Blueware: Bluetooth Simulator for *ns*

Godfrey Tan
MIT Laboratory for Computer Science
Cambridge, MA 02139
{godfreyt}@lcs.mit.edu

October 29, 2002

1 Introduction

Bluetooth [2] is emerging as an important standard for short range, low-power wireless communication [3, 5]. Bluetooth operates in the 2.4 GHz frequency band and a Bluetooth link has a maximum capacity of 1Mbps. Its link-layer Baseband protocol is designed to facilitate the construction of *ad hoc* networks without manual configuration or wired infrastructure.

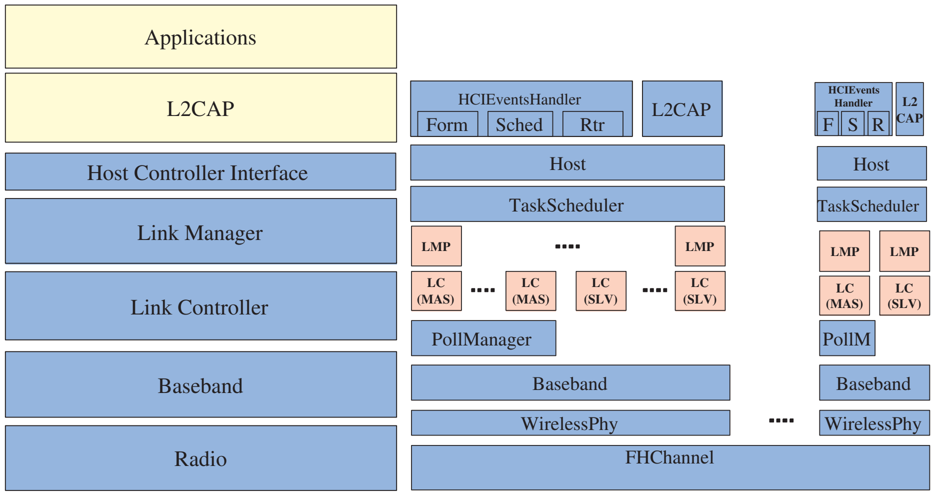
Bluetooth communication is based not on distributed contention resolution, as in traditional wireless LANs, but on a *time-division duplex (TDD)* master-slave mechanism. A Bluetooth *piconet* consists of one master and up to seven slaves. The master allocates transmission time slots, each of which takes 625 microseconds, to the slaves in the piconet. The master and slaves use alternate transmission slots, with each odd slot being used only by the slave to which the master sent a frame in the previous even transmission slot.

Frequency hopping is used to permit multiple concurrent Bluetooth communications within radio range of each other, without adverse effects due to interference. This facilitates high densities of communicating devices, making it possible for dozens of piconets to co-exist and independently communicate in close proximity without significant performance degradation. In principle, this raises the possibility of internetworking multiple piconets called *scatternets*. However, the Bluetooth Specification does not specify how this is to be done Providing support for self-organizing scatternets has been an active area of research [10, 7, 11, 9, 6, 12, 13]. In particular, there are three main challenges: i) topology formation, ii) link scheduling and iii) packet routing. The TDD nature of Bluetooth combined with its neighbor discovery protocol makes the first two problems new and interesting.

To evaluate performance of Bluetooth piconets and scatternets, we have developed an extensible Bluetooth simulator, called *Blueware*, as an extension to *ns* [8]. Our simulator implements many aspects of the Bluetooth protocol stack according to the Bluetooth specification (version 1.1). Our development efforts were eased by the *Bluehoc* [1] simulator implementation released by IBM in 2001. Bluehoc simulator works well in evaluating performance within a single piconet. However, Bluehoc lacks support for scatternet protocols such as topology formation and link scheduling schemes. Blueware addresses these issues and provides an extensible architecture for various scatternet formation and link scheduling schemes. Using this architecture, we have developed both scatternet formation [14] and link scheduling [13] schemes and evaluated their performance. The Blueware simulator module is available for public use under the IBM Public License [4]. The rest of this paper describes in detail the Blueware simulator and its extensible architecture support for scatternet protocols.

