes the *RDCP* and enters the waiting mode.

As shown in Fig. 16, when device N changes its role, it will enter the inquiry state and begin executing the *RDCP*. In Step 1, device N evaluates the $CN(n, DOC)$ with $n = 25$ and $DOC = 4$ and hence, understands that no more than two devices can be collected. In Steps 3 and 4, device N connects to device V which is assumed to be in the inquiry scan state. The *CK* value of device N is 1100 and thus, its subsequence is $\{100, 00, 0\}$. In Step 5, device N assigns and sends a new *CK* value of 100 to device V. In Step 6, device V sends out the inquiry scan request packets to inform all the devices $\{Q, R, S, T, U, W, Y\}$ which belong to its original scatternet to enter the inquiry scan state. In Step 7, upon receiving the inquiry scan request packet, the informed devices cut all links with themselves and become isolated. Then, these devices enter the inquiry scan state and try to connect with the ring scatternet. After devices N and V have established a connection, device N checks whether the number of collected devices is *CN*. Device N has collected only one device and thus, has not reached $CN = 2$. Hence, it attempts to enter the inquiry state in order to collect another device. Device N is assumed not to receive any return message from any device after the timeout, and then enters the waiting mode to facilitate Phase III.

Like device N, other master devices in the ring scatternet, including B, D, F, H, J, L and P, enter the inquiry state to connect the isolated devices and link all of the devices into a single scatternet, as shown in Fig. 17. When the masters in the ring scatternet connect the desired number of devices, or when the timeout for connecting devices occurs, Phase II ends while Phase III begins.

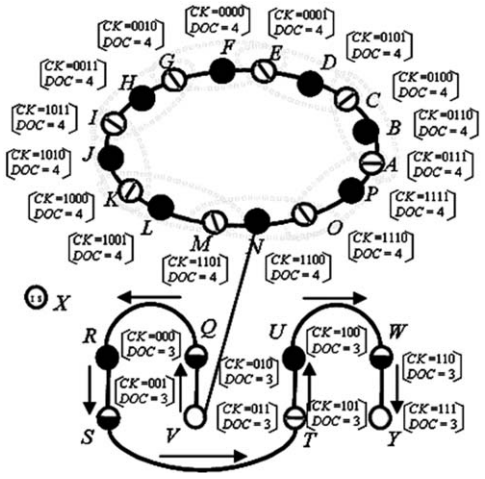

Fig. 16. Linking of devices N and V.

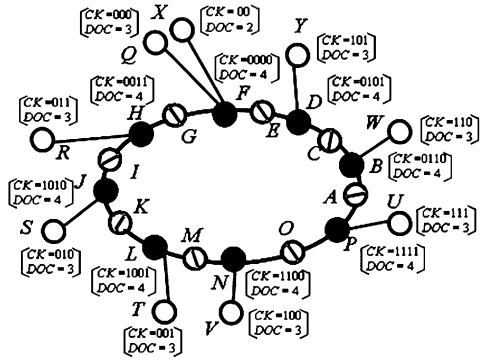

Fig. 17. All devices linking as a single scatternet.

3.3. BlueCube construction phase (Phase III)

When Phase II is completed, all devices in the ring scatternet enter the BlueCube construction phase. The following definitions are introduced to detail Phase III.

Definition (*Hamming distance*). The *Hamming distance*, $HD(A, B)$, is the number of distinct corresponding bits between two *CK* values, A and B. Two devices with a HD of one will connect to each other to establish a BlueCube structure.

Definition (*BlueCube construction information packet (BCI packet)*). The format of a BCI packet is shown below. In constructing a BlueCube structure, the master whose *CK* value is zero sends out a *BCI* packet in the ring scatternet in order to collect the *CK*, *Clk_offset* and *BD_ADDR* from every S/S bridge device.

The goal of Phase III is to construct a BlueCube that is structurally similar to a Hypercube so as to have its favorable characteristics. Devices in the ring scatternet use the *BCI* packet to exchange *CK*, *Clk_offset* and *BD_ADDR*. If the HD of two *CK* values is one, then the information from *Clk_offset* and

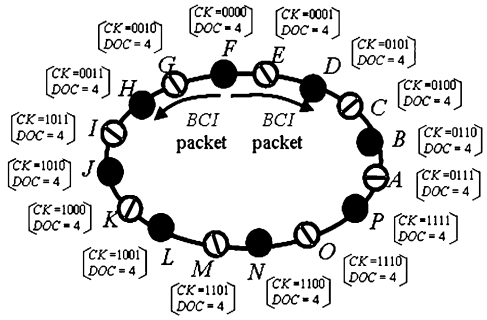

Fig. 18. Directions of *BCI* Packet sent in the ring scatternet.

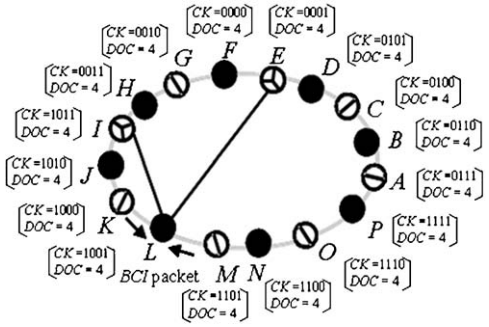

Fig. 19. The master device links with those bridge devices whose *CK* value has a hamming distance of 1 with the master's *CK* value.

