# The COMPASS Multihop Framework for TinyOS

Ryan Stinnett
ECE Department
Rice University
jryans@rice.edu

November 14, 2007

## 1 Introduction

TinyOS 2.x supplies a well-designed single hop messaging layer for applications, but there is no standardized multihop framework. To address this issue, we surveyed a large number of applications [1] to determine the addressing techniques and messaging stack features used in sensor networks. From the results of this survey, we designed a multihop networking API [2] to address the needs of application designers at the messaging layer in a unified way.

This framework [3] intends to implement that API. The key goals of this include:

- Multihop routing with several addressing modes based on:
  - Node address (single node, multiple nodes, or a list of nodes)
  - Geographical region
  - Device hierarchy
- Intercept messages to be forwarded
- Control transmission effort
- Congestion management
- Support self-organized device hierarchies

Our design allows application designers to focus on the application layer without being bogged down by the details of routing protocol implementation. Similarly, routing protocol designers can make their improvements more accessible to the rest of the community since only a few wiring changes are needed to alter the routing protocol used by a given addressing mode, leaving application code untouched.

In its current form, the framework implementation supports addressing and multihop routing by a single node address. Our version of Dynamic Source Routing (DSR) [4] modified for TinyOS is used as the routing layer.

This document summarizes the key components in the framework. Section 2 describes the components needed to take advantage of this system in your own application. Section 3 delves into the internal components of the framework and how they work together. For more detailed information, consult the framework's HTML source code documentation [5].

## 2 Using the Framework in Applications

The single hop messaging layer [6] provided by TinyOS 2.x, also referred to as Active Messaging (AM), provides the fundamental components used by nearly all applications to move data to other nodes within range of a given source node. It supplies the *Send*, *Receive*, and *Snoop* interfaces for this purpose, which include the essential commands and events required to send and receive data to a node's neighbors. Packets can easily be separated by different packet types for each application, similar to the use of ports in TCP or UDP.

In an effort to reduce the amount of changes required in existing code, our multihop layer is used in a similar manner to the AM layer provided with TinyOS. The application level components and interfaces for sending, receiving, and manipulating packets of each addressing type are described below. Only brief descriptions are given for components that are very similar to their AM counterparts. A packet type is also used during component instantiation in the current structure, allowing for simple separation of the send and receive paths for various packet types.

### 2.1 Single Node Addressing

In this addressing mode, a packet has a single node as its destination, just like with the AM layer.

#### 2.1.1 Sending

The *SingleSenderC* component provides the *AMSend* interface as *Send* for sending packets. This is the same interface provided by *AMSenderC* and is used in the same manner.

#### 2.1.2 Receiving

The *SingleReceiverC* and *SingleSnooperC* components provide the *Receive* interface for receiving packets. This is same interface provided by *AMReceiverC* and *AMSnooperC* and is used in the same manner. *SingleReceiverC* provides only those packets whose multihop destination is the current node, whereas *SingleSnooperC* provides any packets whose multihop destination is not the current node.

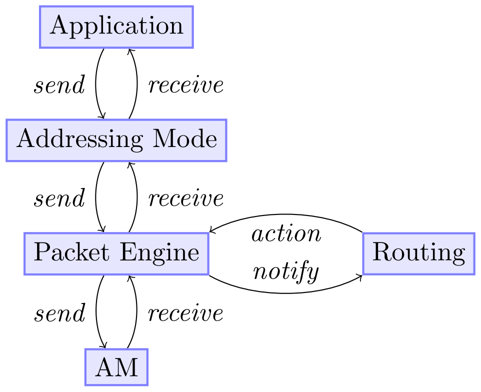

Figure 1: Simplified view of internal components

#### 2.1.3 Packet Manipulation

All of the above *Single*C* components also provide a new interface, *SinglePacket*, for reading and writing the multihop source and destination of a given packet. This is especially helpful if you are buffering packets, since you may no longer be within the context of the *send* command or *receive* event, where these fields are simpler to access.

The AM layer packet interface, *AMPacket*, is also provided, mainly for the *address* command. The AM layer fields in a packet should generally not be modified at the application layer, nor should they need to be in any case.

## 3 Framework Internals

This section is primarily of interest to those who would like to extend the framework itself or adapt additional routing protocols to work with it. If you are only interested in making use of multihop communication in your applications, then you should be able to do so with just the information from Section 2.

The various internal components of the framework can divided into a number of groups. Figure 1 shows a simple diagram of how these groups fit together. As usual, more complete diagrams are available within the source code documentation. Each group is described below, along with the interactions between the basic framework and available routing protocols. This discussion uses single node addressing and DSR as an example, so a basic knowledge of the operation of DSR may be helpful prior to reading this section.

