An ns-based Bluetooth Topology Construction Simulation Environment

Chia-Jui Hsu
r89012@im.ntu.edu.tw

Yuh-Jzer Joung
joung@ccms.ntu.edu.tw

Department of Information Management, National Taiwan University Taipei, Taiwan

Abstract

Bluetooth is an emerging technology in wireless applications, and many related issues are yet to be explored both in academia and industry. Because of the complexity and the dynamics of computer networks, a good simulation tool plays an important role in the development stage. Of the existing simulation tools, ns *is a popular, open-source package that has a substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. It also has* BlueHoc *as its extension for Bluetooth. Although BlueHoc offers many simulation functions for Bluetooth, all simulations must be done in a virtually fixed topology. Hence simulation about dynamic topology construction—the first and an important step in establishing a Bluetooth network—cannot be conducted. Besides, BlueHoc offers only a limited support for building a network. It also lacks flexibility in device control, in animated presentation, and in modeling mobility. The main contribution of the paper is therefore to enhance BlueHoc to support the aforementioned functions.*

1 Introduction

Wireless technologies have been rapidly developed recently. One of them is *Bluetooth*, which was initially developed to eliminate the need for inconvenient cable attachments. The specification of Bluetooth also supports *Ad Hoc Networks* (*AHN*s), a self-organizing wireless network formed by mobile nodes. No infrastructure is needed in AHNs, and they are usually deployed on demand. To promote the technology and to lead the evolution of Bluetooth, a number of companies have formed *Bluetooth SIG*. In addition, the *IEEE 802.15 Wireless Personal Area Network Group*, formed in 1999, has also adopted Bluetooth as the foundation of IEEE 802.15. So, undoubtedly, Bluetooth has become one of the most important technologies in wireless applications.

Wireless networks need to deal with some properties that are not found in wired networks, including, for example, mobility, channel transmission quality, and the interference and fading of signal. These properties make it impossible to port some technologies developed in wired networks to wireless environments. Even between wireless technologies, there are distinctive features that make them non-compatible.

For example, IEEE 802.11 assumes a single channel and a CSMA/CA style MAC protocol for its nodes. Nodes can communicate with one another via broadcast, and the topology of the data link layer is implicitly determined by radio range. Bluetooth, on the other hand, uses frequency hopping and time-division duplex scheme. Bluetooth nodes must synchronize in hopping sequence and clock before communication. This means that Bluetooth topology is not implicitly determined by radio range. Rather, a topology needs to be constructed before actual data can be delivered in the network [12, 11, 8, 7, 13, 4].

Because real networks are highly dynamic and complex, they are often hard to analyze mathematically. Simulation therefore plays an important role in network researches, especially for Bluetooth for which related researches are yet to be explored both in academia and in industry.

There are many simulation tools for network researches, aiming at different features and goals. One of the most popular network simulators in academia is *ns* (*Network Simulator*) [3, 9]. It provides a substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. It also has a Bluetooth extension, called *BlueHoc*. BlueHoc focuses on the performance of Bluetooth networks, including, for example, device discovery, connection establishment, QoS negotiation, and performance of TCP/IP.

Although BlueHoc has already offered many simulation functions, all simulations must be done in a virtually fixed topology. To see this, we first observe that a Bluetooth topology in the data link layer consists of *piconets*, where a piconet is a star-like topology consisting of one *master* node, and a number of *slave* nodes. All devices in a piconet share the same communication channel (or, more precisely, use the same hopping sequence that is determined by the master's device address). The master schedules traffic in the piconet. Direct communication is allowed only between the master and its slaves. Piconets can be connected together via *bridge* nodes to form a *scatternet*. A bridge node can be a slave in all piconets it connects, or a master in one piconet, and a slave in all other piconets. Prior to deploying any useful application to a network of Bluetooth devices, a (data link layer) topology must be constructed over the

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

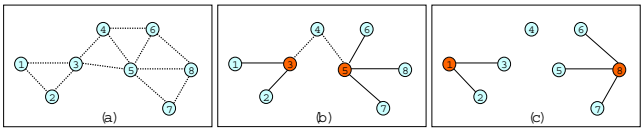


Figure 1. (a) A network of Bluetooth nodes, over which two topologies shown in (b) and (c) can be constructed.

network. For example, consider Figure 1. Part (a) shows the initial layout, where a dashed line between two nodes represent that the two nodes are within radio range of each other. Part (b) shows a possible topology over the nodes. The scatternet consists of two piconets, with masters 3 and 5, respectively. Node 4 acts as a bridge between the two piconets. Part (c) shows another topology, in which there are two disconnected piconets with masters 1 and 8, respectively. Node 4 is isolated as it is not within radio range of the two masters.

From the above example we can see that there are many possible ways (at least $O(2^n)$) to construct a topology over a set of n Bluetooth nodes. In particular, once the role (master/slave) of a node is determined, the topology is virtually determined (the rest is to decide to which piconet a slave node should belong, if there is more than one choice). Moreover, different topologies result in different performance. So the study of Bluetooth topology construction is the first and an important step in studying the performance of a network. Unfortunately, BlueHoc currently does not support this kind of studies, as the role of a Bluetooth node must be determined in advance.