BD_ADDR is used to make a connection quickly. Phase III is described below.

3.3.1. BlueCube construction procedure

Step 1: In a ring scatternet, the device whose *CK* value is zero sends out *BCI* packets to the master and S/S bridge devices in both clockwise and counter-clockwise directions in the ring scatternet.

Step 2: Upon receiving the *BCI* packet sent from one neighboring master, the bridge adds the information, including *CK*, *Clk_offset* and *BD_ADDR*, to the *BCI* packet, and sends it to another neighboring master device. Then, the S/S bridge switches alternately between the page scan state and active mode. The S/S bridge either waits to receive the *BCI* packets in active mode, or connects to other devices at the page scan state in order to construct a BlueCube.

Step 3: Upon receiving the *BCI* packet sent from one neighboring bridge, the master device checks each *CK* value, e.g., v, in the *BCI* packet and determines whether the HD between v and its own *CK* value is one. If this is the case and the corresponding devices are not connected in previous phase, then the master device records their *Clk_offset* and *BD_ADDR* values from the *BCI* packet, and sends this *BCI* packet to another neighboring bridge device before entering the page state. The recorded *Clk_offset* and *BD_ADDR* values are used to connect quickly to the device whose *CK* value has a HD of one. The *BCI* packets are sent clockwise and counter-clockwise in the ring scatternet. When a *BCI* packet eventually returns to the device whose *CK* value is zero, the transmission of the *BCI* packet is halted.

Step 4: After a link is established, the master device determines whether it is linked to fewer devices than the *DOC* value. If yes, the master device enters the waiting mode upon receiving another *BCI* packet and repeats Step 2.

As shown in Fig. 18, in Step 1, master F with a *CK* value of zero sends *BCI* packets to neighboring devices, E and G. In Step 2, upon receiving the *BCI* packet, devices E and G append their *CK*, *Clk_offset* and *BD_ADDR* to the packet and forward it to neighboring devices D and H, respectively. Afterwards, devices E and G alternate between the page scan state and the active mode. The forwarding of *BCI* packets ends only when the packets have returned to master F.

In Fig. 19, the *CK* values of devices E and M are 0001 and 1101, respectively. In Step 3, upon receiving the *BCI* packet from device M, master L determines that $HD(1001, 0001)$ and $HD(1001, 1101)$ are both equal to one. Master L extracts *Clk_offset* and *BD_ADDR* of devices E and M from the *BCI* packet, and then sends the *BCI* packet to bridge K. Then master L uses the recorded *Clk_offset* and *BD_ADDR* of device E to establish a connection with device E, which has already entered the page scan state. Although device M also fulfills the requirements to connect with master L, devices M and L are already connected to the ring scatternet. Hence, device L does not reestablish a link with device M. Now, only the three devices K, M and E are linked to device L. Since the number of links connected to device L is not equal to the *DOC* value, master L continues to wait to receive another *BCI* packet. Once it receives another *BCI* packet from device K, it checks the *CK* values in the *BCI* packet, and determines that $HD(1001, 1011)$ and $HD(1001, 1000)$ are both one. As indicated above, master L sends out the *BCI* packet to master M and then connects it to device I. Since the number of links to master L equals its *DOC* value, device L terminates the BlueCube construction phase.

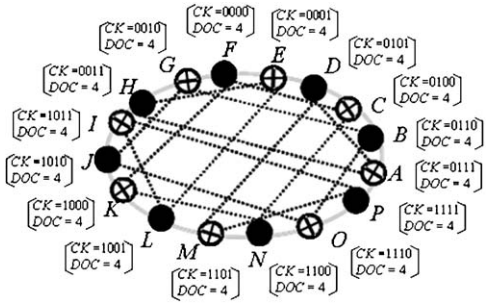

Fig. 20. Completion of connection of all master devices in the ring scatternet.

In Phase III, every master repeatedly establishes links until its number of connections equals the *DOC* value. When all devices in the ring scatternet have completed the BlueCube construction phase, the resultant ring scatternet is as shown in Fig. 20. Fig. 21 presents another perspective of the constructed BlueCube scatternet, in which the backbone of a Hypercube structure has been constructed to provide slaves with a good environment for computation and communication.