2 Blueware Simulator

The Blueware simulator implements most aspects of the Bluetooth protocol stack according to the Bluetooth specification (version 1.1). Figure 2 shows the Bluetooth protocol stack and Figure 1 depicts various modules of the Blueware simulator. Each device in the simulator is equipped with several modules through which it discovers and communicates with other devices. The WirelessPhy module provides functionality of the Bluetooth radio. The FhChannel module simulates the frequency hopped wireless medium. The Baseband module of the simulator implements the pseudo-random frequency-hopping technique, and the Inquiry, Inquiry Scan, Page, Hold, and Role-Change operations as specified in the Bluetooth Baseband specification. PollManager manages master-slave links within a piconet and decides which active slave to poll. The LMP module implements detailed LMP protocols and the LC module implements link control opera-



(a) Bluetooth Stack (b) Blueware Simulator

Figure 1: Comparison between Bluetooth stack and Blueware modules

tions such as the Automatic Repeat Request (ARQ) scheme. As shown in the figure, every master-slave link has unique instances of the LC and LMP modules which are responsible for carrying out all the Bluetooth primitive link establishment, control and communication operations. TaskScheduler implements the scheduling schemes for various Bluetooth operations related to communication and neighbor discovery. The Host module provides the Host Controller Interface (HCI) and interacts with various scatternet protocols. Each scatternet formation or link scheduling scheme can be implemented by extending the HCIEventsHandler module as we discuss later in this section. Finally, the L2CAP module provides segmentation and re-assembly of application data units. The rest of this section describes in detail each simulator module.

2.1 Baseband

Bluetooth Baseband implements two main functions: i) to discover neighboring nodes using Inquiry and Page operations and ii) to generate pseudo random frequency hopping sequence. In this section, we explain the details of both functions and how they are implemented in the Blueware Baseband module.

The link formation process specified in the Bluetooth Baseband specification consists of two processes: Inquiry and Page [2]. The goal of the Inquiry process is for a master node to discover the existence of neighboring devices and to collect enough information about the low-level state of those neighbors (primarily related to their native clocks) to allow it to establish a frequency hopping connection with a subset of those neighbors. The goal of the Page process is to use the information gathered during the Inquiry process to establish a bi-directional frequency hopping communication channel. Figure 2 illustrates the Bluetooth link formation process.

During the Inquiry process, a device enters either the Inquiry or the Inquiry Scan state (mode). A device in the Inquiry state repeatedly alternates between transmitting short ID packets containing an Inquiry Access Code (IAC) and listening for responses. There are a total of 64 reserved IACs but only two of them are officially defined: Generic Inquiry Access Code (GIAC) and Limited Inquiry Access Code (LIAC). A device in the Inquiry Scan state constantly listens for packets from devices in the Inquiry state and responds when appropriate. It is important to note that once a node is in the Inquiry state, it remains in that state for several seconds. A node periodically (every $1.28s$ or so) enters the Inquiry Scan state to scan continuously over a short window of $11.25ms$, and thus, can communicate with other nodes or sleep in between consecutive scans. Thus, the duration that a node stays in Inquiry or Inquiry Scan mode is asymmetric.

Multiple Inquiry Scan nodes can simultaneously receive messages from the same Inquiry node. To avoid contention, each scanning node chooses a random back-off interval, T_{rb}, between 0 and 1023 time slots before responding with the signaling information. Let T_{sync} be the delay before two nodes can synchronize their frequencies during the Inquiry process. The time taken to complete the Inquiry process is given by

$$T_{inq} = 2T_{sync} + T_{rb} \tag{1}$$

T_{sync} varies according to the differences in clock values between nodes. If the two nodes have synchronized clocks, T_{sync} will be very short and thus, T_{rb} dominates T_{inq}. However, if they are not synchronized, T_{sync} will dominate.

A node remains in the Inquiry state and keeps track of which nodes respond during this time. The Inquiry operation is terminated when the number of responses has reached the maximum value, $num\_responses$, or the Inquiry timer, $inqTm$, has expired. The Baseband modules will then generate an event to report the results of the Inquiry operation.

When an upper layer receives this event, it can decide to enter the Page state to create connection with one of those devices which responded. Analogously, a node in the Inquiry Scan state periodically enters the Page Scan state. A device in the Page state uses the signaling information obtained during the Inquiry state and sends out trains of ID packets based on the discovered device's address, BD_ADDR[1]. When the device in the Page Scan state responds back, both devices proceed to exchange necessary information to establish the master-slave connection and eventually enter the Connection state. The device in the Page state becomes the master and the device in the Page Scan state the slave. Figures 3 and 4 illustrate the state transitions during the Inquiry and Page processes respectively.

