Bluetooth Simulations for Wireless Sensor Networks using GTNetS *

Xin Zhang, George F. Riley
Department of ECE, Georgia Institute of Technology
Atlanta, GA 30332-0250
{*xinzhang, riley*}*@ece.gatech.edu*

Abstract

We introduce a simulation environment for wireless sensor networks using the Bluetooth wireless protocol in our Georgia Tech Network Simulator (GTNetS). Our goal is to explore the applicability of the Bluetooth protocol for wireless sensor networks. Our Bluetooth simulator implements detailed behavior of lower layers of Bluetooth protocol stack, including Baseband, LMP, L2CAP, and BNEP, with the emphasis on interference resilient and power efficient characteristics. The implementation is based on the design of GTNetS and will allow our Bluetooth simulator to be used for large-scale network simulations once an effective scatternet protocol implementation is completed. Further, our Bluetooth simulator is designed by using a strict layered model, which makes it easy to extend in order to accommodate modifications to Bluetooth protocol or new MAC protocols for sensor networks. We present some simulation results with a simple network configuration to measure the performance of Bluetooth networks in terms of power consumption.

1. Introduction

Wireless Sensor Networks (WSNs) have drawn more and more attention as their applications grow to ubiquitous domains. The concepts of micro–sensing and wireless communication in WSNs promise a wide variety of applications such as military battlefield surveillance, biocomplexity mapping of the environment, telemonitoring of human physiological data, and seismic structure response [1]. Bluetooth [2] is a short-range wireless system designed to replace a physical cable connecting portable or fixed electronic devices. It is a low power, low cost design operating in the unlicensed ISM band at 2.4 GHz. The use of time-division multiple access (TDMA) and frequency hopping spread spectrum (FHSS) makes Bluetooth a robust protocol resilient to interference.

Bluetooth enabled sensor nodes have been used for prototyping wireless sensor networks since many off-the-shelf Bluetooth devices can be used as gateways to construct sensor networks and bridge with external networks such as the Internet [3, 4]. Developing a Bluetooth simulation environment unwraps the Bluetooth protocols encapsulated in device hardware and makes protocol modification and optimization tailored for wireless sensor networks feasible. As an enhanced generation of the Berkeley Mote, Intel Mote [5] is being developed based on Bluetooth. Bluetooth's characteristics of interference resilience, interoperability with a large variety of devices, increased link reliability, and security features make it appealing to wireless sensor networks. Moreover, the synchronized TDMA access scheme in Bluetooth allows the sender and the receiver of a communication link to idle between active slots, thereby reducing overall power consumption which is essential for wireless sensor networks.

The Bluetooth simulator for wireless sensor networks we have developed provides a detailed and precise Bluetooth stack implementation in order to benefit the simulation of sensor networks as well as other Bluetooth applications.

One of the earliest efforts to develop a Bluetooth simulator was Bluehoc [6] from IBM. It extends *ns2* [7] and implements the fundamental Bluetooth piconet connection functionalities. Another Bluetooth simulator, Blueware [8] developed by MIT, is based on Bluehoc. Blueware addresses the issues of scatternet formation and link scheduling schemes. All of these existing Bluetooth models are based on *ns2*. Although *ns2* is a venerable and widely used network simulator, it can only model networks with a few hundred to a few thousand network elements [9]. Normally wireless sensor networks have large-scale topologies that consume excessive amount of simulation resources. *GTNetS* is

* This work is supported in part by NSF under contract number ANI-9977544.

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 **IEEE**

designed for scalable distributed network simulations. In this case, it is necessary to develop a Bluetooth model for this large-scale network simulator. Our Bluetooth simulator is intent to be more complete and precise base on real packet transmission. It also includes Bluetooth Network Encapsulation Protocol (BNEP) to support network protocols over Bluetooth media. Additionally, our Bluetooth model for *GTNetS* implements power saving modes and power consumption measurement, which are essential for wireless sensor networks.

In the Bluetooth simulator for *GTNetS*, we implement the Bluetooth protocol stack according to the Bluetooth specification version 1.1 [2]. It includes Baseband with low power modes and power consumption measurement, LMP, L2CAP, and BNEP modules. With these modules, it is easy to develop simulations of wireless sensor network applications running on top of Bluetooth. Our simulator is also useful for the evaluation of improvements in the Bluetooth protocol and Bluetooth sibling protocols.