### 3.1 Addressing Mode

Each addressing mode (single node, geographical, etc.) has its own set of sending, receiving, and processing components. While this does result in a large number of components in the end, it allows application developers to clearly specify the way they intend to address their multihop packets. Also, most of the addressing modes differ in the data types used to specify an address, thus necessitating slightly different interfaces for each.

The sending and receiving components for single node addressing have already been mentioned in Section 3. These *Send* and *Receive* interfaces are used to move packets between the application and the addressing mode components. There are also additional lightweight components that implement read and write operations on the packet fields that are specific to the particular addressing mode in use. A packet's multihop source would be one example of such a field.

## 3.2 Packet Engine

Moving down the chain from the addressing mode brings us to *PacketEngineC*, a generalized "packet engine". I borrowed this term from a similar component in the TYMO (DYMO for TinyOS) [7] implementation. Essentially it's a fancy name for an intelligent packet queue or buffer. This centralizes packet storage and management in one location and simplifies routing protocol development because these pieces do not need to reinvented each time.

All packets coming down from the addressing mode layer and up from the AM layer are stored in a buffer. If there is a routing protocol (such as DSR) associated with the type of packet, then the protocol's component is notified of the new packet and also the method through which it entered the buffer. No immediate response to this notification is needed. If there is no routing protocol for a given packet type, then the packet is dropped from the buffer.

Once the routing component has been notified that a given packet has entered the buffer, it is free to reply with any of several actions to for the buffer take on that packet, such as sending (down to the AM layer), send with interception (application layer can block or modify packet), receiving (up to the application layer), and silently dropping. Both the send and send with interception actions are queued since they must first wait for the radio to be free. There is also a short, randomized delay between successive transmissions to reduce the probability of collisions.

The notification events and action commands are all contained within the interface *PacketEngine*. This is the main mode of communication between a routing component and the rest of the framework. Since the notification/action strategy used by this buffer does not force routing components to make decisions immediately, they are free to do a reasonable amount of processing as long as the buffer does not fill up.

## 3.3 Routing

In this framework, there are typically two kinds of triggers that will initiate some kind of processing in routing components:

- notification from the buffer of a new packet or

- expiration of a timer.

For example, DSR makes use of the notification of a packet waiting to be sent to search for a route to the packet's final destination in its route cache. If no route is found, then route discovery is initiated in an attempt to find one by sending out route request packets. Timers are also involved here as well. If no route has been found after a set amount of time, then the route discovery process is initiated again. Of course, this is a simplified description of only one portion of the overall protocol, which involves many other triggers like these.

While at first it may seem like a better idea to tightly integrate the buffer with the routing component, separating the two actually simplified our implementation of DSR by allowing us to focus on the core of the routing algorithm without being bogged down by tedious buffer management in the same component. Also, the routing component is free to modify packet headers and other information before send a command back to the buffer. Thus, nearly all of the complex processing that one would have done in an integrated design can be done within this framework as well.

## References

[1] R. Wagner, M. Duarte, J. R. Stinnett, T. S. E. Ng, D. B. Johnson, and R. Baraniuk, "A network API-driven survey of communication requirements of distributed data processing algorithms for sensor networks," Rice University, Tech. Rep., 2006. [Online]. Available: http://www.ece.rice.edu/~rwagner/IPSN-API-survey.pdf

[2] R. Wagner, J. R. Stinnett, M. Duarte, R. Baraniuk, D. B. Johnson, and T. S. E. Ng, "A Network Application Programming Interface for Data Processing in Sensor Networks," Rice University, Tech. Rep. TREE0705, Jan. 2007. [Online]. Available: http://www.dsp.rice.edu/~rwagner/docs/wagnerTREE0705.pdf

[3] The COMPASS Multihop Framework for TinyOS. [Online]. Available: http://www.owlnet.rice.edu/~jryans/framework/

[4] D. Johnson, Y. Hu, and D. Maltz, *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*, IETF RFC 4728, Feb. 2007.

[5] The COMPASS Multihop Framework for TinyOS: Source Code Documentation. [Online]. Available: http://www.owlnet.rice.edu/~jryans/framework/nesdoc/

[6] P. Levis. Packet Protocols. [Online]. Available: http://www.tinyos.net/tinyos-2.x/doc/html/tep116.html

[7] R. Thouvenin. TYMO: An implementation of the DYMO protocol on TinyOS. [Online]. Available: http://tymo.sourceforge.net/