(Re)Enabling support for the CC2420 chip on the TinyOS Simulator

Report for the EE652 Fall 2011 project at USC

Mo Zhu
Department of Electrical Engineering
University of Southern California
mozhu@usc.edu

Srikanth Nori
Department of Computer Science
University of Southern California
snori@usc.edu

Abstract

Simulation tools for networked devices are a valuable resource to speed up software development.

- TOSSIM (TinyOS SIMulator)[1] is a network simulator that is part of TinyOS [2] (an operating system for networked wireless sensors).
- The CC2420 chip [3] is a radio frequency transciever used in many wireless sensor devices.

The TOSSIM maintained as part of the main TinyOS trunk does not implement simulation of the CC2420 chip. This has been succesfully implemented outside the primary TinyOS trunk in the past [4], however this support has not been maintained with newer versions of TinyOS.

In this paper we describe in detail the process that was followed to add CC2420 simulation support to the latest TOSSIM. We also describe our evaluation method to confirm that the implementation is functional and accurate. We publish the results of our evaluations and show that the CC2420 is succesfully simulated within the TOSSIM environment.

1 Introduction

The process of developing software systems for wireless sensor networks (WSNs) faces some common challenges, including: rapid development and deployment, obtaining repeatable results across tests, debugging, and collection of data. These can ideally be addressed by providing a powerful simulation suite.

1.1 Need for simulation

Considering that most WSN hardware sensors (motes) are typically small, low-powered devices deploying code onto these devices is not trivial. It often requires connecting to a separate hardware interface (such as the USB/Serial port on Micaz[6] or Tmote Sky[7] motes). While this approach provides the developer a great level of flexibility in configuring the mote, it also becomes cumbersome to make repeated changes, and to rapidly test and debug code on the mote. Furthermore, if the software needs to be deployed onto a large number of motes, it quickly becomes a very time-consuming task to program each mote with a software update. Sensor network testbeds have addressed this problem by providing infrastructural support (i.e. a parallel hardware path to configure motes), but this may not be accessible and/or cost-effective for all users at all times. Data gathering can also become a slow and time-consuming task. If the motes are programmed to store data (such as test results, logs, etc.), developers may need to spend considerable amounts of time in retrieving the data from each mote.

Wireless transmissions are also very susceptible to environmental factors. Most WSN deployments use 802.15.4 protocols for communication, and this makes use of a network frequency band that is unlicensed and shared with other devices. This makes it susceptible to interference. Couple this with the fact that wireless transmissions are heavily influenced by other loss factors such as multi-path fading or diffraction and it becomes very difficult to obtain a controlled environment in which to test a network protocol. The same test performed at different times is likely to give widely varying results, often for reasons that are unrelated to the projects concerns. This makes the development process complicated since (especially in early stages of software development) having a controlled environment where repeatable tests can be performed can greatly speed up development and debugging by allowing developers to focus on issues with the system under development without worrying about environmental factors.

All these issues can be addressed by using a network simulation environment. As stated in [1], a robust simulator system greatly aids in systems research. Simulation can provide a controlled environment for development and experimentation, and a platform for performing repeatable experiments. Experiments can be repeated with similar results, and the data is easier to analyze to localize issues. Simulation can also be a cost-effective method for research groups to work on wireless networks without making large investments in testbeds. Developers can use simulations for preliminary testing (with lesser investment into testing infrastructure) and later deploy and test code on real-world testbeds.

1.2 TinyOS and TOSSIM

Simulation platforms operate in different ways. Some, such as NS-2 [8], provide a platform-independent environment for simulation. The algorithm to be simulated has to be separately implemented and executed within the simulator's environment. It is unlikely that code written for the real simulator can run on the final device on which it is being deployed. On the other hand, simulators such as TOSSIM

provide a method where the same application code targeted to be deployed on the end-device can be reconfigured and deployed into the simulator environment with minimal user intervention.