The Page process is similar to the Inquiry process except that the paging device already knows the estimated clock value and BD_ADDR of the paged device. However, there will still be some synchronization delay before the pager and the paged devices can communicate. We define T_{pg} as the time taken to complete the Page process. It will be most efficient for the two nodes in the Inquiry process to enter the Page process as soon as the inquiring node has received the inquiry response. Thus, the total time taken to establish a link between two nodes is

$$T_{conn} = T_{inq} + T_{pg} \tag{2}$$

T_{inq} is typically much larger than T_{pg} and dominates the delay to enter the Connection state[2].

[1] BD_ADDR is the globally unique 48-bit address of the Bluetooth device.

[2] T_{inq} is in the order of seconds whereas T_{pg} is in the order of milliseconds if both nodes in the Inquiry process enter the Page process immediately after the inquiry response is received.

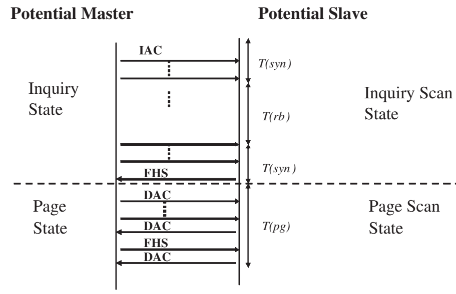


Figure 2: Bluetooth link formation process

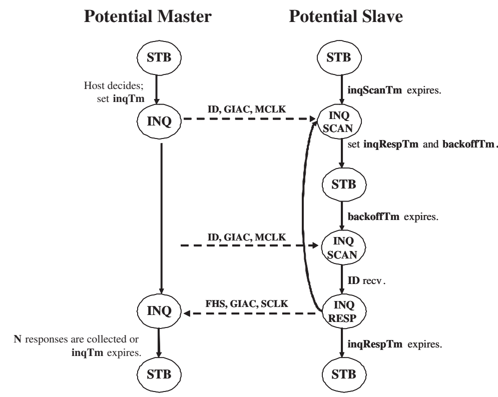


Figure 3: State transitions during the Inquiry process

The Baseband module completely implements all aspects of Inquiry and Page operations.

The Baseband module also implements the frequency hopping sequence generator. At each clock tick (312.5μs), each node computes the frequency for transmitting or receiving packets based on a Bluetooth device address, clock values, and the Baseband state. Although all nodes in Inquiry and Inquiry Scan states hop over a well-known sequence[3], their phases may differ since the phase is decided by the native clock value of the node. Computing the a radio frequency for each clock tick, in fact, consumes significant amount of CPU cycles. Thus, we added a cache to store the resulting frequency based on the parameters, such as the clock bits and the Baseband state, that are used to compute the channel frequency. However, the frequency hopping operation can still hamper the performance of the simulator significantly for the reasons we explain in the next subsection.

[3] GIAC is used an address input to generate the well-known sequence.

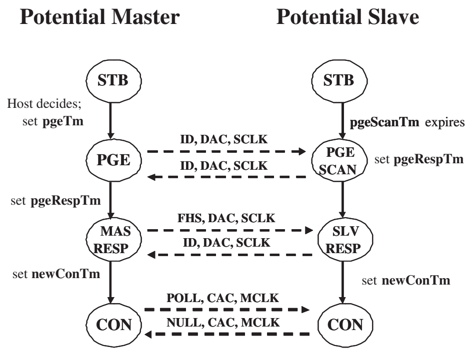


Figure 4: State transitions during the Page process

2.2 Frequency Hopped Channel

Bluetooth operates in the 2.4GHz band and separates the medium into 79 radio frequency channels. Since each node either transmits or listens on a single channel at a time, there could be as many as 79 parallel communications within a single radio range. Frequency hopping technique coupled with the TDD scheme presents an interesting issue in modeling the wireless medium for simulation purposes.