Besides, BlueHoc is still a developing project. Some of its functions can be improved, including, for example, lack of flow control in the Bluetooth Baseband link controller for topology construction, lack of mobility, poor presentation of simulation results, and limited support for building scatternets, etc. The main contribution of this research therefore is to develop a more flexible and complete simulation environment for Bluetooth with the following features:

- Support dynamic topology construction.
- Enhance the behavior control of devices.
- Provide animated simulation results.
- Model mobility.
- Support scatternets.

The rest of the paper is organized as follows. Section 2 gives a brief description of Bluetooth and ns. Section 3 presents our simulator and its design issues. Section 4 uses our simulator to study a new topology construction algorithm. Conclusions and future work are offered in Section 5.

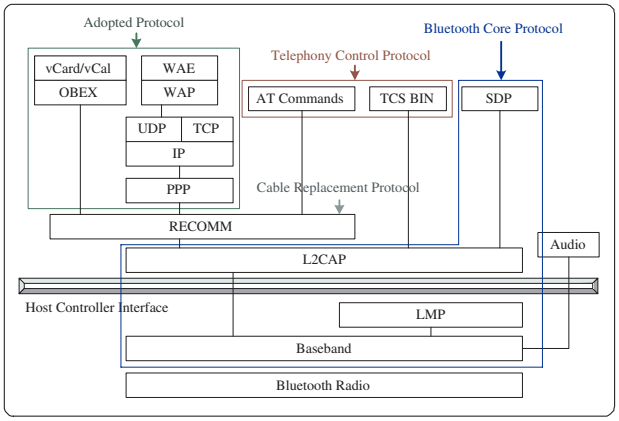


Figure 2. Bluetooth protocol stack.

2 Bluetooth and *ns*

Figure 2 shows Bluetooth protocol stack [1, 10]. Of the protocols, the Baseband is the most comprehensive and important part of the Bluetooth specification. It is the physical layer of Bluetooth which manages physical channels and links, and governs other services such as error correction, data whitening, channel control, hop selection, and Bluetooth security. The Baseband also provides asynchronous and synchronous links, handles packets, and does paging and inquiry to access and inquire other devices in the area.

One thing to note in the Baseband is that two Bluetooth devices establish a link through a sequence of *inquiry* and *paging* procedures. The inquiry procedure enables a device to discover which devices are in range, and what their device addresses and clocks are. The page procedure then establishes the actual connection. The state diagram of the Bluetooth link controller is shown in Figure 3. There are two main states: STANDBY, in which the device is in a low-power mode not connecting to any other device; and CONNECTION, in which the device has established a link with another device. The other seven substates are used during the link establishment procedure. For details of the state transition, please refer to [1].

Ns is an object-oriented, discrete event driven network simulator written in C++ and OTcl. It adopts a *split-programming model* to balance between performance and flexibility [2]: In order to reduce event and packet processing time, ns uses compiled C++ to implement the event scheduler and the basic network component objects. It also makes these C++ objects available to OTcl and gives it the control of them by creating matching OTcl objects. In this way, tasks such as dynamic configuration of protocol objects and the specification and placement of traffic sources become more flexible.

Ns currently has BlueHoc [5] as an extension for Bluetooth. BlueHoc allows users to evaluate how Bluetooth performs under various ad-hoc network scenarios. In practice, BlueHoc implements basic features of Bluetooth radio, Baseband, LMP, and L2CAP. The key issues addressed in-

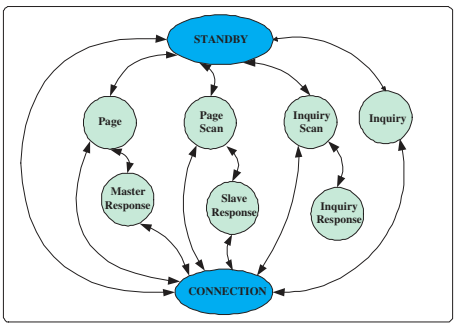


Figure 3. State diagram of Bluetooth link controller.

clude: device discovery performance of Bluetooth, connection establishment, QoS negotiation, medium access control scheduling schemes, radio characteristics of Bluetooth system, statistical modeling of the indoor wireless channel, and performance of TCP/IP based applications over Bluetooth.

Figure 4 shows the architecture of a Bluetooth node in BlueHoc. The implemented components include: Baseband, DRR-based Scheduler, LinkController, LBF, LMP, L2CAP, BTHost, and BTClassifier. By using OTcl, these components are put together as a Bluetooth node named *BTNode*. One thing to note about BTNode is that a BTNode has a different structure according to its role. Therefore, a BTNode that acts as a master or slave cannot switch its role once simulation starts. In other words, users must determine the role of each node before simulation.