The remainder of this paper is organized as follows. Section 2 gives the motivation of using Bluetooth technology for sensor networks. Section 3 describes the Bluetooth protocol layers and their implementations in *GTNetS*. Section 4 presents some simulation experiments. Finally, section 5 summarizes the paper and gives future directions.

2. Motivation

Our Bluetooth simulator is designed to provide a simulation environment for wireless sensor networks adopting Bluetooth related technologies. However, the design is such that it can provide the basis for simulation models of other Bluetooth applications as well. There are several sensor network platforms using Bluetooth as their communication means such as the BTnode prototype from ETH Zurich [4], the iBadge from UCLA [10], and the Intel Mote from Intel [5]. The WINS prototype from UCLA [11] also relies on similar spread spectrum radio. For these designs, the choice of choosing Bluetooth as the radio type is based on a number of Bluetooth features as follows:

- **Ubiquity** Most sensor network nodes, e.g. the motes, have limited resources for data storage, processing, and power. They rely on base stations or gateways to connect to external networks for tasks requiring more storage or higher performance. Such gateways with Bluetooth interfaces are widely spread. If sensor nodes are equipped with Bluetooth, they can communicate seamlessly with these ubiquitous gateways. In addition, some sensor network applications need immediate reactions taken by actuators when certain events are sensed by sensors. If Bluetooth is supported by the sensor nodes, lots of commercial products with Bluetooth interfaces can be used as such actuators. Furthermore, using Bluetooth as the radio for sensor nodes enables easy communication with personal devices such as PDAs and laptops since most of them have Bluetooth interfaces. This allows direct user interaction with sensor nodes for data collection, controlling, and debugging.
- **Power Efficiency** For sensor networks, low power consumption is essential. The Bluetooth protocol allows the radio to enter low power modes while maintaining synchronization when no transmission or reception is active on a communication link. These modes greatly reduce the power consumption, as we later show with our simulation results. The modes used by Bluetooth, listed in increasing order of power efficiency, are: sniff mode, hold mode, and park mode.
- **Interference Resilience** Wireless sensor networks consist of a large number of elements. Radio level interference can be a significant problem when sensor nodes within communication range compete for a shared channel. Bluetooth, on the other hand, exploits frequency hopping spread spectrum. Sensor nodes within communication range can use separate channels to transmit data. Bluetooth's resilience to interference makes it a good candidate in the context of sensor networks.

3. Bluetooth Simulation in *GTNetS*

The *Georgia Tech Network Simulator* (*GTNetS*) developed by our research group was designed for efficient simulation of large-scale networks. It achieves good scalability by using distributed simulation methods, as well as an efficient design for both memory and computational resources [12]. The protocol stack architecture in *GTNetS* maps exactly to real networks and hardware. Each protocol stack is implemented as a stand-alone class in object-oriented C++. It is easy to understand and extend based on the existing simulation models. *GTNetS* supports a number of protocols at different layers.

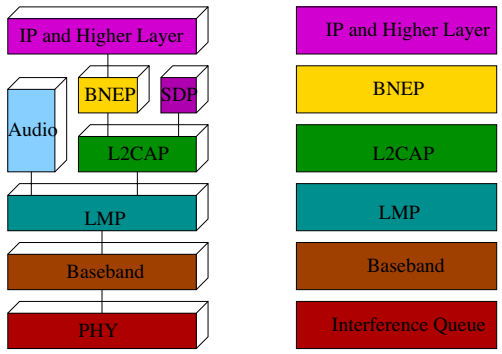
Our Bluetooth simulator takes advantage of the scalability of *GTNetS* as well as layered design architecture, and extends it for sensor network and Bluetooth network simulations. One can easily make further modifications to the Bluetooth protocols based on our Bluetooth simulator, in order to validate the performance

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 **IEEE**

and suitability of TDMA FHSS based schemes for sensor networks. Details on our implementation of the Bluetooth protocol in the *GTNetS* simulator are given in the following paragraphs.