A simple approach is to model a single wireless channel module that delivers a copy of every transmitted packet to each node[4]. The node then computes its receiving frequency to decide whether it should receive the packet. The problem with this approach is that the processing cost of each packet grows linear with the number of nodes within the transmission range. Assuming half of the nodes transmit in every time slot, the total processing cost of all the transmitted packets grow quadratic in the number of nodes.

To avoid this, we model the wireless medium for the frequency hopped networks like Bluetooth as a single FhChannel module consisting of m channels. At every listening time-slot, each device registers its physical interface (WirelessPhy) with the FhChannel module so that it only receives the packets transmitted on that frequency. When a packet is transmitted on the FHChannel, it delivers a copy of the packet to each node that has registered to the channel corresponding to the transmitted frequency. Compared to the previous approach, this approach effectively reduces the processing cost of each packet by a factor of m, where $m = 79$ for Bluetooth. Although the worst case time complexity of the packet processing cost remains the same, when the number of nodes is less than or equal to 79, the average packet processing cost grows only linearly in the number of nodes.

[4] This is the approach taken by [1].

2.3 Wireless Physical Interface

The WirelessPhy module implements the Bluetooth radio. As of Blueware 1.0, this module simply forwards the packets from(to) Baseband to(from) FHChannel. In the future, this module should implement a reasonable radio propagation model so that proper energy and interference models can be developed.

2.4 Link Controller

Each point to point (master-slave) link is managed by the LinkController module. LinkController implements Automatic Repeat Request (ARQ) to provide the reliability on a single link. We distinguish between two types of links: master links and slave links. A link is considered a master(slave) link by an end node if the node is acting as a master(slave) on that link. For each slave link, the LinkController module also maintains the active member address, *am_addr*, assigned by the master during the connection setup. Each Bluetooth node can have a maximum of 7 master links and 8 slave links[5].

2.5 Poll Manager

A Bluetooth master node can have a maximum of 7 slaves and the PollManager module is responsible for managing these master-slave links within a piconet. In particular, PollManager decides which active slave to poll in every master time slot. The default PollManager implements a simple round-robin scheduling scheme.

2.6 Link Manager Protocol (LMP)

The LMP module implements several operations that are used for link setup and control. More specifically, LMP is responsible for

1. Setting up and configuring a new link,
2. Switching master and slave roles on a particular link,
3. Putting a link into low power modes such as Hold, Sniff and Park, and
4. Tearing down an existing connection.

[5] Bluetooth Specification does not limit the number of slave links. The maximum number of links can easily be configured in Blueware.

Link Manager communicates with neighboring devices using the Link Manager Protocol to carry out specific link control operations. As shown in Figure 2, each link or connection is managed by an instance of LinkController and an instance of LMP.

The LMP module also maintains a number of output packet queues. There are three separate output queues for packets associated with the LMP layer, the L2CAP layer, and the rest of the upper layers. The LMP layer packets are given the highest priority whereas the L2CAP packets (application data) are given the lowest priority. Blueware implements queues using BTQueue class which provides a uniform interface across existing queuing schemes such as RED and Fair Queuing. By default, DropTail queues are used.

2.7 Host Module

The Host module implements the Host Controller Interface (HCI) that separate upper and lower layers. In some Bluetooth systems, the lower layers, Radio, Baseband, LinkController and LinkManager, may be implemented on a single chip whereas the rest of the layers and applications could be running on a separate host processor. HCI cleanly defines the separation of lower and upper layers using HCI commands and events. As explained later, Blueware provides an extensible HCI event mechanism so that multiple scatternet protocols can interoperate.

2.8 Miscellaneous Aspects

In this subsection, we present various implementation aspects of Blueware that are interesting. One challenge in implementing Baseband is that multiple operations could be occurring simultaneously and some operations are bound to fail upon interruption. Imagine a situation where a periodic Page Scan event gets triggered while the node is in the middle of setting up a connection. To solve this problem, we have introduced a notion of *session* at the Baseband module. The Baseband will change its state only if there is no other Baseband sessions in progress. Thus, in the aforementioned scenario, the Page Scan operation has to wait until the connection is properly established. This design effectively serializes the Baseband operations and avoid undesired effects. The layers above Baseband such as LMP can also find out whether there is any ongoing session in progress. Blueware also allows Host to setup an un-interruptible session.