As for the simulation procedure, it starts with the creation of a number of master nodes and corresponding slave nodes. Each master node then begins an inquiry procedure to obtain addresses and clocks of slave nodes in range. Once the inquiry procedure is over, a master uses a paging procedure to make the slaves it has inquired tune to the master's hopping sequence. LMP then establishes ACL links between the master and its slaves. The master next carries out QoS setup, which depends on the application for which the connection is required. After the QoS negotiation, L2CAP signaling takes place between L2CAP peers. Finally, higher layer packets are transferred through service primitives provided by L2CAP.

3 System Design and Implementation

As noted in the previous section, the architecture of a BTNode is different according to its role. Although using OTcl can dynamically create and compose objects, a BTNode cannot act both as a master and as a slave simultaneously. That is, users need to explicitly define the role of each node before simulation, and the role cannot be changed during the simulation. Therefore, before Bluetooth devices in BlueHoc can communicate to each other, all of them have a permanent role and architecture. This makes BlueHoc unable to support dynamic network topologies.

The lack of flow control in the Baseband link controller is another problem in BlueHoc. At the OTcl level, the parameters that users can configure include: the number of devices, coordinates of devices, DIACs (Dedicated Inquiry Access Codes) of devices, higher layer protocols of devices, the number of piconets, and the simulation time, the timeout and termination condition of the inquiry procedure, etc. These parameters have little to do with the control of the state transition in the link controller (see Figure 3). So if one wishes to conduct various state transitions in the link controller (that arises from the study of link/topology constructions), he has to change the code at the C++ level. According to the BlueHoc manual [5], only the BTHost implementation (`bt-host.cc`) needs to be changed when users wish to implement a different flow control. However, without an appropriate control interface, it is difficult and even impossible for users to control the behavior of devices. For example, in the current implementation, once the lower level procedures of inquiry, paging and connection establishment are completed, what users can do is only to start traffic sources and begin data transmission. Hence, scenarios such as disconnection and reconnection cannot be supported by BlueHoc.

Some other features in BlueHoc can also be improved as well. These include, first, the simulation result of BlueScat 0.6 is simply some text messages on the screen; no detailed trace information is provided (BlueHoc 2.0 has integrated support for traces, but it can simulate only one single piconet). Second, mobility is yet to be considered in BlueHoc. Third, by examining the source code in BlueHoc we found that the number of piconets that can be simulated is limited to seven.

Based on the above discussions we have set up the following goals to improve BlueHoc:

- Enable dynamic topology construction: allow each device to change its role dynamically in simulation.
- Enhance the behavior control of devices: provide some standard control interfaces that can be used to easily implement a topology construction algorithm.
- Integrate simulation results: generate trace files that can be integrated into NAM.
- Model mobility: implement a mobile model to handle device movement, and allow simulation scenarios to take mobility into account.
- Support scatternets: remove the restriction that the number of piconets is up to seven.

Figure 5 shows the new architecture of BTNode in our simulator. As can be seen, the new architecture is based on the two existing architectures in BlueScat 0.6 (see Figure 4). In order to support role switch between master and slave, we combine the two isolated architectures to form this compound one. Note that we do not use a uniform architecture because the tasks of the two roles are quite different. For a master, it plays a more active role and has to manage the one-to-many relationship between the device and