Note that, although a master can only connect to seven slaves, there is no limitation in the number of masters to which the

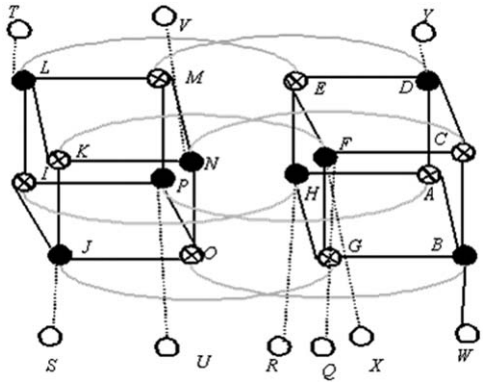

Fig. 21. A constructed BlueCube scatternet where a Hypercube backbone has been constructed for providing slaves with a parallel computing and communication environment.

slave could connect. It is possible to construct a BlueCube with a dimension higher than 8. Constructing a BlueCube with higher dimensions requires extra operations in Step 4 in the *BlueCube construction procedure* to arrange the role of the devices. In the extra operations, the master device checks if its *DOC* value is greater than seven. If yes, the master device applies the role switching operation to switch its role with the connected device during the link establishment. As a result, the master will become an M/S bridge since it uses the master role to connect with seven slaves in the original piconet, but uses the slave role to establish new links. However, M/S bridges reduce the piconet performance since intra-piconet communication must be suspended when the master participate and play a slave role in another piconet and hence, results in inefficient communication. In the present paper, we aim to construct the BlueCube with a reasonably sized scatternet that includes less than 128 devices in the personal area and uses only the roles of master, S/S bridge and slave.

4. Performance

This section investigates the performance of the constructed BlueCube using a Java-based simulator. The size of the simulation region is set to 10×10 units, whereas the range of the radio transmission is set to a constant of 10 units. The number of devices varies from 10 to 80, and their locations, *BD_ADDRs* and native clocks are randomly chosen. The source and destination of each route are randomly selected from the scatternet, and the routing protocol proposed in [1] is used to establish the route. The constant bit rate (CBR) model is utilized to generate the traffic load for each route. Each performance result was obtained from the average results of 100 experiments. To evaluate the effectiveness of fault-tolerance, *the device failure ratio*, which is defined by the ratio of the number of failure devices to the number of all devices, is investigated. The performance of the proposed BlueCube and three existing structures—Mesh [10], Mesh_h and Seven_Ary Star [16]—is examined. The topology of the Mesh_h structure has the same number of slave devices and internal bridges as BlueCube, and its internal bridges are linked in a mesh. The performance is

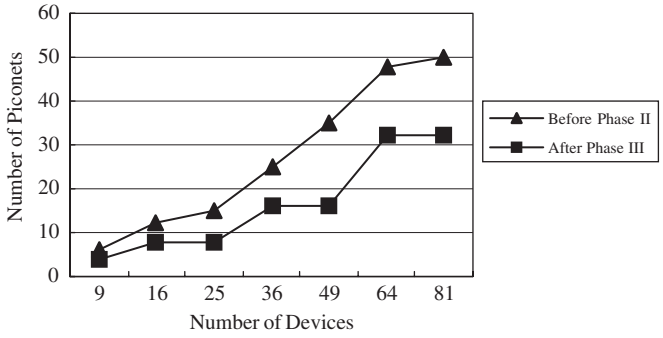

Fig. 22. The effect of piconet reduction in the BlueCube protocol.

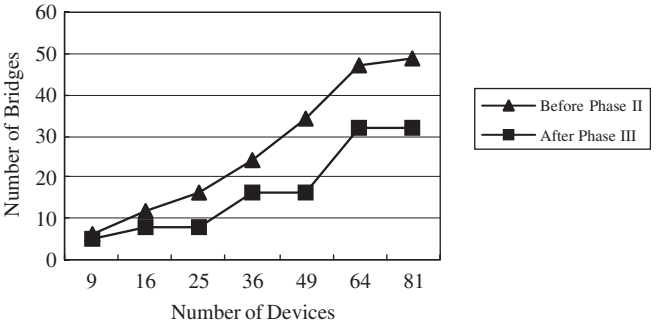

Fig. 23. The effect of bridge reduction in the BlueCube protocol.

measured by the number of piconets, the number of bridges, the average length of the routing paths, the average time for route construction, the congestion rate for communication, the effectiveness of disjoint routes and fault-tolerance.

Increasing the number of piconets in a specific region increases the packet collision rate and hence, increases packet retransmission. Fig. 22 shows the effect of role switching operations in Phase II of the proposed protocol. The number of piconets increases with the number of devices. Applying the role switching operations markedly reduced the number of piconets. An excess of bridges in the scatternet wastes the guard time required to switch among the participating piconets and increases the master's difficulty in scheduling. Reducing the number of bridges in the scatternet reduces the delay in inter-piconet data transmission and hence, increases the throughput of the scatternet. Fig. 23 shows the effect of applying role switching operations to reduce the number of bridges. The proposed role switching operation in Phase II significantly reduces the number of bridges. This saves the guard time and increases the rate of successful data transmission and thus, improves the performance of the network.