3.1. General

The Bluetooth stack and *GTNetS* Bluetooth modules are shown in Figure 1. In *GTNetS*, a *Node* object represents the functionality of a network node. We derive the Bluetooth *BlueNode* class from *Node* class. Besides *Node's* common functionality, a *BlueNode* has a 48-bit Bluetooth device address, which identifies a Bluetooth device uniquely. Another extension of *BlueNode* is that it has a neighbor list. This neighbor list is used to store all the *BlueNodes* found within its radio range during the *Inquiry* phase. The neighbor list is checked and updated each time before a transmission occurs to insure reachability. In addition to the address information of the neighbor *BlueNodes*, the estimated clocks of these neighbors obtained at *Inquiry* are also stored in this list for the purpose of *Page* later. The *BlueNode* derived from *Node* also enables it to inherit the animation feature of *Node* in *GTNetS*.



(a) Bluetooth Stack (b) Simulator in GTNetS

Figure 1. Bluetooth stack and GTNetS Bluetooth modules

In *GTNetS*, a packet consists of a stack of *Protocol Data Units* (*PDUs*). Thus it is easy to extend the existing *GTNetS* packets for the Bluetooth simulation by defining all packet headers of the Bluetooth layer modules to be derived from class *PDU*. In this case, when a new packet is generated by the application and moves down to the Bluetooth protocol stack, the header for each layer is generated and pushed into the *PDU* stack. At the destination, the *PDUs* are pop out for processing. The *PDU* stack is shown in Figure 2. An *Freq* field is added to Baseband *PDU* for simulation use. This field has no correspondence to real network packets. The *Freq* field identifies the packet transmission hopping frequency. It is a number between 0 and 78 (we use the 79-hop system as default thereafter) corresponding to the hopping frequency. The fields and length of the packet header at each layer are implemented exactly as the definition in the specification. Therefore, it is easy to understand and simulate the Bluetooth behavior with high accuracy.

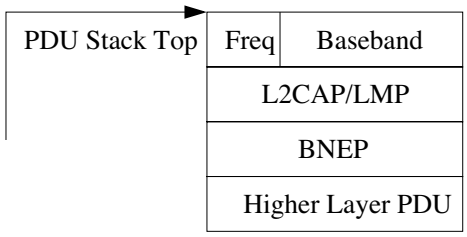


Figure 2. PDU stack in GTNetS Bluetooth modules

In the following sections, we will describe the Bluetooth protocol implementation in the order from lower to higher layers.

3.2. Baseband

The function of Bluetooth Baseband includes:

(i) Discover neighbor nodes by *Inquiry* in order to get the neighbor information including device address and clock.

(ii) Establish frequency hopping communication channel through *Page*.

3.2.1. Inquiry and Page A Bluetooth node has a 48-bit randomly generated device address. The native clock (*ClkN*) at every node is also randomly generated and asynchronous at the beginning. The clock is implemented as a 28-bit counter with the least significant bit (LSB) representing the clock tick of 312.5μs (1/2 of slot time 625μs). The clock counter is updated by scheduling a timer at every clock tick.

During the *Inquiry* process, the *Master* sends out two inquiry trains denoted *A* and *B*. Each train contains 16 distinct hop frequencies. The *Slaves* in the *Inquiry Scan* state listen to one of the 32 hop frequencies the *Master* is sending and change the listening hop frequency once every 1.28s. To avoid contention when multiple *Slaves* send responses at the same time, a backoff interval between 0 and 1023 time slots is randomly generated by every *Slave* when it gets the *Inquiry* message for the first time. After the timeout of

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 **IEEE**

the backoff timer, the *Slave* scans the *Inquiry* message again and transmits response that includes its device address and *ClkN* once the *Inquiry* message is received.

The time for the *Master* to get *Inquiry* response depends on the clock alignment between the *Master* and the *Slave* as well as the random backoff. From our simulation results, we found 3 to 5 seconds is sufficient for neighbor discovery.

After the *Inquiry* is complete, the *Master* has obtained the device addresses of each of the *Slaves* within its radio range and their native clock values (*ClkNs*). The *Master* can then selectively connect to the *Slaves* by using a *Page* message. The *Page* processing is similar to that of the *Inquiry*, except that the device address is used instead of general inquiry access code (GIAC). In this case, only the *Slave* with the specified device address will respond. The average and maximum time for *Page* is 1.28s and 2.56s respectively assuming the repetition time of each hopping sequence train is 128.