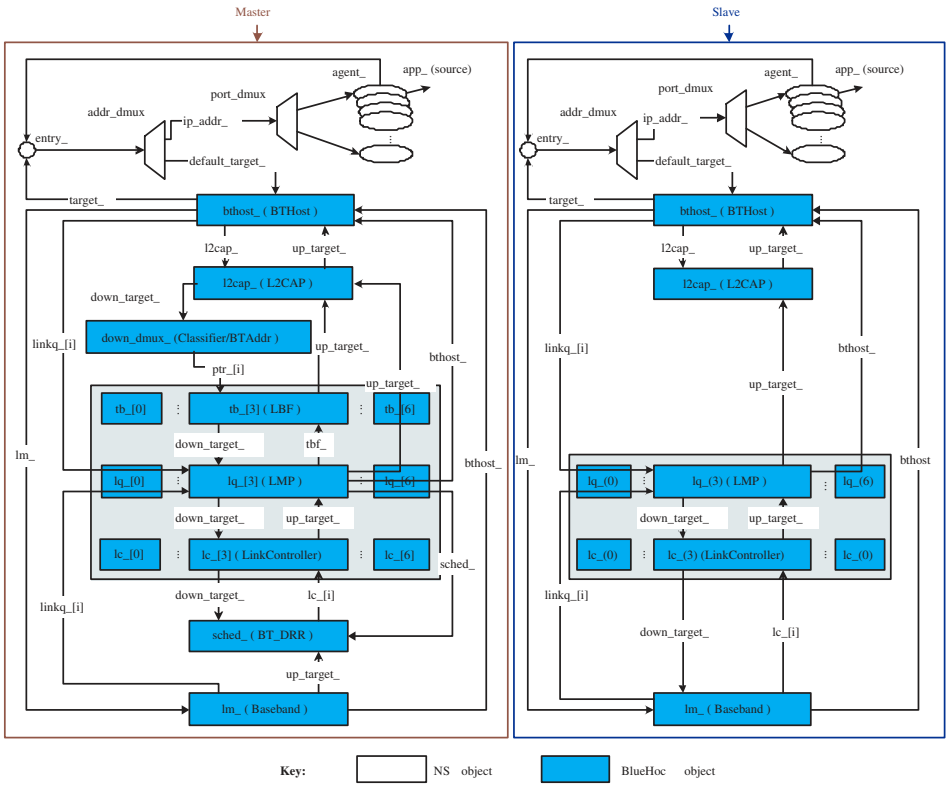


Figure 4. Architecture of a Bluetooth node (BTNode).

its slaves. Thus it needs components to handle some extra tasks, such as scheduling (BT_DRR), admission control (LBF), and member identification (Classifier/BTAddr), etc. On the other hand, a slave plays a more passive role, and only needs to handle the one-to-one relationship between the device and its master.

To enhance the behavior control of devices, we first observe that the function of BTHost in BlueHoc is responsible for controlling the behavior of a device. This design mixes the code of topology construction algorithms user wrote with the protocol code of the system itself. Thus, in our new design, we change BTHost's role to simply as a supporter that provides standard control interfaces to a new component called *BTAlgo*. The new component now is responsible for algorithm implementation. We also define a communication protocol between BTAlgo and BTHost, and enable each BTAlgo component to access standard control interfaces provided by BTHost. Finally, by using OTcl, all BTAlgo components developed for various algorithms can be substituted dynamically. So users can replace BTAlgo components without modifying any source code in the device. As such, we have separated the implementation of algorithms from the implementation of Bluetooth protocols, and provide a more convenient and flexible way for users to simulate their algorithms.

As for the last three design goals described in the previous section, all of them can be accomplished by enhancing the function of the components in the existing simulator. We will address this later in this section.

Figure 6 shows the lower software layer in Bluetooth specification. By comparing this figure with Figure 5, we can see that the architecture in Figure 6 is almost fully implemented in our system (like BlueHoc, the physical bus is replaced by function calls). Furthermore, we can also see that our new architecture that incorporates both BTAlgo and BTHost is more conformable to the design of the Bluetooth Host in Figure 6.

Below we discuss some of the detailed implementation.

Enable dynamic topology construction

As discussed before, to accomplish this goal we combined the existing master and slave modules to form a new compound one. Some of the existing components have to be modified as well. Among them, Baseband is the most important one as it needs to handle almost all new conditions arise from the role-switch capability of a device. For example, to support master/slave bridges, each device needs to maintain information about the piconets it has participated in. According to Bluetooth specification, before a device in CONNECTION state can enter another state, it needs to properly handle its current connection. That is, the device should hold, sniff, park, or disconnect before it performs piconet switch or enters a non-connection state. For this, we offer two modes in the CONNECTION state: **Active**,

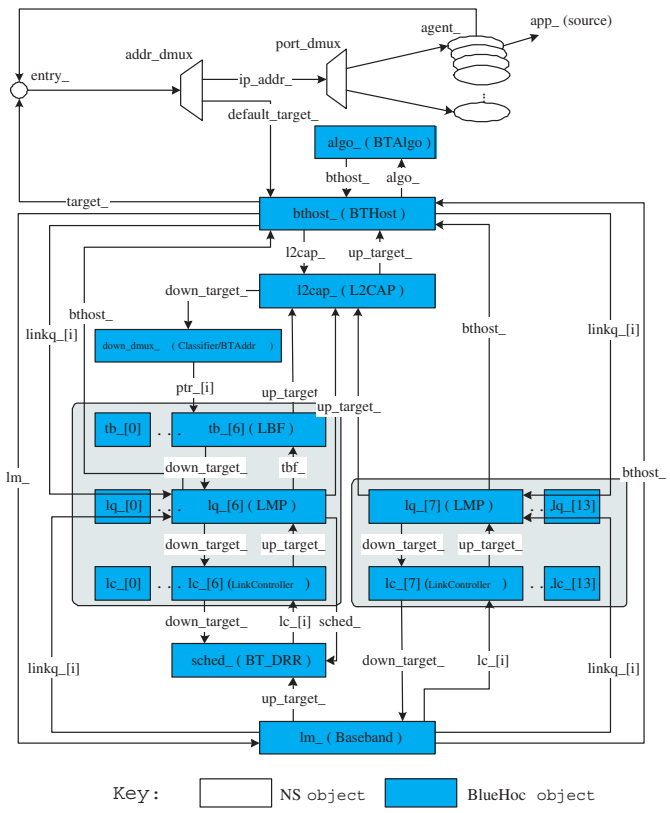


Figure 5. New Architecture of a Bluetooth node (BTNode).