TOSSIM is designed specially for TinyOS, and many WSN protocols and applications are implemented using TinyOS. As such, adding support for a particular radio platform within TinyOS allows many users to make use of the implementation. The design and internals of TOSSIM are described in [1]. Some points relevant to this project are stated here.

- TOSSIM operates by recompiling the same code that runs on WSN hardware to run within the TOSSIM environment. It makes use of a modified compiler to compile nesC component graphs into a format usable by the simulation environemt.
- TOSSIM is a discrete event system simulator. The compiled code is then run within a system where a series of events are passed into the code and excuted "transparently". i.e. no modifications needed to run in a simulated environment.
- Essentially, TOSSIM takes the software system and replaces key hardware interfaces (such as the radio interface, sensor interfaces, hardware clocks, etc) with simulated components.
- TOSSIM does not simulate bit level network interactions. Instead, it models the behaviour of network interactions. E.g, instead of modeling latency it models contention and radio backoff, which are the causes of latency.

Certain components within TinyOS cannot be used directly. For example, it would make no sense to attempt to use the interface to the radio hardware directly in a simulator. TOSSIM provides a mechanism to handle such situations wherein components can have a parallel implementation that will be used when the code is compiled for simulation. The parallel implementation of that component may, e.g., interface with the simulator components instead of interfacing with the hardware. We make extensive use of this parallel implementation to add support for the CC2420 chipset.

1.3 Motivation

The authors of TinyOS maintain TOSSIM as part of TinyOS itself, and it is regularly updated to stay in synch with the latest features that TinyOS provides. However CC2420 is not supported as part of the primary TinyOS distribution. Consequently, many applications that make use of CC2420 specific features cannot be run within the simulation environment.

The HiNRG team at JHU [9] has added CC2420 support to TOSSIM in the past [4]. However, as can be seen from the version control snapshots, this has been implemented on a TinyOS version that is several revisions behind the latest revision [10].

The current revision of TinyOS adds support for many more features as compared to the codebase that the JHU team started with, such as:

- Support for multiple new platforms
- Support for source routing protocol
- Modified Printf implementation

As stated before, adding support for CC2420 to the latest TinyOS would be very beneficial for developers to prototype applications that use CC2420 features on TOSSIM before moving them to real WSN testbeds and deployments. Further, ensuring that it is on the latest version would allow developers to make use of the newer features of TinyOS.

1.4 Contribution

Through the course of this paper, we hope to make the following contributions:

Our primary objective is to describe the steps that we took to add CC2420 support to TinyOS. We describe the system architecture of the existing codebases that we started with at a high level. We then go into more detail specifically on the changes that we needed to make to enable CC2420 support. To the best of our knowledge, this is not available today and would be a valuable addition to the TinyOS source.

We also describe our evaluation methodology and present the results of our evaluation. We focus on one important class of WSN applications - Collection. In particular, we test our modified TOSSIM implementation by performing data collection using the Backpressure Collection Protocol [11] and describe the results that we obtained.

Furthermore, we attempt to describe the internals of TOSSIM and TinyOS to an extent where it should be possible for the reader to understand and hopefully adapt it for their own uses. We hope that this document can serve as a guide for future developers interested in understanding some of the internals of TinyOS and TOSSIM that are relevant to simulation.

2 Related Work

In this section, we will review some of the previous work done on wireless sensor networks. Two representatives are reviewed and discussed in detail. The first one is TOSSIM [1], on which our project is based on. The other is COOJA [5], which is the simulator for anther WSN operating system Contiki.

2.1 TOSSIM [1]

An accurate and stable simulator is the key to network protocol development. TOSSIM is a simulator for TinyOS. Obviously, the design of TOSSIM is closely related to that of TinyOS. By using the simulator, we can capture network behavior even when the network is at the scale of thousands of nodes. A probabilistic bit error mode makes TOSSIM expressive enough to capture a wide range of network interactions, while remaining simple and efficient.