In the Bluetooth simulator for *GTNetS*, we implement all functionality of *Inquiry* and *Page* processes. Four timeout events (*INQUIRY_TO, PAGE_TO, PAGE_RSP_TO, and NEW_CONNECTION_TO*) are scheduled at the appropriate time. After the *Master* gets enough *Inquiry* responses or *INQUIRY_TO* occurs, selective *Page* can follow to establish frequency hopping communication channels.

3.2.2. Frequency Hopping As we mentioned in section 2, one of Bluetooth's compelling features for use in sensor networks is its resilience to interference. This comes from the use of frequency hopping spread spectrum (FHSS) scheme. Bluetooth operates at the 2.4GHz ISM band, which is divided into 79 RF channels with 1MHz spacing. The hopping channel is represented by a pseudo-random hopping sequence hopping through the 79 RF channels. Each hopping channel is divided into slots with a dwelling time of 625 μs. Since different piconets have different hopping sequences, there can be multiple piconets existing simultaneously within the same radio region.

We have implemented a frequency hopping kernel which generates pseudo-random hopping sequences. Our implementation has been verified by comparing to the sample data given in [2]. Three different combinations of addresses and initial clock values were used as the input for this validation test.

As we mentioned earlier, the *Freq* field is inserted into the packet to indicate the transmission frequency. If the packet is composed at its sending slot, the current clock value is used as the input to the frequency hopping kernel. However, in some cases, especially during the connection establishing phase, the control packets are generated as the responses to the received packets and scheduled for transmission in the next time slot in the simulator. Therefore, the corresponding clock counter is adjusted for the correct *Freq* insertion.

Another synchronization key is to distinguish the responses to *Page* messages sent in the first or the second half of a time slot. Figure 3 illustrates the message sequence for *Page*. During the *Page* process, two ID packets are sent by the *Master* within one time slot. The *Slave* responds either to the first or the second ID packet depending on the hopping frequency matching. The FHS packet from the *Master* always begins from the start of the following slot (slot3 in Figure 3). In this case, the *Master* needs to check the clock value when the response is received in slot 2 of Figure 3 in order to get the correct *Freq* for the FHS packet.

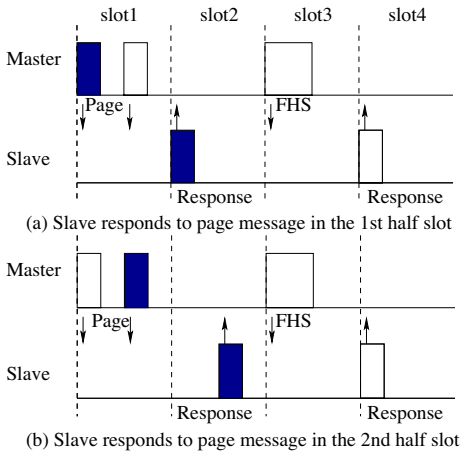


Figure 3. Page message sequence

The Bluetooth wireless channel in *GTNetS* is modeled by broadcasting packets to every neighbor node within the transmitter's radio range. When a Bluetooth non-broadcast packet arrives, the receiving node computes its current receiving frequency and compares it to the *Freq* field in the packet header to decide whether this packet should be received and processed.

The Bluetooth simulator in *GTNetS* provides a user friendly animation for frequency hopping. The color of a node is changing with its sending or receiving frequency. So it is intuitive from the animation to get synchronization information among the nodes within a topology. The screenshot is shown in Figure 4.

3.2.3. Power Saving Modes Power saving is essential for sensor networks since the sensing devices typically use battery power. The Bluetooth design defines methods that can significantly reduce power consumption by allowing the radio to enter low power modes between active communication slots. Since a standard

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 **IEEE**

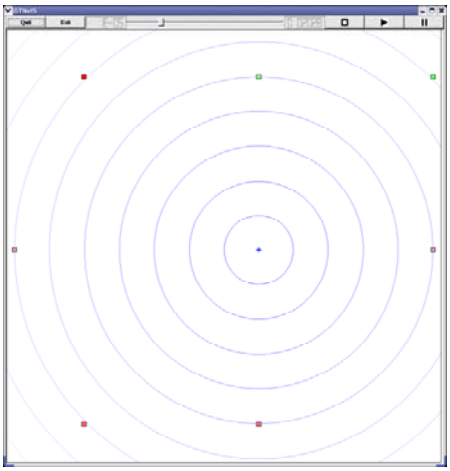


Figure 4. Screenshot

assumption of the duty cycle for a number of sensor network applications is about 1%, the Bluetooth low power modes can be highly efficient. We implement the three low power modes in Bluetooth: sniff mode, hold mode, and park mode. The implementation of the low power modes is by event scheduling at Link Manager Protocol (LMP) layer and transmission control by checking status word at Baseband layer.