Figs. 24 and 25 show the number of piconets and bridges associated with various scatternet structures, including BlueCube, Mesh, Mesh_h and Star. A centralized formation algorithm [16] is applied to construct a scatternet with a Seven_Ary Star structure. The Star structure has fewer piconets than BlueCube, Mesh and Mesh_h. The centralized collection of information about the number of devices helps to control the construction of the star structure, but it creates an overhead of

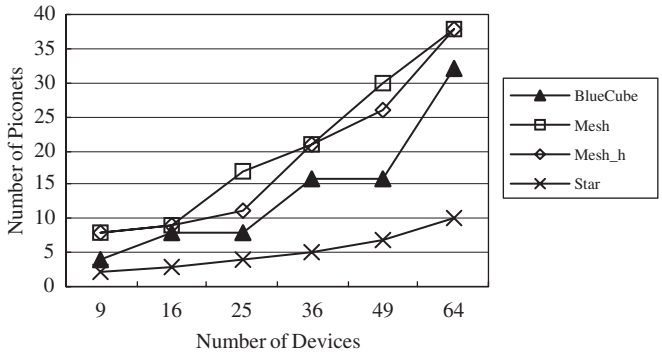

Fig. 24. Comparison of the average number of piconets constructed for various scatternet structures.

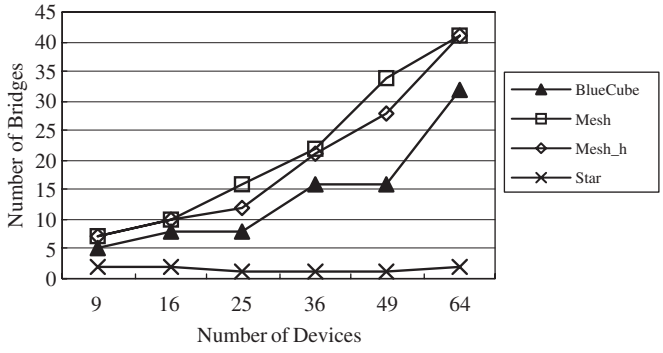

Fig. 25. Comparison of the average number of bridges constructed for various scatternet structures.

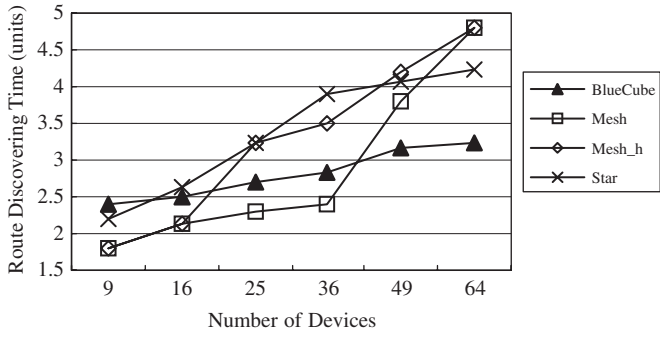

Fig. 26. Comparison of average time for route discovering for various scatternet structures.

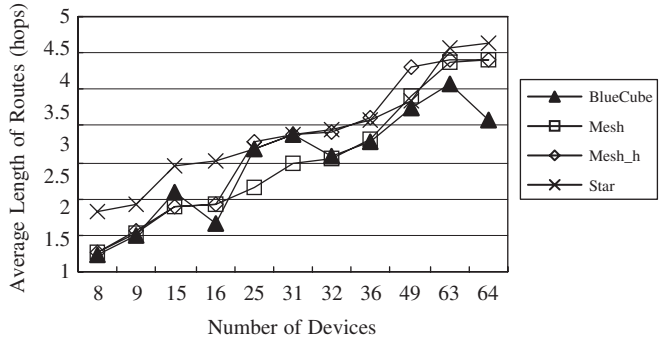

Fig. 27. Comparison of average routing length discovered for various scatternet structures.

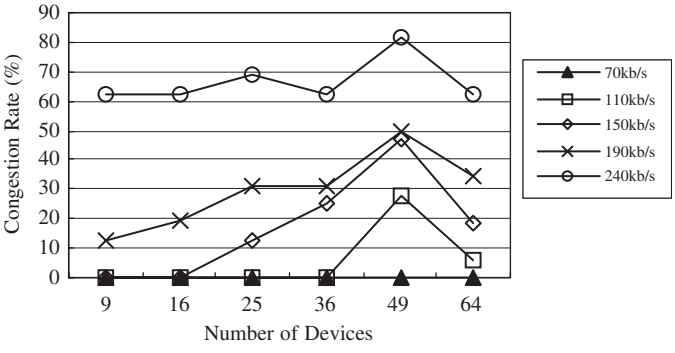

Fig. 28. Comparison of the congestion rate of BlueCube structure for various traffic loads.

control packets for collecting information. The proposed BlueCube construction protocol applies role switching operations, such as piconet combination and piconet splitting, to reduce the number of piconets and bridges in a distributed manner. As shown in Figs. 24 and 25, BlueCube has fewer piconets and bridges than those of Mesh or Mesh_h. Thus, it suffers fewer collisions in the hopping sequence and has an easier scheduling.

Two devices which are in different piconets but intend to communicate must establish a routing path and then use the master and bridge devices to forward packets. Properly structuring the scatternet reduces the length of the route and provides disjoint paths and hence, reduces the transmission delay. Fig. 26 compares the route-discovery times of various scatternet structures using the RVM [1] and the proposed BlueCube with 9, 16, 25, 36, 49 and 64 devices. On the average, the BlueCube structure requires less time to discover a route compared to the other structures, because the Hypercube structure has the smallest network diameter. Fig. 27 shows the average route length of various scatternet structures. The routing length typically increases with the number of devices. The Hypercube structure has the smallest network diameter. Thus, the average length of the routes constructed by BlueCube is lower than those constructed by Mesh, Mesh_h and Star.