Blueware allows switching of roles between individual nodes. In particular, when a master node switches its role with one of its slave, it becomes a

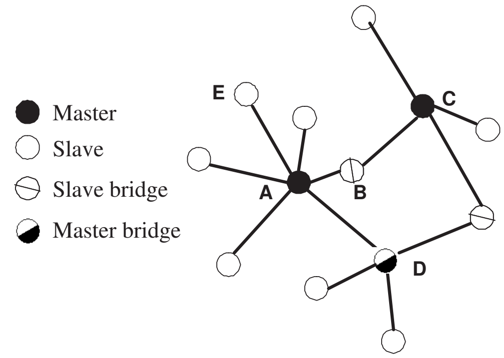


Figure 5: A Bluetooth scatternet

master-bridge node (see Figure 3 acting as a slave in the newly created piconet where its former slave is now the master, and as the master of its own piconet as before. Note that the role switch operation does not affect other slave nodes in the piconet.

A node can enable(disable) various link policies such as role-switching and hold mode by invoking appropriate HCI commands. The LinkManager will carry out the corresponding LMP operations based on these policies.

3 Support for Scatternet Protocols

Blueware provides an extensible architecture for various scatternet formation and scheduling schemes. Figure 5 shows a Bluetooth scatternet containing 3 piconets. The piconets are connected by two different types of bridge nodes: slave and master. In Figure 5, master bridge D participates as slave in A's piconet and as master in its own piconet. Slave bridge B communicates as slave in both A and C's piconets. Unlike previous work [1], our simulator allows both kind of bridges. In addition, a node can request to change role with its peer by issuing a corresponding HCI command.

In dynamic environments, where nodes arrive and depart arbitrarily, it is important that the scatternet formation scheme and the link scheduling scheme interoperate seamlessly so that both efficient connectivity and communication are possible. The scatternet formation scheme dictates how connected nodes in an existing scatternet conduct neighbor-discovery operations so that network partitions can be healed and new arriving nodes can be connected. Meanwhile, a link scheduling scheme is essential in coordinating communication between neighboring nodes to carry out successful packet transfers. Since Bluetooth devices, each with a single radio chip, can only be active on one channel at a time, the link scheduling scheme must schedule communication events on different links based on the availability and traffic patterns. In particular, link scheduling schemes can invoke appropriate HCI commands to hold or park a particular link for a certain amount of time. Thus, a task scheduling mechanism is essential so that various discovery and communication events can be carried out without scheduling conflicts.

Blueware also provides a flexible interface so that various scatternet formation, link scheduling, and packet routing schemes can interoperate seamlessly. In particular, the simulator defines the HCIEventsHandler interface that each of these schemes can implement so that they can receive appropriate events related to discovering and communicating with neighbors. The rest of this section explains in detain the TaskScheduler module and HCI Events handling mechanism.

3.1 Task Scheduler

Although the Bluetooth specification provides necessary HCI commands to carry out Inquiry or Page operations and to activate or hold a communication link, it does not provide any commands for scheduling those operations so that efficient scatternet communication protocols could be developed. We implement a task scheduling framework in Blueware and suggest that such a framework be incorporated in the next version of the Bluetooth specification.

Figure 3.1 shows the main routines provided by the TaskScheduler module. The scatternet formation and link scheduling schemes can schedule future events using the Schedule methods. *Event* is a C++ object and contains information that the Host module wants to store. *Schedule* routine creates a new task of at least *mindur* duration at the earliest time possible after the *start* time. *ScheduleHard* is similar to Schedule except that it creates a new task only if the start and finish times of the task fall between *start* and *finish*. An existing task can be canceled using the *Cancel* method. There are also other routines that can list all existing tasks that are currently scheduled. Using these methods, the Host can schedule various tasks effectively.

TaskScheduler interacts with the lower Bluetooth layers such as LMP and Baseband to provide the necessary functionality for scheduling tasks. TaskScheduler maintains an ordered list of tasks each of which is denoted by a triple $(event, start, finish)$. TaskScheduler also maintains a timer to trigger events related to scheduled tasks. For each scheduled task, TaskScheduler reports two out of three possible events back to the Host which implements the scatternet formation and link scheduling schemes. The event *type* can be *Begin*, *End*, or *Cancel* denoting that the scheduled interval associated with the *event* has begun, has ended or has been canceled respectively. If the *Begin* event is received, the Host requests Baseband to carry out the corresponding operations associated with the event.

```
Schedule(event, start, mindur);
ScheduleHard(event, start, finish, mindur);
Cancel(event);

// callback
handle(event, type);
```

Figure 6: Major routines provided by TaskScheduler.