The Bluetooth Baseband power saving modes can reduce power consumption. However, there is a tradeoff between power saving and latency including wakeup latency and buffer latency. Choosing the proper parameters is important to correctly predict power consumption. According to [2], the average clock drift should be less than 20 parts per million (ppm) relative to the slot time of $625\mu s$, and the instantaneous deviation should be less than $1\mu s$ when a node is in active mode. The uncertainty window allowed for *Master/Slave* misalignment is $10\mu s$. Thus, the interval (T_{max}) for the *Master* to provide synchronization messages must fulfill (1):

$$\begin{aligned} T_{max} &= \frac{(10\mu s - 1\mu s) * 10^6}{20ppm * 2} \\ &= 225ms. \end{aligned} \quad (1)$$

This indicates that the *Master* should transmit a synchronization message at least every 225ms. When a Bluetooth device operates at low power modes, the clock is driven by a low power oscillator with the accuracy of drift no more than 250ppm and jitter of 10 μs. If the synchronization time for a device returning from low power modes is one time slot ($625\mu s$), the maximum sleep time for a node (S_{max}) is shown in (2):

$$\begin{aligned} S_{max} &= \frac{(625\mu s - 10\mu s) * 10^6}{250ppm + 20ppm} \\ &\approx 2.3s. \end{aligned} \quad (2)$$

Therefore, *Slaves* can sleep within an interval of 2.3 seconds or less. Another consideration for the sleep time is the delay requirements of the applications. The application data from the *Master* to the sleeping *Slaves* must be buffered until the *Slaves* wake up from low power modes. In this sense, the sleep time is application dependent and can be negotiated through Link Manager Protocol (LMP) commands.

In addition to low power modes, we also implement the Bluetooth power saving in packet level: access code checking. If the access code checking (*CheckAccessCode*) fails, the packet is discarded and the transceiver consumes less power than receiving the whole packet.

The Bluetooth simulator in *GTNetS* has been designed from the beginning to model and track power consumption. This measurement is based on power consumption levels: transmitting/receiving, receiving with access code error, active without transmitting/receiving, and low power modes. The power consumption values are from the data sheet of the Bluetooth radio transceiver [13]. Our design is such that it is easy to adjust and extend these parameters when new hardware is introduced.

3.2.4. Link Control To provide reliability for the point to point Baseband link, we implemented a link control mechanism. It includes a one bit acknowledgement indication *ARQN* for Automatic Repeat Request (ARQ) and a one bit sequence number *SEQN* in the Baseband packet header.

3.2.5. Piconet Management In a Bluetooth piconet, the *Master* can have up to 7 active *Slaves*. The *Master* communicates with each active *Slave* according to the scheduling scheme. In our simulator, a round-robin scheduling scheme is implemented. It is easy to make extension to support other scheduling schemes. To avoid the initiation of multiple link control and link management objects for each master-slave link within a piconet, the concept of context is introduced. The content of the context includes all the information specific to a master-slave link such as active member address, slave device address, *ARQN*, *SEQN*, etc. When the *Master* changes the active *Slave* to which it wants to communicate, the method *SwitchContext* is called. In *SwitchContext*, the current master-slave link parameters are saved and the next scheduled master-slave link parameters are pop out as the current context.

3.3. Link Manager Protocol (LMP)

The LMP protocol is used for link initialization, security, and control. The functionality implemented in our Bluetooth simulator includes connection setup/tearing down, link supervision for

detecting devices moving out of range, quality of service parameters negotiation, authentication, multislot packets handling, and low power modes switching. LMP packets are differentiated from Logical Link Control and Adaptation Protocol (L2CAP) packets, which may include application data, by one bit in header and have the higher priority. Thus a separate queue is used for LMP packets.