Reducing the average routing length also reduces the average network traffic. The congestion rate is the ratio of the number of relaying devices which suffer transmission congestion to the total number of relaying devices. Reducing the congestion rate also reduces the delivery delay and the packet loss rate in inter-piconet communication. Fig. 28 shows the congestion rate in BlueCube for various traffic loads. In Fig. 28, the sources and destinations of the routes are randomly selected and the bandwidth required for the route is determined using the CBR model to generate the traffic loads from 70 to 240 Kb/s.

In general, the congestion rate increases with the traffic load. Notice that most curves have high congestion rates when the number of devices is equal to 25 or 49. The reason for this is

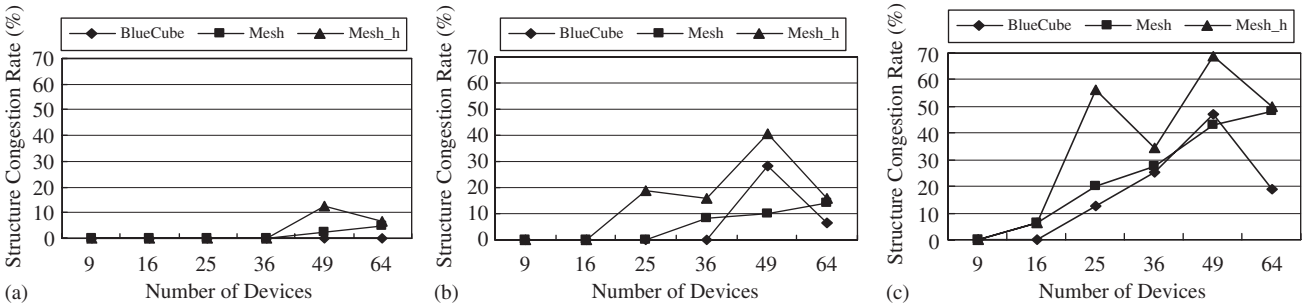

Fig. 29. Comparison of the congestion rate of different structures for various traffic loads: (a) traffic load at 70 Kb/s; (b) traffic load at 110 Kb/s; (c) traffic load at 150 Kb/s.

analyzed below. The excess devices in the BlueCube increase the number of slave devices in a piconet. The degree of traffic congestion at the master device increases with the number of slaves in a piconet. The constructed BlueCubes with 9, 16, 25, 36, 49 or 64 devices have the number of slave devices 1, 0, 9, 4, 17 or 0, respectively. Hence, the constructed BlueCubes with 25 or 49 devices have a higher congestion rate than the BlueCubes constructed with 9, 16, 36 or 64 devices. Fig. 30 indicates that BlueCube has a low congestion rate when the traffic on each route is under 110 Kb/s. Fig. 29 uses different traffic loads to compare the congestion rates of BlueCube, Mesh and Mesh_h structures. In Figs. 29(a)–(c), the traffic loads are controlled at 70, 110, and 150 Kb/s, respectively. In general, the congestion rate increases with the traffic load for all structures. Both Mesh and Mesh_h typically exhibit more congestion than BlueCube under different traffic loads because the average routing length discovered in BlueCube is smaller than those discovered in Mesh or Mesh_h. In particular, the average traffic at 70 Kb/s in Mesh or Mesh_h has the same congestion as those at 110 Kb/s in BlueCube. Moreover, BlueCube constructed with 64 devices at 150 Kb/s suffers half the congestion rate of Mesh or Mesh_h with the same number of devices.

A well-structured scatternet should also support disjoint paths between any pair of devices. Disjoint paths provide not only backup routes but also higher bandwidths because data can be transmitted through disjoint paths in parallel. The backup path immediately supports the data communication service as another transmission route if the original route breaks and thus, faults caused by route breakage can be tolerated. Let α be the ratio of the number of slave devices to the number of internal bridges. If the number of bridges (or internal bridges) is high, then many disjoint routes are candidate backup paths. Thus, a smaller α value means that more candidate main routes or backup routes are available. A route is said to be tolerated if the packet transmissions are always successful due to the existence of disjoint routes. The device fault-tolerance rate (DFTR) is defined by the ratio of the number of tolerated routes to the number of all routes at a specific device failure ratio. The number of devices is set to 16, 36, 50 and 66 to represent various α values. The relationship between DFTR and the device failure ratio is examined by setting the device failure ratio to 20%, 30% and 40%. Fig. 30 shows that