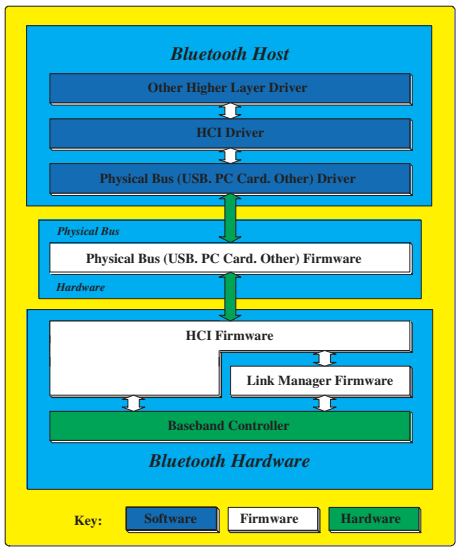


Figure 6. Overview of the lower software layer.

in which the device actively participates on the channel it connects to, and **Hold**, in which the device can switch to another piconet or do scanning, paging, and inquiring. [1] We also implement a disconnect command that allows a device to entirely leave a piconet.

In fact, BlueScat 0.6 also uses hold mode to support the piconet-switch procedure. However, the device that can perform this procedure is restricted to slave only. Some new problems need to be addressed if we will allow a master to perform the procedure. For example, because a master can have up to seven active slaves simultaneously in a piconet, it needs to maintain the information of its slaves, and hold them before switching to another piconet. That is, each master, when wishing to perform the piconet-switch procedure, has to monitor the state of its connections, and uses the PDUs in LMP to negotiate with its slaves to decide when they should enter and leave the hold mode. Therefore, both the Baseband and LMP components need to be modified as well.

In addition, some design issues of BlueHoc are based on the assumption that knowledge about master-slave pairs are available before simulation. So the source-destination pairs of traffic sources can simply map to the master-slave pair given in the Tcl script. This has to be modified as now each traffic source needs to dynamically determine its destination. So we need to extend the function of the BTHost component to support this. Finally, components such as LinkController and L2CAP also need to handle the changes of the new architecture.

Enhance Behavior Control

We added some HCIs (Host Controller Interfaces) for the BTHost and BTAlgo components to work together to facilitate the implementation of various algorithms. As shown in Table 1, most procedures, such as inquiry, page, inquiry scan, page scan, hold, and disconnect, etc., can be invoked by using these HCIs. In addition, there are HCIs for event notifications, parameter setting, information acquirement, and connection support. Other HCIs can be added in the future if needed. As for the control of L2CAP, we follow the design of BlueHoc, and let the BTHost component provide six service primitives (L2CA_ConnectReq, L2CA_ConnectRsp, L2CA_ConnectInd, L2CA_ConnectCfm, L2CA_DataRead, and L2CA_DataWrite). The first four are for the creation of a channel, and the last two are for data transmission after the channel is established. Finally, we also provide a basic control of the traffic sources, e.g., start and stop to generate traffic.

Integrating Simulation Results

We modified the BTNodeTrace component to generate ns traces and NAM traces. We define four classes of event,

[1] Bluetooth specification also defines other two modes—sniff and park—in the CONNECTION state. Although for our purpose they are not necessary, they can be implemented in the future.

HCI Command – Link Control Commands	
HCI_Inquiry	HCI_Create_Connection
HCI_Disconnect	HCI_Accetp_Connection_Request
HCI_Read_Clock_Offset	
HCI Command – Link Policy Commands	
HCI_Hold_Mode	HCI_QoS_Setup
HCI Command – Host Controller & Baseband Commands	
HCI_Write_Scan_Enable	HCI_Write_Inquiry_Scan_Activity
HCI_Write_Page_Scan_Activity	HCI_Write_Page_Timeout
HCI Command – Informational Parameters	
HCI_Read_BD_ADDR	
HCI Events	
HCI_Inquiry_Result_Evnet	HCI_Connection_Complete_Event
HCI_QoS_Setup_Complete_Event	HCI_Connection_Request_Event
HCI_Disconnection_Complete_Event	HCI_Mode_Change_Event
HCI_Read_Clock_Offset_Complete_Event	

Table 1. HCIs supported in our simulator

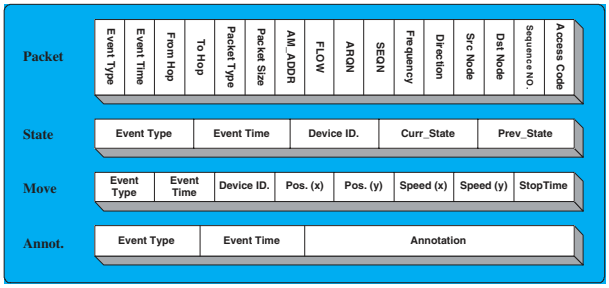


Figure 7. File format of four events.

packet, *state*, *movement*, and *annotation*, which will be recorded during simulation. Among them, packet events trace the transfer, receipt, and drop of each packet. State events trace the states, such as standby, inquiry, and connection etc., of the Baseband of each device. As for the last two classes of events, movement events are designed for mobile devices, and annotation events provide a flexible way to let users record the messages of their algorithms. Moreover, it is easy for users to extend the functions of BTNodeTrace, and to add new information to the traces.