Since the scatternet formation scheme operates independently from the link scheduling scheme, scheduling conflicts may arise between different types of tasks. We distinguish between two kinds of task: i) Form and ii) Comm. The Form tasks represent scatternet formation related operations such as Inquiry whereas a Comm task specifies when and how long a node communicates on its communication channel. TaskScheduler resolves the conflicts using the following rules:

1. The ongoing task is never interrupted or preempted by another task.
2. A future Form task is scheduled as soon as the current task's finish time has reached. If necessary, all the existing tasks which fall in between the start and finish times of the Form task are canceled.

The rules guarantee that every Form task will be scheduled no later than the maximum duration of a Comm task. Form tasks are given higher priority since carrying discovery operations out in a timely fashion shorten the time required to create a connected scatternet. We note that various policies can be set between the scatternet formation scheme and the link scheduling scheme to resolve scheduling conflicts. For instance, a scatternet formation scheme may only require the task scheduler to apply the second rule if the Form task is to carry out Page or Page Scan operations instead of the Inquiry and Inquiry Scan operations.

3.2 HCIEventsHandler

Blueware provides a flexible interface so that various scatternet formation (Form), link scheduling

(Sched) and packet routing (Rtr) schemes can interoperate seamlessly. As shown in Figure 2, the simulator defines the HCIEventsHandler interface that each of these schemes can extend so that they can receive appropriate events related to discovering and communicating with neighbors.

As mentioned before, the Host module implements most HCI commands to allow upper layers to easily use the functionality of the lower Bluetooth layers. To report the results of a particular operation, the Host module callbacks the upper layer that issued the HCI command. Blueware allows each upper layer module to register for all HCI events. When a particular HCI event is triggered by a lower layer, the Host module iterates through the registered event handlers and delivers the HCI event sequentially in the order of their registration.

3.3 Implementing Scatternet Protocols

We have developed a scatternet formation scheme, TSF, and a link scheduling scheme, LCS, using the Blueware simulator. TSF (for Tree Scatternet Formation) is an efficient topology construction algorithm that are targeted for dynamic environments [14]. TSF extends TopologyConstructor class which in turn extends HCIEventsHandler. LCS (for Locally Coordinated Scheduling) is a meeting-based protocol that schedule Bluetooth links effectively by coordinating one-hop neighbors [13]. LCS extends TaskScheduler[6] which extends HCIEventsHandler. Extending TaskScheduler is unnecessary if the link scheduling scheme does not want to change the default policy provided by TaskScheduler. We note that HCIEventsHandler contains a small number of HCI events that are not specified in Bluetooth Specification 1.1. These new HCI events are necessary if an efficient link scheduling scheme like LCS is to be implemented strictly above the Host module. However, the link scheduling schemes may also be integrated with the lower layers, in which case, those new HCI events are unnecessary.

4 Running Simulations

In this section, we explain in detail how to run simulations using Blueware. There are two main aspects of scatternet protocols that one can evaluate using Blueware. First, users can specify the node arrivals model to evaluate the delays associated with forming a connected scatternet topology and energy consumed (in terms of the times nodes spend in busy states). Second, a mixture of TCP and CBR flows can be setup to evaluate the throughput, delay and energy consumption. In short, various scatternet formation and scheduling algorithms can be evaluated using Blueware.

[6]There are no particular reasons why TaskScheduler should extend HCIEventsHandler. We do this for implementation convenience.

4.1 Modeling Node Arrivals

Figure 4 shows the example TCL script to run the simulation. Configuration parameters can be set in the Sim array. $NumDevices$ specifies the total number of nodes which must not exceed the maximum number of nodes allowed by Blueware. You can change the maximum value by setting $MaxNumNodes$ in bt-def.h to a desired value. X and Y defines the length and width of the area (in meters) where nodes are placed randomly. If they are not specified, all nodes will be placed within the radio range of each other. $Seed$ is simply the random seed used for the simulation run. Using the same $Seed$ results in the same sequence of random numbers and thus, provides predictable outcome.

