CSE 466 Exam II 16 March 2006

1. TinyOS Terms (30 points)

Define the following terms related to features of the TinyOS operating system and provide a basic definition as well as an example of when each feature would be used. Relate your examples to the Flock project, if possible.

a) Asynchronous code

Asynchronous code is code is invoked from the triggering of an interrupt handler. What makes is "asynchronous" is that the timing of its execution is not under the control of the run-time system. In the Flock project, asynchronous code services interrupts from the Yamaha sound generation chip to eventually refill its FIFO.

b) Posting a task

A task is a long-running computation that is posted onto a FIFO queue for later execution – it is too long to be executed asynchronously. A task is an example of "synchronous" code. In the Flock project, a task was posted after the FIFO interrupt from the Yamaha sound chip so that the FIFO queue could be refilled at leisure.

c) Active messages

Active messages are packets of data that contain a pointer to the code of the handler to be executed when they are received. This allows TinyOS to automatically dispatch messages to the appropriate handler. There were about 6 different active message types used in the Flock project from everything to setting global variables to turning on an LED on a particular mote.

d) "HPL" designated files

HPL files are the lowest level components of the TinyOS abstraction stack and directly interact with microcontroller control registers. When porting TinyOS to a new platform the HPL files have to be modified for the new hardware. Similarly, when connecting a new sensor or peripheral device, a corresponding HPL component must be created. In the Flock, such a component was the Yamaha sound chip.

e) Components and interfaces

TinyOS applications are composed of hierarchical components that connect through interfaces. Commands and events flow across interfaces from one component to another. In the Flock project, several components were used to compose the application with at least one interface needing to be defined.

f) Parameterized interface

A parameterized interface allows the virtualization of an interface into multiple instances. In the Flock project, parameterized interfaces were used to create multiple timers for different components to use (even though there was only one timer) and to create different handlers for active messages.

2. Flock Improvement (30 points)

During the flock concert, it became clear that we needed a more effective way of specifying which birds should perform a command. Our current method specified a range of nodeIDs. The protocol required the node to check whether the node's ID was within the specified range, before performing the command it received. If not, it ignored the packet. However, the nodes we wanted to control were not in a contiguous range making the process of selecting a set of nodes difficult and we ended up sending individual messages to each node.

a) Suggest a way that we could easily assign a node to a set that we could later refer to in the command packets. Describe the communication between the node and controller during this process and what would be the key payload elements of each packet sent between them.

The first question is to decide whether the number of sets can be fixed or variable. If it is fixed, then we could refer to sets using a bit-vector (where a bit is assigned to each possible set). This would work for a reasonable number of sets for flock concerts, say, 8 to 16 (one or two bytes). If it is variable, then we may want to refer to only one set at a time which will mean multiple messages if we want two different sets to do the same thing. The alternative is listing multiple sets in a packet, but this will make the packets have variable length. Let's choose the bit vector approach using only 1 byte. Therefore, we can only have 8 sets.

Here is the communications that happen during the identification process:

1. *Bird is identified – in our case, by the appropriate light/shadow pattern on the light sensor.*
2. *The bird sends a packet to the controller identifying itself.*
3. *Controller sends back a command packet to that bird setting its membership to a particular set (e.g., set 4 would mean the set-membership byte would be 00001000).*

b) Describe how you would go about setting up three different sets, then get two of these sets to execute the same command, and the third to do a different one, and, finally, get each of the three sets of birds to sing a different song. Consider a total of 6 birds, 2 in each set to start but then discuss how your approach scales to larger flocks (is the number of packets linear in the number of birds, linear in the number of songs we want to get them to sing, etc.).

1. *Identify the first 2 birds.*
2. *Send them a set-membership message to join set 1 (e.g., 00000001).*
3. *Identify the next 2 birds.*
4. *Send them a set-membership message to join set 2 (e.g., 00000010).*
5. *Identify the next 2 birds.*
6. *Send them a set-membership message to join set 3 (e.g., 00000100).*
7. *To get the birds in sets 1 and 2 to execute the same command, we can add a new byte to the command packet payload that states which sets should execute the command. In this case, for sets 1 and 2 we would send a command with the following set-selection byte: 00000011.*
8. *Send a command to set 3 to play a different song using a set-selection byte of 00000100.*
9. *To get the birds in each set to sing a new different song, send a command for each set.*

This approach requires at least one packet to each bird to set its membership in up to 8 sets (linear in the number of birds). If the birds happen to have contiguous ID numbers, then this number could be reduced further. The number of commands to sing songs is linear in the number of different songs we want the birds to sing as any number of sets (up to 8) can be commanded to sing together.

<u>3. Radio Protocols (40 points)</u>

In the Flock project, AdjustGlobals packet were broadcast from the controller (node 0) to all the birds. The assumption was that all birds would be within radio range. Let’s consider how to handle a much larger flock where many nodes are not in range of the controller. Clearly, a bird receiving an AdjustGlobals packet may have to rebroadcast it.

Discuss the following aspects of this problem:

a) How would you forward the AdjustGlobals packet? Would you change anything in the packet payload?

To make sure that AdjustGlobals packets are not forwarded indefinitely we can do two things:

1. *add a sequence number to the AdjustGlobals packets so that a node can tell if this is an outdated command that is still being forwarded, and*
2. *add a hop count to the packet to limit how many times it is forwarded.*

An algorithm to handle AdjustGlobals would be:

```
if (packet.seqNo > previousSeqNo) {
   process packet;
   previousSeqNo = packet.SeqNo;
   packet.hopCount – – ;
   if (packet.hopCount > 0) forward packet;
}
```

b) How would you ensure that the propagation of the packet will eventually end? Do you need to add to or change the payload of the packet to help with this issue?

With a sequence number, a packet will propagate until every node has seen it at least once (and has set its previousSeqNo to that value).

Using hop count alone is more problematic as we have to estimate the number of hops that may be possible in the flock.

c) Does your approach guarantee that each node will eventually receive the packet (assuming there is at least one path connecting it to the controller)? How much waste is there in your scheme (that is, how many times might a node hear the same AdjustGlobals packet get repeated)?

Nodes that are within range of another will definitely received the packet. In the worst case, of a fully connected network, a node will hear the same AdjustGlobals packet from every other node as each forwards it for the first time. Therefore the same AdjustGlobals packet will be transmitted as many times as there are birds in the flock. Note that this would also be the case in a degenerate network where one node can only one other.