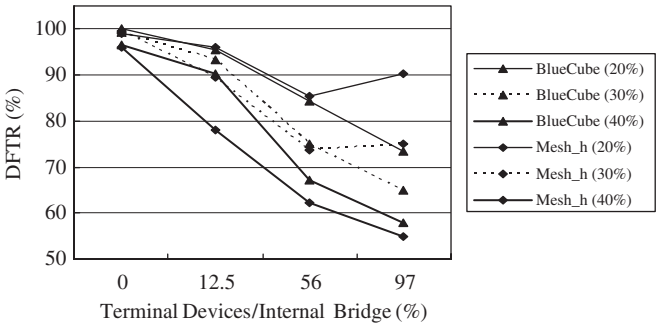

Fig. 30. Comparison of DFTR for communicating at various device failure ratios in the BlueCube.

the fault-tolerance generally falls as α value increases. Both Mesh_h and BlueCube structures exhibit this trend. The DFTR generally falls as the device failure ratio increases. However, BlueCube outperforms Mesh_h in terms of DFTR, for the following reasons. A structure with more disjoint paths has more backup routes and hence, can better tolerate device faults. When few devices are involved, each device in the Mesh_h structure has a larger degree than in the BlueCube structure and thus, Mesh_h tolerates faults better than BlueCube. However, the degree of devices in Mesh_h cannot exceed four. When the number of devices is large, BlueCube tolerates faults better.

The device fault-tolerance overhead (DFTO), which is related to the DFTR, is used to evaluate the overhead associated with the extra length beyond that of the original path for a backup route. The DFTO is the ratio of the number of hops on the backup path to those on the original path. A smaller DFTO means a lower overhead associated with the backup path when the bridge devices fail. Fig. 31 compares the lengths of the backup paths. The backup paths in BlueCube are shorter than those in Mesh_h because the Hypercube structure has the smallest network diameter.

The probability that a route fault occurs increases with the device failure ratio. A high device failure ratio requires rerouting and hence, increases the traffic load in the scatternet. Fig. 32 shows the DFTR of BlueCube and Mesh_h against the number of devices from 9 to 64 for device failure ratios of 20%, 30% and 40%. Fig. 32 shows that Mesh_h has a better fault-tolerance

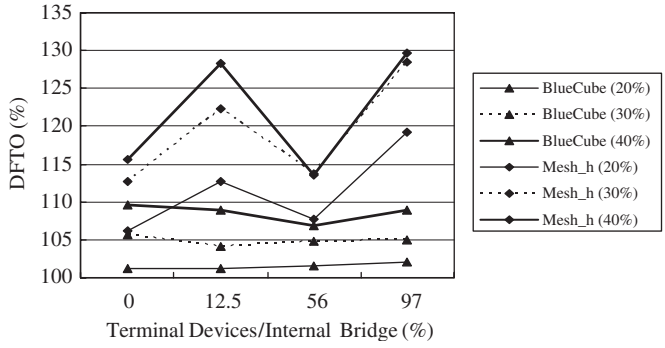

Fig. 31. Comparison of DFTO for communicating at various device failure ratios in the BlueCube.

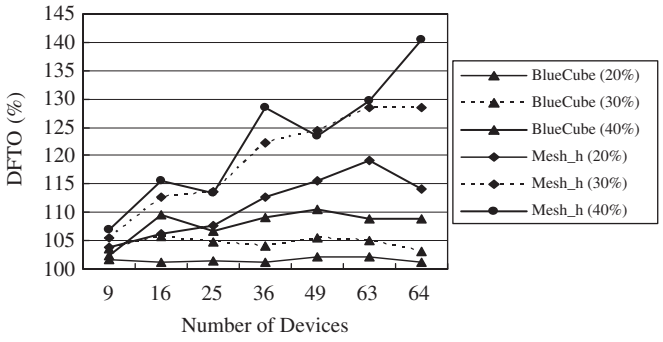

Fig. 33. Comparison of the fault-tolerance overhead for communication at various device failure ratios in structures.

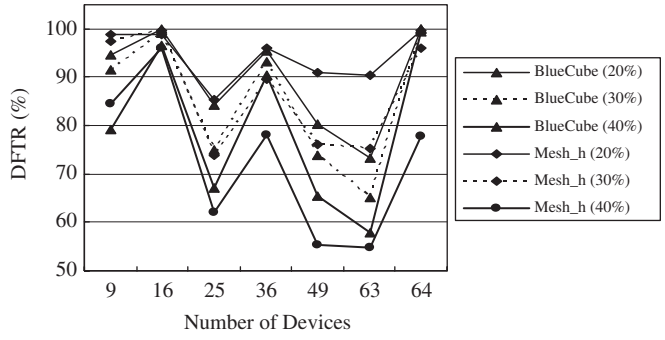

Fig. 32. Comparison of the fault-tolerance rate for communication at various device failure ratios in structures.