2.1.1 TinyOS aspects related to TOSSIM

TinyOS is not a conventional OS. It is a set of programming resources for embedded system that enable application specific services. The OS builds abstractions of hardware as components. For example, calling the getData() command on a sensor component will cause it to later signal a dataReady() event when the hardware interrupt fires. On the other hand, some components are purely software implementation. However, the combination of split-phase operations and tasks makes this difference transparent to application developers.

Another aspect worth mentioning is the ”fan-in” and ”fan-out”, i.e. an event or command call path can traverse several components simultaneously. This makes debug very difficult if we do not have a decent simulator.

2.1.2 TOSSIM design principles

The compiling process determines the target is a mote application or TOSSIM library. This is very easy for programmer if they want to turn to simulator for debugging. By only replacing a few low-level TinyOS system components that uses hardware resources, TOSSIM can capture mote behavior at a fine grain, allowing a wide range of experimentation.

TOSSIM can simulate many motes at once because an individual motes uses very limited CPU and memory

2.2 COOJA [5]

COOJA is a simulator for Contiki sensor node operating system. A novel cross-level simulator enables holistic simultaneous simulation at different levels. The main difference between TOSSIM and COOJA is how several nodes are represented in the different simulators. TOSSIM changes the variables to arrays, where each element in an array corresponds to a node. In COOJA, all the nodes are executed one by one in the same process. In addition, TOSSIM only supports simulations of nodes at operating system level, while COOJA can work across instruction level, network level and operating system level.

2.2.1 COOJA design principles

The key character of COOJA is that it enables cross-level simulation: simultaneous simulation at many levels of the system. It combines low-level simulation of sensor node hardware and simulation of high-level behavior in a single simulation

COOJA simulates networks in which each node can be of a different type, not only in the application running in nodes, but also the node hardware.

2.2.2 COOJA architecture

A simulated node in COOJA has three basic properties:

- Node data memory
- Node type
- Node hardware peripherals

Several nodes in the network may share one node type. The levels supported by COOJA are as the following:

Network level - This can be very helpful when designing and debugging a, for example, routing protocol. At this phase, the hardware is usually not the focus. Instead, the network itself is the most important. Thus factors like radio medium quality and duty cycles are concerns.

Operating system level - COOJA simulates OS (Contiki) by executing native operating system code. Since the whole OS is executed, it is useful for example to test and evaluate changes in the OS libraries.

Instruction set level - It is possible to add new nodes with a very different underlying structure. For example, nodes connected to a Java-base microcontroller emulator can be used instead of a compiled OS (Contiki). Events in the node are simulated at a bit level.

Although COOJA can simulate nodes at the three levels, the individual node is always simulated at one of these levels. The main advantage of doing this is that nodes from each of the levels can co-exist and further interact with each other in the same simulation

3 System design

In this section we focus on describing the internal architecture of TinyOS as relevant to data transmission and reception.

The primary interface for data send/receive in TinyOS is the ActiveMessage. [1] describes them as: ”The TinyOS packet abstraction is an Active Message. AM packets are an unreliable data link protocol, and the TinyOS network stack handles media access control and single hop packet transmission. Active Messages provide precise timestamps as well as synchronous data-link acknowledgments.” Since we are primarily concerned with packet transmissions and reception we will focus on the operation of Active Messages.

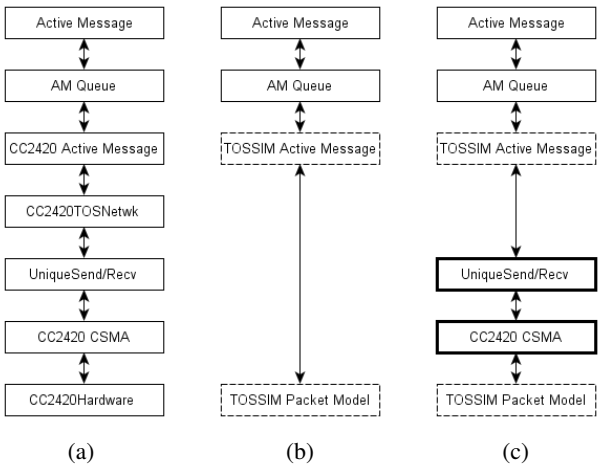

Figure 1: Application architectures

A typical application utilizing the ActiveMessage has an architecture as in Figure 1 (a). When executed within the simulated environment the architecture gets modified to the one shown in Figure 1 (b).