Figure 7 shows the file format of the four classes of events in the ns traces. The file format of the NAM traces needs to follow the format defined by NAM [3] so that our simulation results can be understood. Finally, information such as location and communication distance of each device has been added to our simulator to allow users to 'view' each device in the animation.

Model Mobility

To model mobility we first need to define a command that can describe various movement paths. In fact, ns already has this kind of command for the movement of its MobileNode [9], as shown below:

Syntax: \$ns at \$time "\$BTNode setdest ⟨x2⟩ ⟨y2⟩ ⟨speed⟩"
Example:\$ns at 0.5 "\$Sim(dev:0) setdest 10 3 0.65"

The command contains the starting time, affected device, destination, and velocity of this movement. Although such a command can only describe a straight-line movement, all movement paths can be approximated by the straight lines. Furthermore, this command is familiar to ns users, and may gain support to generate movement files from ns. Hence, we adopt it in our simulator.

To describe the continuous movement of devices in our discrete-event environment, we calculate the coordinate of a device using the following continuous function:

$$\begin{cases} x_t = x_s + \frac{v \times (x_d - x_s)}{\sqrt{(x_d - x_s)^2 + (y_d - y_s)^2}} \times (t - t_s) \\ y_t = y_s + \frac{v \times (y_d - y_s)}{\sqrt{(x_d - x_s)^2 + (y_d - y_s)^2}} \times (t - t_s) \end{cases}$$

where

x_s: initial location of the device.
x_t: location of the device at time t.
x_d: destination of the device.
v: velocity of the device.
t_s: initial time of the device.
t: the time at which the device's location is observed.

To cope with the problem that packet transmission delay is not an instant but rather an interval, we first observe that Bluetooth packet size is relatively small: the largest packet is 2871 bits. So the transmission delay of such a packet is only 0.002871 seconds when the bandwidth is 1 Mbps. So suppose a mobile device 1 is sending a packet to another mobile device 2. Suppose further that we use the initial distance $d1$ to estimate the distance between the two mobile devices ($d1$, in turn, is used to determine the frame error rate of this packet delivery). Then, even if the two mobile devices are moving at a relative speed of 17.415m/s, the resulting distance error is only 5cm, which is well within the accuracy of [6] (the distance in the frame error rate table in [6] is on a 1m-scale). For Bluetooth devices whose operating distance is in 10-100m, mobility up to 17.415m/s should satisfy most circumstances.

Support Scatternets

While studying the source code of BlueScat 0.6, we noticed that BlueHoc can simulate at most seven piconets at a time. That is because each master in BlueHoc is assigned a global sequence number, which is used to index an array that stores information of its piconet. It is easy to see that the number of piconets, which the system can simulate, is limited by the size of the array. A simple way to increase the number of piconets is to enlarge the size of the array. Unfortunately, devices in BlueHoc all maintain their own arrays. Thus relaxing scatternet size by enlarging the size of the array will cause the consumption of memory to grow at an n^2 scale, where n is the number of devices in simulation.

We instead let each device maintain a mapping table that associates each of its masters with a local sequence number, and use these local sequence numbers to index its own information array. It is easy to see that with the same memory space, the new method still has a limitation that a device can participate in up to seven piconets at a time. However, it does not restrict the total number of piconets in simulation. Moreover, if we use a link list to substitute the fixed array, the above limitation can be efficiently eliminated.

4 Example

In this section we use our simulator to evaluate a new Bluetooth topology construction algorithm proposed by Huang and Joung [4]. Before presenting the algorithm, we first note that a distinguishing characteristic of Bluetooth link formation procedure is that two devices must be in a pair of complementary states—*inquire* and *inquire scan*—in order for them to discover each other and establish a link. To avoid the role of a device being predetermined statically (which would otherwise cause a topology construction algorithm to lose its generality), a plausible way is to let each device randomly determine its state before it starts neighbor discovering [11], or randomly alternate between the inquire and inquire scan states [12]. These two schemes generally require the underlying physical topology to be completely connected.