3.4. Logical Link Control and Adaptation Protocol (L2CAP)

The L2CAP layer is above the Baseband protocol and interfaces with upper level protocols as shown in Figure 1. The L2CAP module in our simulator provides protocol multiplexing, segmentation, reassembly, and quality of service configuration.

L2CAP is based on the communication channel model. There are three types of L2CAP channels, as follows. The Bidirectional signalling channel, (class *L2capSignalChannel*), is the unique channel between any two entities. It carries all channel control commands. Connection-oriented channels (class *L2capConnChannel*) are used for bidirectional point-to-point connections. Unidirectional connectionless channels provide point-to-multipoint communication. Every endpoint of an L2CAP channel is identified by a logical channel identifier (CID). The signalling channel and connectionless reception channel CIDs are fixed, while the connection-oriented channel CIDs are dynamically allocated by calling the method *AllocateChannelID*. The configuration and data packets exchange are all based on the concept of channels. When an *L2CAP* object is constructed, an *L2capSignalChannel* object is constructed at the same time. The *L2capConnChannel* object is constructed by the corresponding *L2capSignalChannel* when data transmission is required.

3.5. Bluetooth Network Encapsulation Protocol (BNEP)

Sensor network nodes such as the Intel Mote [5] employ Bluetooth as the radio media to construct the mesh sensor network and exchange information within it. It is necessary to further extend this network to connect to some external networks, e.g. Internet, through gateways. Bluetooth Network Encapsulation Protocol (BNEP) provides an interface to transport common networking protocols over the Bluetooth media. It provides capacities that are similar to capacities provided by Ethernet. BNEP supports networking protocols such as IPv4 and IPv6.

Our Bluetooth simulator contains a detailed model of the function of BNEP, including BNEP connection control and the interface with L2CAP. With BNEP, it is possible to build a Bluetooth network access point as a bridge between Bluetooth devices and an Ethernet network (for example), as well as sending IP packets between Bluetooth devices.

4. Experiments

We ran some simple sensor network data collection application scenario simulations based on a Bluetooth piconet configuration. Our goal was to demonstrate the effectiveness of Bluetooth low power modes and measure energy consumption under different modes and traffic loads.

We modeled the traffic generated by the sensor nodes with a simple exponential on-off model. This is a typical traffic type for sensor applications such as habitat and environment monitoring. The on period follows an exponential distribution with its mean value equal to the duty cycle multiplied by the period, while the off period is an exponential distribution with the mean value of $(1 - dutycycle) * period$. To model traffic generated by medium data rate sensors, the traffic generating rate for the on period is set to be $64kb/s$. This is corresponding to the maximum DM1 packet rate ($108.8kb/s$) for Bluetooth Baseband considering BNEP and L2CAP packet headers. Moreover, this is also the worst case scenario for light sensor traffic and the case for aggregated sensor traffic close to the sink. The packet size follows DM1 packet requirement.

For the first set of experiments, we investigate the effect of Bluetooth hold mode on the tradeoff between average packet delay and power saving from entering this low power mode. The active periods between hold modes have the same exponential distribution as the traffic on period. This setting allows the Bluetooth links to deliver the generated traffic in a timely manner. We varied the time during which the nodes stay in hold mode in order to introduce different packet delay. The traffic loads are set to be 1%, 2%, and 5% duty cycle respectively considering the typical assumption for sensor network applications.

Figure 5 shows the percentage of hold time vs. average packet delay. The percentage of time that a node stays in hold mode is in direct proportion to the amount of energy saving. As we list in Table 1, the energy consumption is 20mA when a node is in active mode without any transmission or receiving (ActiveNoTxRx), while it is only $60\mu A$ in low power modes such as hold mode. Putting an idle (without Tx/Rx) node into low power modes wisely is energy efficient

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 **IEEE**

since idle in active mode is a major energy drain for typical MAC protocols. The tradeoff for energy saving in the low power mode is sleep delay. In Figure 5, when the traffic load is very light (1% duty cycle), the sensor nodes can enter hold mode for 90% of the entire simulation time while only incur 0.15s average sleep delay per packet. As the traffic load increases, the average sleep delay per packet goes up for a specific percentage of hold time. This is the trend demonstrated by the three curves in Figure 5 representing 1%, 2%, and 5% duty cycle of traffic. However, even with 5% duty cycle of traffic load, the sensor nodes can stay in hold mode for 91% of the simulation time and the average sleep delay is only 1.27s/packet. This level of average packet delay is acceptable for lots of sensor application scenarios. Therefore, putting sensor nodes in hold mode under light traffic can save much energy without introducing large packet delay. In addition, adjusting the time instant to enter hold mode and the duration to stay in hold mode according to the specific application and quality of service requirements can further improve the performance in terms of power consumption and packet delay.