It can be observed that certain layers in the application change when run within the simulator. The CC2420 related layers are replaced by the equivalent layers that are designed for the TOSSIM. (Layers shown in dotted boxes indicate new layers)

Figure 1 (b) shows the default behaviour of TOSSIM when it runs applications that utilize Active Messages. It can be observed that there are no layers here that simulate the CC2420 behaviour.

TinyOS applications are free to bypass the ActiveMessage layer and directly interact with lower layers. Many applications, such as BCP, do make use of this functionality to utilize CC2420 interfaces to provide advanced features, such as snooping on broadcast packets or writing low level timestamps into outgoing packets. To enable simulation of such applications, it is necessary to provide proper support for the CC2420 radio stack within TinyOS.

The developers of [4] provided this exact feature. The architecture of applications run within the simulation environment in [4] is as described in Figure 1 (c), with additional changes to the underlying simulator (explained in subsequent sections) to support these features. The new replacement layers in this architecture are indicated with bold outlines.

3.1 Description of enhancements

Multiple changes were made at different layers of the TinyOS system to enable CC2420 support. All of these changes are taken from [4] into the latest TinyOS.

The enhancements can be summarized as:

- CC2420 application level interface support (PacketLink, ActiveMessage, etc).
- Packet level Link Quality Indication (LQI) and Received Signal Strength Indication (RSSI) information.
- Multiple channel support for 802.15.4 link (upto 16-channels).

We have described some of the critical changes here to the best of our understanding. Many changes that were primarily made for internal TinyOS purposes (e.g. include paths, build settings, etc) have been left out for the sake of brevity.

3.1.1 Simulation internals

- CpmModel modified to enable simulated reading of (RSSI). Multiple changes done for this across CpmModel, sim_mote, packet interface, etc.
- Duplicate suppression functionality added to enable unique send/recv functionality.
- Support added to change radio channel (consequently, noise and RSSI are simulated on a per-channel basis)

3.1.2 CC2420 modifications

The following modifications are provided specifically to enable simulation of CC2420. These are a combination of changes made in [4] along with additional changes needed to integrate this with newer versions of TinyOS.

- Parallel CC2420ActiveMessage implementation provides AMSend, Receive, Snoop, AMPacket, Packet and SendNotifier via TossimActiveMessageP. It also provides CC2420Packet, PacketAcknowledgements, SplitControl and PacketLink via CC2420RadioC.
- Parallel CC2420RadioC implementation wires to the appropriate low level components.
- Parallel CC2420 IEEE 802.15.4 implementation provides Ieee154Send/Receive and Packets functionality.
- IEEE EUI support added to CC2420ActiveMessage
- Parallel CC2420ControlC implementation provides CC2420Config functions such as get/set ShortAddr and get/set PanAddr.
- Parallel CC2420CsmaC implementation provides CSMA level send/recv and RadioBackoff functions for simulation
- Parallel CC2420PacketBody implementation provides access to headers, metadata etc from packets
- Parallel PacketLink implementation provided for TEP 127 functionality [12]. Also includes dummy packetlink layer.
- TOSSIM does not set CRC on outgoing packets (like CC2420), so disabled checking for CRCs during packet reception when run in TOSSIM in CC2420TinyosNetworkP
- Parallel CC2420Receive implementation provides a packet Receive interface connected to TossimPacketModel
- Parallel Unique send/receive implementation provides individual packet send/receive interfaces

Most of the modifications lie within modules related to CC2420ActiveMessage. A graphical architecture is presented in Figure 2. In this figure: a block indicates a TinyOS module, an arrow from src to dest indicates that dest module provides a service that is being used by the src module, and a number along an arrow represents the number of interfaces utilized. Blocks with bold borders are critical - these represent the modules that have been modified to enable CC2420 simulation.