$SimulationTime$ specifies how long the simulation last. The interval of node arrivals is defined as $[ArrivalStart, ArrivalStart + ArrivalTime]$. In addition, the number of nodes that arrive simultaneously at time $0s$ can be specified with $NumStartDevices$. By configuring these parameters, users can generate various arrival scenarios.

4.2 Generating Traffic

The simplest way to generate traffic is to setup a flow between a source and destination pair. Src and Dst specifies the IP addresses of the source and destination respectively. Currently, two kinds of applications are supported: TCP and CBR. $TrafficRatio$ can be set to get a desired mixture of flows. FTP applications are used to generate TCP traffic whereas CBR agents are used to generate UDP packets periodically at a rate defined by $CBRRate$. The packet size of each type of flow can also be configured ($TCPSize$ and $CBRSize$). $AppStart$ defines the start time of all applications. It is important to note that if $AppStart$ is too early, there may not exist a connected scatternet and thus, the destination may be unreachable.

$Topo$ and $Sched$ specifies which topology formation and scheduling schemes are used respectively. As mentioned before, we have provided implementations of TSF and LCS in Blueware 1.0.

```
# number of nodes: the node's bd_addr
set Sim(NumDevices) 20

# length and width of the area in meters
set Sim(X) 7.07
set Sim(Y) 7.07

# random seed for this simulation run
set Sim(Seed) 12

# simulation time in seconds
set Sim(SimulationTime) 235

# start time of node arrivals in seconds
set Sim(ArrivalStart) 0
# interval of node arrivals,
# ArrivalEnd = ArrivalStart + ArrivalTime
set Sim(ArrivalTime) 0
# number of devices that arrive at time 0s.
set Sim(NumStartDevices) 0

# source and destination addresses
set Sim(Src) 0
set Sim(Dst) 14
# the start time of applications in seconds
set Sim(AppStart) 35
# traffic ratio of TCP to CBR:
# 1 means 100% TCP and 0 means 100% CBR
set Sim(TrafficRatio) 1
# total number of applications
set Sim(NumApps) 1
# TCP packet size in bytes
set Sim(TCPSize) 512
# CBR packet size in bytes
set Sim(CBRSize) 335
# CBR data rate in bps
set Sim(CBRRate) 5kb

# link scheduling scheme
set Sim(Sched) LCS

# topology formation scheme
set Sim(Topo) TSF
# tsf parameter to test healing performance
set Sim(TSFheal) 0

# alternative tracing support
set Sim(Trace) OFF
set Sim(AgentTrace) OFF

source ./topo.tcl
```

Figure 7: Example TCL script

4.3 Tracing and Analysis

Blueware provides extensive tracing support. Trace statements can be put anywhere in Blueware code by using *TRACE_BT* macros. One of the 32 possible trace levels must be specified when using the macro. Individual trace levels can be turned on and off by setting *trace_level* variable in bt-baseband.cc. At the minimum, *trace_level* should be set to *LEVEL_ACCT* so that analysis can be done using the scripts provided. *Trace* and *AgentTrace* can also be turned on to use the alternative tracing support provided by Bluehoc.

When *LEVEL_ACCT* is turned on, you can use the analysis script *analys.pl* to analyze the simulation run. Figure 4.3 shows the sample output of *analys.pl*. For space constraints, we have removed some parts of the output. *ConDelay* (for connection setup delay) is the time it takes for a node to establish its first communication link. *LastLink* is the total time elapsed until a connected scatternet containing all nodes is formed. The output also shows the break down of the time nodes spend in each Baseband state as well as the traffic statistics of each application.

```
Node      ConDelay
0   ...   2.66e+00
1         3.28e+00
2         2.66e+00
3         3.10e+00
Avg       2.92e+00 StdDev   3.16e-01
SuccRate 100.00 LastLink 3.28 ..[
.
.
.
Node  Life      INQ      INQ_SC       CONN
0  2.71e+02  0.00e+00  6.54e-02 (0) 1.26e-02
1  2.71e+02  1.33e+00  1.92e-01 (1) 1.28e-02
2  2.71e+02  5.30e+00  1.12e-01 (0) 3.07e+00
3  2.71e+02  9.81e-01  1.85e-01 (0) 9.92e+01
AVG          1.90e+00  1.39e-01     1.00e+02
Avg disc time(%):         1.09    ...
.
.
.
Traffic Stats
src->dst Thruput  Delay   NPkts  Dur ...
0->1  2.58e+05  3.81e-01  12578  2.00e+02
Total 2.58e+05  3.81e-01  AvgHops 2.00e+00
.
.
.
```