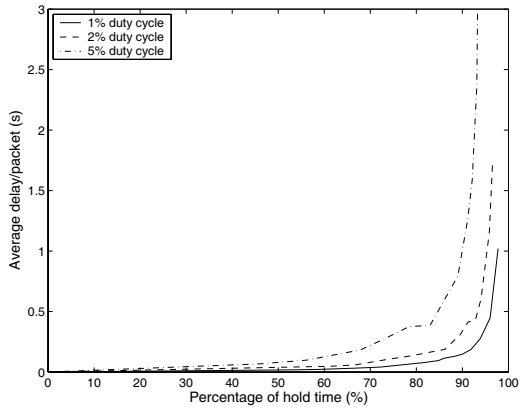


Figure 5. Percentage of hold time vs. average packet delay

Another set of experiments were conducted to measure the value of energy consumption with and without Bluetooth sniff mode as well as investigate energy distribution on each radio state. These states include transmission (Tx), receiving (Rx), receiving access code error (RxAcErr), active without Tx/Rx (ActiveNoTxRx), and low power modes (sniff, hold, and park). Table 1 shows the average electrical current under different states.

In these experiments, we varied the traffic loads from 1% to 20% duty cycle for both scenarios with and without Bluetooth sniff modes and measured the average energy consumption per packet. The parameters for sniff mode are: sniff interval (T_{Sniff}) 1 second, consecutive receiving slot ($N_{SniffAttemp}$) 16 slots, and sniff timeout value ($N_{SniffTimeout}$) 2 slots. The simulation results are shown in Figure 6. In both cases, the average energy consumption per packet decreases as traffic load increases. When the traffic load is light, without sniff mode, the energy consumption per packet is rather high. This is due to the large amount of energy spent in the state of ActiveNoTxRx. In contrast, under 1% duty cycle traffic load, when the Bluetooth sniff mode with 1 second interval is introduced, the energy consumption per packet in terms of mA is reduced to 51% of the former case without sniff mode. This energy saving benefit of sniff mode can be illustrated by the energy distribution on different radio states shown in Figure 7. In the case of 1% duty cycle of traffic, most of the energy is consumed in the ActiveNoTxRx state rather than spent for Tx/Rx. The sniff mode with the set of parameters declared earlier in this paragraph greatly reduces the energy drain in ActiveNoTxRx state. Therefore, the average energy consumption per packet with sniff mode is significantly reduced.

State	Current
Tx/Rx	50mA
RxAcErr	24.2mA
ActiveNoTxRx	20mA
Low Power	60μA

Table 1. Average energy consumption of bluetooth in various states

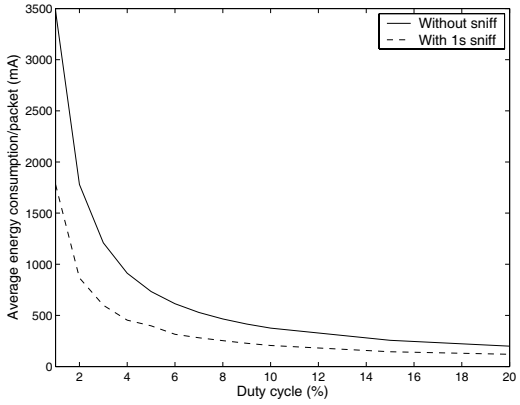


Figure 6. Energy consumption comparison for scenarios with and without sniff

Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)
1526-7539/04 $20.00 © 2004 **IEEE**