The new proposed algorithm improves upon previous results in that the underlying physical topology needs not be a complete graph. Rather, if the physical topology is weakly connected, then the algorithm guarantees to construct a connected scatternet topology with high probability. The key idea is to let each device alternate between inquiry and inquiry scan in some pattern, called *state alternation sequence*, such that two devices have a chance to discover each other if they have different state alternation sequences, and will not discover each other otherwise (because they are always either both in inquiry state, or both in inquiry scan). The algorithm proceeds in two phases:

1. Max ID Propagation:

   During this phase, each device has a variable called ROOT, which records the maximum id it has learned. ROOT is initialized to the device's id. The variable is also used to determine the device's state alternation sequence. When two devices discover each other, they compare their ROOT values. The loser becomes a 'child' of the winner. It discards all its children it has won, and sets its ROOT value to the winner's id. The winner's id will be used for subsequent competitions with other devices the loser has discovered. It is also used to determine the loser's new state alternation sequence so that the loser can synchronize its state alternation sequence with the winner's. This ensures that the two devices will not rediscover each other, while still allowing both of them to continue their discovery process, so as to avoid partitions. When the maximum id in the system is propagated to all other devices, all devices will use the same state alternation sequence, and so will not be able to discover any more device. They will then eventually time out and enter the following phase.

2. Connection Establishment:

   In this phase, each device first pages all its children to make connection to them, and then waits for connection from its parent. As a result, a scatternet is constructed in a tree structure from bottom up.

To implement the algorithm in our simulator is to develop the corresponding BTAlgo components for it. When a device starts, we can carry out the state alternation scheme by calling the HCIs, HCI_Inquiry and HCI_Write_Scan_Enable. The Baseband component of the device will then enter inquiry and inquiry scan alternately. Eventually, the Baseband component will report the result of its operation to the BTAlgo component through proper events (INQ_FINISH, PAGE_FINISH, or PAGE_SCAN_FINISH). Then we can ask the device to exchange information when connection is established, and continue the state alternation scheme afterward.

After developing the BTAlgo components, we need to write a Tcl script to configure the simulation environment and actually simulate various scenarios. A Tcl script for the simulation is shown in Figure 8. From the script we can see that there are fifteen devices in the simulation environment. The devices are within a 10m square space. The BTAlgo component of each device is assigned as GDH, and the total simulation time is set to 12 seconds, etc.

The snapshots of the animated simulation are shown in Figure 9. Snapshot (a) shows the initial layout. In (b), the devices start to discover each other. The messages at the bottom show that device 13 with $\mathrm{ROOT} = 13$ has beaten device 9 with $\mathrm{ROOT} = 9$. Similarly, device 11 with $\mathrm{ROOT} = 11$ has beaten device 5 with $\mathrm{ROOT} = 10$ (so device 5 has been beaten before by a device with $\mathrm{ROOT} = 10$). The losers then set their new parents, and discard the children they have captured before. In (c), all devices have set their R OOTto 14, and so they have synchronized their state alternation sequences so that they are always all in inquiry state (e.g., (c)), or all in inquiry scan (e.g., (d)). As a result, none of them will be able to find a device in their neighbor discovery process, and so will all eventually time out and enter Phase 2 to connect to their children and parents they have found in Phase 1. For example, in (e), devices 4, 5, and 8 have timed out and have started to page their children. Moreover, devices 4 and 5 do not have any child and so they enter page scan to wait for connection from their parents. In (f), devices 6, 8, 5, and 4 have formed two piconets and are waiting for connection requests from device 14. Meanwhile, devices 0, 2, 12 are still waiting for their children to finish connecting to their children. The message "Device_13 : Master – $(N, C, S, S, R, R) = (5, 1, 6, 90, 6, 90)$", at the bottom indicates that device 13 is not a pure slave; it totally finds five neighbors during the construction, and takes one of them

```
set Sim(Trace) ON
set Sim(NumDevices) 15
set Sim(SmartMode) ON
set Sim(xmax_) 10
set Sim(ymax_) 10
set Sim(xpos_) [list 3 0 3]
set Sim(ypos_) [list 8 5 2]
set Sim(Algorithm) [list GDH]
set Sim(Transport) [list UDP]
set Sim(Application) [list Traffic/Exponential]
set diac [list 2 2 2 2]
set StartTime [list 0.001 0.005 0.015 0.011]
set SimulationTime 12.0
set Sim(NamFile) "bt-gdh.nam"
source ./proc.tcl
source ./run.tcl
```

Figure 8. A Tcl script.

as its child. The last four variables are the number of sent packets, the amount of sent data, the number of received packets, and the amount of received data, respectively. Note that the position of devices has been rearranged in such a way that slaves in a piconet are placed near their master. In (g), device 14 makes connection to its child 10, and the final topology is then established. In (h), the devices start to use the topology they constructed to communicate.

In addition to NAM traces, users can also extract some algorithm-specific information from the ns traces. This is because the predefined events, packet, state, movement, and annotation, have been recorded during the simulation. Users can process and analyze them according to the file formats discussed in Section 3. For example, the packet events in the ns traces can tell users the number of sent/received/dropped packets of each device in each phase, and hence users can evaluate the performance of each phase and each device.

5 Conclusions and Future Work

We have presented an extended package for BlueHoc. The package allows a node to dynamically switch its role between master and slave, thereby allowing users to simulate dynamic topology construction of Bluetooth devices—the first and an important step in establishing a Bluetooth scatternet. We have also removed the restriction of the number of piconets (seven) in a scatternet imposed by BlueHoc, and so a more general type of scatternets can be supported in our package. By offering a better device control interfaces, users can easily implement their topology construction algorithms to simulate various scenarios that are not possible in BlueHoc. The new package has also considered mobility, and has provided an animated presentation of simulation results. The source code and tutorials are available at `http://joung.im.ntu.edu.tw/bluehoc_ex/`.