Figure 8: Output of the analysis script

5 Limitations

Blueware simulator has its limitations. The simulator does not implement a reasonable interference model[7]. Clearly, a good and realistic interference model for Bluetooth is necessary to evaluate the performance of Bluetooth networks. Thus, when using Blueware, one must bear in mind that the results achieved using simulation are the best case results. We note that when evaluating topology formation schemes, a lack of interference model does not significantly impact the results when the node density is not too high. The reason is that Bluetooth Inquiry procedure uses ID packet that are very short (72 bits) and that the chances of ID packet collisions are rare. Without the interference model, the throughput and delay achieved by a link scheduling scheme are also the best possible results.

Although Blueware implements many aspects of Bluetooth, there are some features of Bluetooth that are not implemented. Blueware currently does not support synchronous connections (SCO) and also lacks proper support for QoS. Lastly, Blueware is not currently integrated with existing *ad hoc* routing schemes such as DSR and AODV. However, it does have a "hook" in place in bt-host.cc so that routing schemes can easily be integrated.

6 Summary

We have developed a Bluetooth simulator that close follows the Bluetooth Specifications. Although the simulator (like any other simulators) cannot possibly provide an absolutely correct and realistic model of real world scenarios, it offers a quick and easy way to evaluate novel scatternet protocols. Building the simulator gives us insights in understanding many of the engineering difficulties as well as performance aspects. We have also offered suggestions to be included in the future Bluetooth Specification so that various scheduling and topology formation schemes can be interoperated seamlessly. Blueware 1.0 is available for public use and can be downloaded from http://nms.lcs.mit.edu/projects/blueware.

7 Acknowledgments

The efforts of implementing Blueware are significantly eased by an existing implementation of the Bluetooth simulator, Bluehoc, and as such we would like to thank the Bluehoc team.

[7]There exists some code to deal with interference. In particular, bt-distfer.h contains a table of forward error rate provided by the Bluehoc team

References

[1] Bluetooth Extension for ns. `http://oss.software.ibm.com/developerworks/opensource/bluehoc/`.

[2] Specification of the Bluetooth System. `http://www.bluetooth.com/`, December 1999. Bluetooth Special Interest Group document.

[3] J. Bray and C. Sturman. *Connection Without Cables.* Prentice Hall, 2001.

[4] Blueware: Bluetooth simulator for ns. `http://nms.lcs.mit.edu/projects/blueware/`.

[5] J. Haartsen. The Bluetooth Radio System. *IEEE Personal Communications Magazine*, pages 28–36, February 2000.

[6] N. Johansson, F. Alriksson, and U. Jonsson. JUMP Mode- A Dynamic Window-based Scheduling Framework for Bluetooth Scatternets. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001.

[7] C. Law, A. K. Mehta, and K.-Y. Siu. Performance of a New Bluetooth Scatternet Formation Protocol. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001.

[8] ns-2 Network Simulator. `http://www.isi.edu/vint/nsnam/`.

[9] A. Racz, G. Milklos, F. Kubinszky, and A. Valko. A Pseudo Random Coordinated Scheduling Algorithm for Bluetooth Scatternets. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001.

[10] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed topology construction of Bluetooth personal area networks. In *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

[11] G. Tan. Interconnecting Bluetooth-like Personal Area Networks. In *1st Annual Student Oxygen Workshop*, pages 45–46, Gloucester, MA, July 2001.

[12] G. Tan. Self-organizing Bluetooth Scatternets. Master's thesis, Massachusetts Institute of Technology, Jan. 2002.

[13] G. Tan and J. Guttag. A Locally Coordinated Scatternet Scheduling Algorithm. In *The 27th Annual IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, Nov. 2002.

[14] G. Tan, A. Miu, H. Balakrishnan, and J. Guttag. An Efficient Scatternet Formation Algorithm for Dynamic Environments. In *IASTED International Conference on Communications and Computer Networks (CCN02)*, Cambridge, MA, Nov. 2002.