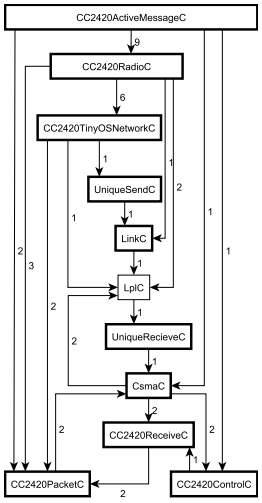

Figure 2: CC2420ActiveMessage layer modifications

3.2 Illustration

An implementation of the Backpressure Collection Protocol (BCP) is used to illustrate the modifications that were described in the previous section. The application performed data collection - i.e. multiple source motes send data to a single sink mote.

The BCP application architecture of packet transmission layers without simulation is as illustrated in Figure 3 (a). The same with CC2420 simulation enabled with our enhancements, it is as illustrated in 3 (b). The blocks with bold borders indicate the blocks that get modified to enable simulation.

It can be observed that the reconfiguration needed for simulation is transparent to the application code, only the underlying substrate is modified to enable simulation.

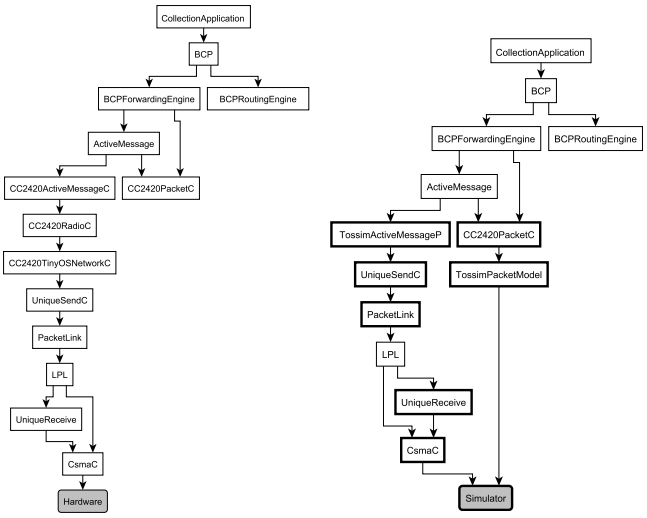

Figure 3: BCP architecture

4 Evaluation

4.1 Evaluation method

To evaluate the operation of the modified simulation we used the collection application described in the previous section. The test was a WSN consisting of multiple source motes transmitting data to a sink mote, as shown in figure 4.

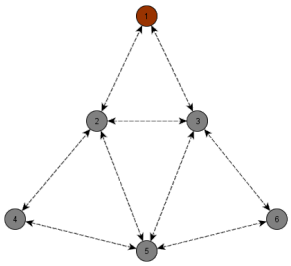

Figure 4: Test topology

We compare the throughput obtained in this application when run in the simulator with our changes, with the throughput obtained when run in the simulator as provided in [4]. The results are presented in Figures 5 and 6.

4.2 Evaluation results

In Figure 6 the expected goodput at sink if all nodes were transmitting all packets succesfully should have been around 250. The losses are because of the network capacity being saturated. The loss rates in both [4] and our modified code are comparable.

4.3 Additional verifications

We ran some additional tests to ensure that neither existing functionality nor recently added functionality continue to work. These tests include:

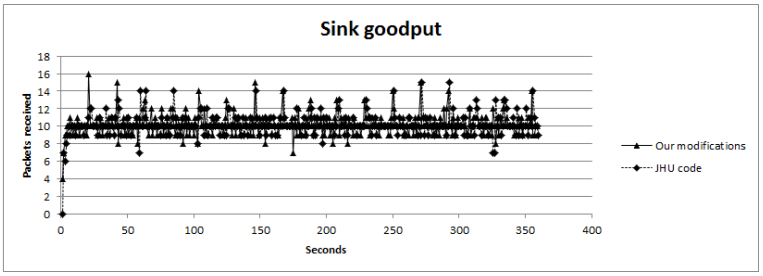

Figure 5: Nodes transmitting 2 packets per second to sink

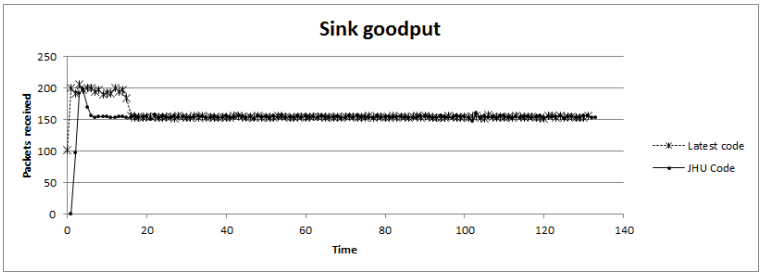

Figure 6: Nodes transmitting 50 packets per second to sink

- The TestRssi application succesfully reports appropriate RSSI values in the simulator.
- TinyOS new reference implementation of Source Routing Protocol (TestSrp) application succesfully works in the simulator.

5 Conclusion

In this paper we have discussed the architecture level details of TinyOS and TOSSIM. We have also described the steps taken in the implementation of CC2420 simulation support.

Based on the verification results, we can safely say that the CC2420 simulation support has been restored onto the latest TOSSIM. We have also been able to succesfully verify the collection application on a sample network, and have verified multiple other applications on varying topologies to confirm that the new system remains functioning along with other TinyOS features.

We are currently working on some additional features, listed in the next section.

6 Future work

There are certain CC2420 features that need to be implemented and verified in a simulation environment. These include:

- Low Power Listening mode implementation
- Security mode implementation

Further verification also needs to be done to confirm that real-world results and TOSSIM results match and that simulation remains faithful.

One of the key new features available within TinyOS is the BLIP implementation [13] to enable IPv6 communication. This makes use of RPL routing as described in [14]. There are two activities related to this that we are pursuing:

Node	Packets lost (JHU)	Packets lost(Our code)
2	15	16
3	16	15
4	21	96
5	20	20
6	172	63
Total	244	210

Figure 7: Total packets lost per node

- Porting of BLIP with RPL as-is onto TOSSIM.
- Modification of BLIP to use BCP, and porting onto TOSSIM.

7 References

[1] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. 2003. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys '03). ACM, New York, NY, USA, 126-137.

[2] TinyOS project website [Online]. Available: http://www.tinyos.net

[3] Texas Instruments CC2420 product page [Online]. Available: http://www.ti.com/product/cc2420

[4] (2009, Jan.). Git snapshot of TinyOS source with CC2420 support in TOSSIM [Online]. Available: http://hinrg.cs.jhu.edu/git/?p=mike/tinyos-2.x.git

[5] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In Proceedings of SenseApp 2006, Tampa, Florida,USA, 2006.

[6] MicaZ mote product website [online]. Available: http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html

[7] Tmote Sky product brochure [online]. Available: http://sentilla.com/files/pdf/eol/tmote-sky-brochure.pdf

[8] The NS-2 network simulator [online]. Available: http://isi.edu/nsnam/ns/

[9] HiNRG at JHU [online]. Available: http://hinrg.cs.jhu.edu/

[10] TinyOS latest main trunk [online]. Available: http://code.google.com/p/tinyos-main/source/checkout/

[11] Scott Moeller, Avinash Sridharan, Bhaskar Krishnamachari, and Omprakash Gnawali. 2010. Routing without routes: the backpressure collection protocol. In Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '10). ACM, New York, NY, USA, 279-290.

[12] David Moss, Philip Levis, TEP 127 - Packet Link Layer [online]. Available: http://www.tinyos.net/tinyos-2.x/doc/txt/tep127.txt

[13] Berkeley Low Power IP [online]. Available: http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip

[14] P. Thubert et al, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks" , IETF draft [online]. http://tools.ietf.org/html/draft-ietf-roll-rpl-19