than BlueCube at low device failure ratios. However, as the device failure ratio increases, the fault-tolerance of BlueCube exceeds that of Mesh_h. Fig. 30 shows the same results. This result is quite reasonable. The number of dimensions of BlueCube increases with the number of devices. If the number of devices is large, the dimensionality of BlueCube is high and thus, BlueCube provides more disjoint routes to help tolerate route faults. Fig. 33 compares the fault-tolerance overhead, DFTO, for backup paths in various numbers of devices. Increasing the number of devices increases the number of disjoint routes for tolerating route faults and hence, reduces the fault-tolerance overhead. The probability of route fault increases with the device failure ratio and thus, the fault-tolerance overhead also increases with the device failure ratio. Fig. 33 shows that the fault-tolerance overhead markedly increases with the number of devices in Mesh_h, but remains steady in BlueCube. Increasing the number of devices in scatternet also increases the average degree of internal bridges in the BlueCube structure. Thus, the lengths of the backup paths in BlueCube are steady. The simulation results in Figs. 31 and 33 show that the backup paths in BlueCube are around half of those in Mesh_h, regardless of α and the number of devices. The fault-tolerance overhead increases with the device failure ratio in both BlueCube and Mesh_h structures. However, BlueCube outperforms Mesh_h in fault-tolerance overhead even when the device failure ratios of BlueCube and the Mesh_h are 40% and 20%, respectively.

5. Conclusions

The present work extends the network model of grid computing from a wired network to an integrated wired and wireless network via a pilot application of Bluetooth wireless technology. A BlueCube protocol is proposed for constructing a Hypercube structure by linking various Bluetooth machines. In the constructed BlueCube structure, machines are able to communicate with each other efficiently and utilize the wireless bandwidth and thus, increase the potential of parallel computing and communication. A three-phase protocol is proposed to enable Bluetooth devices to construct a parallel computing and communication environment. Existing algorithms designed for Hypercube can therefore be applied to the BlueCube scatternet. The BlueCube scatternet includes no extra bridges between two piconets and supports disjoint paths and shorter routing paths. Experimental results reveal that the proposed protocol yields a scatternet structure with favorable characteristics for parallel computing and communication.

References

[1] P. Bhagwat, A. Segall, A routing vector method (RVM) for routing in Bluetooth scatternets, Proceedings of the Sixth IEEE International Workshop on Mobile Multimedia Communications (MoMuC), November 1999, pp. 375–379.

[2] C. Y. Chang, G. J. Yu, C. F. Lin, T. T. Wu, Relay reduction and route construction for scatternet over Bluetooth radio systems, Proceedings of the IEEE 16th International Conference on Information Networking (ICOIN), January 2002, pp. 5B2.1–5B2.10.

[3] S. W. Cheng, D. Garlan, B. Schmerl, P. Steenkiste, Hu. Ningning, Software architecture-based adaptation for grid computing, Proceedings of the 11th IEEE Conference on High Performance Distributed Computing (HPDC), 2002, pp. 389–398.

[4] S. K. Das, D. J. Harvey, R. Biswas, Latency hiding in dynamic partitioning and load balancing of grid computing applications, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), 2001, pp. 347–354.

[5] T. Havashi, K. Nakano, S. Olariu, Randomized initialization protocols for packet radio networks, Proceedings of the 13th International Parallel Processing Symposium (IPPS), April 1999, pp. 554–548.

[6] M. Kalia, S. Garg, R. Shorey, Scatternet structure and inter-piconet communication in the Bluetooth system, Proceedings of the IEEE National Conference on Communications, New Delhi, 2000.
[7] C. Law, K. Y. Siu, A Bluetooth scatternet formation algorithm, Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM), November 2001, pp. 2864–2869.
[8] Y.R. Leu, S.Y. Kuo, A fault-tolerant tree communication scheme for hypercube systems, IEEE Trans. Comput. 45 (6) (1996) 643–650.
[9] T. Y. Lin, Y. C. Tseng, K. M. Chang, A new BlueRing scatternet topology for Bluetooth with its formation, routing, and maintenance protocols, Proceedings of the Wireless Communications and Mobile Computing, June 2003, pp. 517–537.
[10] M. Medidi, A. Daptardar, A distributed algorithm for mesh scatternet formation in Bluetooth networks, Proceedings of the International Conference on Wireless Networks (ICWN) June 2004, pp. 295–301.
[11] Gy. Miklos, A. Racz, Z. Turanyi, A. Valko, P. Johansson, Performance aspects of Bluetooth scatternet formation, Proceedings of the First Annual Workshop on Mobile Ad Hoc Networking and Computing (Mobi Hoc) August 2000, pp. 147–148.
[12] R. Nusser, R. Bosch, R.M. Pelz, Bluetooth-based wireless connectivity in an automotive environment, Proceedings of the IEEE Vehicular Technology Conference (VTC), September 2000, pp. 1935–1942.
[13] P. Zhang, W. Li, J. Wang, Y. Wang, Bluetooth—the fastest developing wireless technology, Proceedings of the International Conference Technology (ICCT), 2000, pp. 1657–1664.
[14] K. Persson, D. Manivannan, Distributed self-healing Bluetooth scatternet formation, Proceedings of the International Conference on Wireless Networks (ICWN) June 2003, pp. 325–334.
[15] C. Petrioli, S. Basagni, BlueMesh: Degree-constrained multihop scatternet formation for Bluetooth networks, ACM/Kluwer Journal on Mobile Networks and Applications (MONET), 9(1) (2004) 33–47.
[16] C. Petrioli, S. Basagni, I. Chlamtac, Configuring bluestars: multihop scatternet formation for Bluetooth networks, IEEE Trans. Comput. 52 (6) (2003) 779–790.
[17] L. Ramachandran, M. Kapoor, A. Sarkar, A. Aggarwal, Clustering algorithms for wireless ad hoc networks, Proceedings of the Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), August 2000, pp. 54–63.
[18] T. Salonidis, P. Bhagwat, L. Tassiulas, Proximity awareness and fast connection establishment in Bluetooth, Proceedings of the First Annual Workshop on Mobile and Ad Hoc Networking and Computing, 2000, pp. 141–142.
[19] T. Salonidis, P. Bhagwat, L. Tassiulas, Distributed topology construction of Bluetooth personal area networks, Proceedings of the (INFOCOM), April 2001, pp. 1577–1586.
[20] A. Sohm, R. Biswas, H. D. Simon, Impact of load balancing on unstructured adaptive grid computations for distributed-memory multiprocessors, Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing, 1996, pp. 26–33.
[21] The Bluetooth Specification, 1.0b & 1.1.
[22] P. J. Wan, L. W. Liu, Y. Yang, Optimal routing based on the super-topology in hypercube WDM networks, Proceedings of the 24th Annual IEEE Conference on Local Computer Network (LCN) 1999, pp. 142–149.
[23] Z. Wang, R. J. Thomas, Z. Haas, Bluenet—a new scatternet formation scheme, Proceedings of the 35th Hawaii International Conference on System Science (HICSS) January 2002, pp. 779–790.
[24] G. V. Zaruba, S. Basagni, I. Chlamtac, Bluetrees—scatternet formation to enable Bluetooth-based ad hoc networks, Proceedings of the IEEE International Conference on Communications (ICC), 2001, pp. 273–277.
[25] S. Zurbes, Considerations on link and system throughput of Bluetooth networks, Proceedings of the 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communication (PIMRC) September 2000, pp. 1315–1319.