Still, there are work to be done in the future. For example, many HCIs defined in Bluetooth specification are yet to be supported in our system. Moreover, some power management schemes may be considered, and a mobility scenario editor may also be added. Finally, in our current implementation packet delivery rate does not reflect packets interference, and does not take velocity and moving direction into account.

References

[1] Bluetooth. Specification of the bluetooth system. Http://www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf.

[2] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[3] Ahmed Helmy and Satish Kumar. VINT—Virtual InterNetwork Testbed. Http://www.isi.edu/nsnam/vint/index.html.

[4] Geng-Dian Hwang and Yuh-Jzer Joung. A symmetric and distributed algorithm for bluetooth topology construction. Technical report, Department of Information Management, National Taiwan University, 2002. Submitted for publication.

[5] Apurva Kumar. BlueHoc Manual. Http://oss.software.ibm.com/bluehoc/tutorial/docpage.html. IBM India Research Lab.

[6] Apurva Kumar and Rajeev Gupta. Capacity evaluation of frequency hopping based ad-hoc systems. In *ACM SIGMETRICS International conference on Measurement and modeling of computer systems*, pages 133–142, 2001.

[7] Ching Law, Amar K. Mehta, and Kai-Yeung Siu. Performance of a new bluetooth scatternet formation protocol. In *Proceedings of the 2001 ACM International Symposium on Mobile ad hoc networking and computing*, pages 183–192, 2001.

[8] Ching Law and Kai-Yeung Siu. A bluetooth scatternet formation algorithm. In *Proceedings of the IEEE Symposium on Ad Hoc Wireless Networks*, pages 2864–2869, 2001.

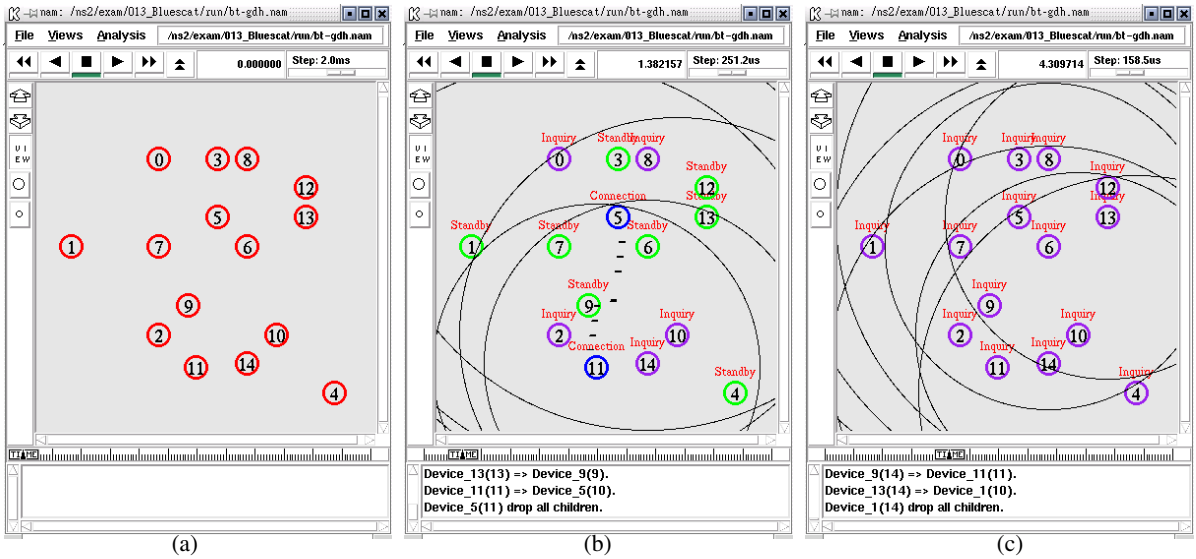
[9] NS. The network simulator: ns. Http://www.isi.edu/nsnam/ns/.

[10] Palo Wireless Bluetooth Resource Center. Bluetooth Tutorial–Specification. Http://www.palowireless.com/infotooth/tutorial.asp.

[11] Lakshmi Ramachandran, Manika Kapoor, Abhinanda Sarkar, and Alok Aggarwal. Clustering algorithms for wireless ad hoc networks. In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 54–63, 2000.

[12] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire. Distributed topology construction of bluetooth personal area networks. In *Proceedings of the INFOCOM-01*, pages 1577–1586, 2001.

[13] Godfrey Tan, Allen Miu, John Guttag, and Hari Balakrishnan. Forming scatternets from bluetooth personal area networks. Technical Report MIT-LCS-TR-826, MIT, Oct 2001.



(a) (b) (c)