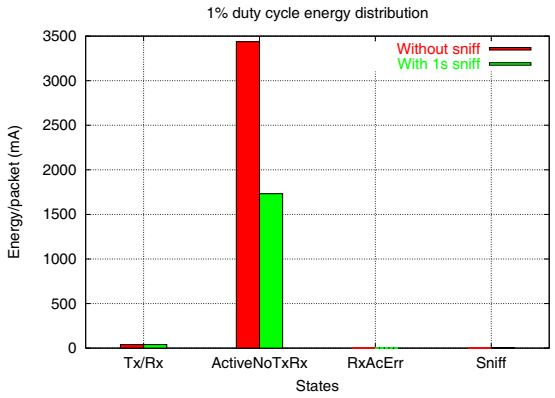


Figure 7. Energy distribution

With our Bluetooth simulator, it is easy to evaluate the performance of appropriate selection of Bluetooth low power modes according to the properties and requirements of wireless sensor network applications. The simulations are also valuable for correct parameters adjustment and protocol optimization.

5. Conclusions

In this paper, we presented a Bluetooth simulation environment for wireless sensor networks implemented in *GTNetS*. Our objective was to investigate the performance of wireless sensor network applications running on top of an interference resilient, power efficient environment with Bluetooth as a representative. We implemented the lower layers of Bluetooth protocol, including Baseband with low power modes and power consumption measurement, LMP, L2CAP, and BNEP. It provides a simulation environment with easy configuration and extensibility. Unlike Bluetooth prototypes with the lower layers encapsulated into the hardware without easy access, the Bluetooth simulator makes it comfortable for modifications to Bluetooth functionalities such as power management, time synchronization, scheduling mechanism, etc.

We also presented some experiment results with our Bluetooth simulator. The results show the power efficiency of Bluetooth low power modes, which makes it applicable for wireless sensor networks. The power measurement level provides a guideline for protocol improvement and sensor node life time estimation. The parameters used for the simulations are easy to adjust in order to optimize the performance.

Our Bluetooth simulator is based on the design of *GTNetS*, which is intended for large-scale simulations. To take full advantage of the efficient design, our next step is to support an effective scatternet protocol to aggregate the Bluetooth's interference resilient and power saving characteristics which are valuable for wireless sensor networks.

Furthermore, a detailed sensor node model including sensing unit, processing unit, and communication unit with battery model are being developed for *GTNetS*. In the future, we will incorporate our Bluetooth model with the sensor node model for realistic simulations of the practical world.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks,* vol. 38, no. 4, pp. 393-422, 2002.

[2] Bluetooth SIG, "Bluetooth specification version 1.1," Available HTTP: http://www.bluetooth.com.

[3] M. Leopold, M.B. Dydensborg, and P. Bonnet, "Bluetooth and sensor networks: a reality check," in *Proc. 1st International Conference on Embedded Networked Sensor Systems,* 2003, pp. 103-113.

[4] J. Beutel, O. Kasten, F. Mattern, K. Romer, F. Siegemund, and L. Thiele, "Prototyping wireless sensor network applications with BTnodes," in *Proc. 1st IEEE Europen Workshop on Wireless Sensor Networks (EWSN),* 2004, pp. 323-338.

[5] R.M. Kling, "Intel Motes: an enhanced sensor network node," in *Proc. International Workshop on advanced sensors, structural health monitoring, and smart structures,* 2003.

[6] IBM, "Bluehoc: Bluetootch performance evaluation tool," Available HTTP: http://www-124.ibm.com/developerworks/opensource/bluehoc.

[7] "The network simulator - ns2," Available HTTP: http://www.isi.edu/nsnam/ns.

[8] G. Tan, "Blueware: Bluetooth simulator for ns," MIT, Cambridge, MA, Tech. Rep. MIT-LCS-TR-866, Oct. 2002.

[9] G.F. Riley, "The Georgia Tech network simulator," in *Proc. ACM SIGCOMM Workshop on Models, Methods, and Tools for Reproducible Network Research,* 2003, pp. 5-12.

[10] UCLA, "iBadge," Available HTTP: http://nesl.ee.ucla.edu/projects/ibadge.

[11] UCLA, "Wireless integrated network sensors," Available HTTP: http://www.janet.ucla.edu/WINS.

[12] G.F. Riley, "Large-scale network simulations with GTNetS," in *Proc. 2003 Winter Simulation Conference,* 2003, pp. 676-684.

[13] J. Linsky, "Bluetooth and power consumption: issues and answers," Available HTTP: http://www.rfdesign.com, 2001.