Chao-Tsun Chang received the Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2006. He joined the faculty of the Department of Department of Information Management, Hsiuping Institute of Technology, Taiwan, as an Assistant Professor in 2006. His current research interests include wireless sensor networks, Bluetooth radio networks, Ad Hoc wireless networks, and mobile computing.

Chih-Yung Chang received the Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 1995. He joined the faculty of the Department of Computer and Information Science at Aletheia University, Taiwan, as an Assistant Professor in 1997. He was the Chair of the Department of Computer and Information Science, Aletheia University, from August 2000 to July 2002. He is currently an Associate Professor of Department of Computer Science and Information Engineering at Tamkang University, Taiwan. Dr Chang served as an Associate Guest Editor of *Journal of Internet Technology* (JIT, 2004), *Journal of Mobile Multimedia* (JMM, 2005), and a member of Editorial Board of *Tamsui Oxford Journal of Mathematical Sciences* (2001–2005). He was an Area Chair of IEEE *AINA*'2005, Vice Chair of *IEEE WisCom*'2005 and *EUC*'2005, Track Chair (Learning Technology in Education Track) of *IEEE ITRE*'2005, Program Co-Chair of *MNSAT*'2005 and *UbiLearn*' 2006, Workshop Co-Chair of *INA*'2005, *MSEAT*'2003, *MSEAT*'2004, and Publication Chair of *MSEAT*'2005 and *SCORM*'2006. Dr. Chang is a member of the IEEE Computer Society, Communication Society and IEICE society. His current research interests include wireless sensor networks, mobile learning, Bluetooth radio networks, Ad Hoc wireless networks, and mobile computing.

Jang-Ping Sheu received the B.S. degree in computer science from Tamkang University, Taiwan, Republic of China, in 1981, and the M.S. and Ph.D. degrees in computer science from National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively. He joined the faculty of the Department of Electrical Engineering, National Central University, Taiwan, Republic of China, as an Associate Professor in 1987. He is currently a Professor of the Department of Computer Science and Information Engineering and Director of Computer Center, National Central University. He was a Chair of Department of Computer Science and Information Engineering, National Central University from 1997 to 1999. He was a visiting professor at the Department of Electrical and Computer Engineering, University of California, Irvine from July 1999 to April 2000. His current research interests include wireless communications, mobile computing and parallel processing. He was an associate editor of Journal of the Chinese Institute of Electrical Engineering, from 1996 to 2000. He was an associate editor of Journal of Information Science and Engineering from 1996 to 2002. He was an associate editor of Journal of the Chinese Institute of Engineers from 1998 to 2004. He is an associate editor of the IEEE Transactions on Parallel and Distributed Systems and International Journal of Ad Hoc and Ubiquitous Computing. He has served as a Program Chair and Vice Program Chair for a number of international conferences including IEEE ICPADS'02, ICPP'03, and IEEE MSN'05. He received the Distinguished Research Awards of the National Science Council of the Republic of China in 1993–1994, 1995–1996, and 1997–1998. He received the Distinguished Engineering Professor Award of the Chinese Institute of Engineers in 2003. He received the Distinguished Professor award of the National Central University in 2005. Dr. Sheu is a senior member of the IEEE, a member of the ACM, and Phi Tau Phi Society.