Homework 2, CSE 461 Fall 2009

Instructor: Wetherall, Out: 10/14, Due: 10/28

Part A – Design your own RFID protocol

The goal of this part of the homework is for you to design an RFID inventory protocol. You will code up your protocol as a program and run it in an emulated environment to see how well it performs. *We recommend that you do this part with a partner but require at most two people per team and a different partner from earlier assignments. Ask Josh for help if you cannot find a partner.*

RFID Scenario

The RFID systems we explore contain an RFID reader and multiple tags. The reader is in charge in the sense that it initiates all communications and the tags can only run when commanded by the reader. RFID tags are small computational devices (literally tags!) that have a unique identifier called an EPC (Electronic Product Code). Since they have no batteries, the reader must supply power for them to run, which is why they only take action when commanded by the reader.

The rules of communication in this system are as follows. The reader broadcasts messages that may be heard by all tags, and provides the tags with power so that they may send an immediate reply back to the reader. Tags can only hear the reader; they cannot hear if there are replies by other tags. If multiple tags transmit then there is a collision at the reader and none of the messages can be successfully received, but the reader knows there was an error. Since this is wireless, any message from the reader or tags may be received in error independently at each receiver (whether the reader or tags) due to noise. These errors are indistinguishable from a collision, but the receiver knows there was an error.

The goal of the RFID reader is to perform an "inventory", in which it gathers the identifiers of the set of RFID tags that are in range. The reader has no a priori information on the number of tags in range nor their identifiers. Its goal is to complete the inventory as quickly as possible – to find all of the tags in the shortest amount of time.

Our Strawman Protocol

We have provided as part of the code a strawman protocol to perform the inventory that runs at both reader and tags. You can run it as:

```
java RFIDSim <#trials> <#tags per trial> <% channel error rate>
```

The number of tags is the key parameter you will be varying as you experiment. You can set the other parameters to reasonable numbers and explore. 1% is a small error rate to remind you that messages do get lost sometimes; 10% is a better value to stress test that your protocol works when there are errors; if you find all tags at 0% error but not 10% error then your protocol may be flawed. The number of trials should be enough that the variation in the average is small. 100 is likely to be ample, and a smaller number of trials will probably suffice if running time is getting large.

The strawman protocol works as follows. Tags start in an uninventoried state. The reader repeatedly sends out query messages. If an uninventoried tag gets a query, it picks a random

number in the range 0 to 31, and only if the number is 0 does it reply with its EPC. If the reader gets an EPC, it adds it to its list and sends an ack. If a tag that just sent an EPC gets an ack as its next message then it sets its inventoried flag and remains quiet. If the reader hears no replies or errors in 32 consecutive queries then it decides it has heard from all the tags and returns.

The simulator then prints out the key evaluation parameters which are the number of bytes sent/received over the channel and average tags missed. The number of bytes is a measure of simulated time. This is your performance metric -- lower is better. The average tags missed is a check that the protocol is finding all of the tags most of the time; if it is more than an occasional missed tag then your protocol is probably broken. Other statistics printed are simply for your information, and you might add more.

As you can imagine, our strawman is not very good. For very few tags, say 2, there will often be no reply to a query, wasting time. For many tags, say 100, there will often be collisions in reply to query, again wasting time. It is also the case that our strawman can easily miss tags by returning too early.

Your RFID Protocol

Your job is to design an RFID inventory protocol that delivers good performance consistently over a range of numbers of tags. It will be (much) better than the strawman. The protocol will include the messages sent between the reader and tags and vice versa as well as what happens at the reader and at the tags in response to these messages and to generate the messages in the first place. You should code up your protocol in both RFIDReader and RFIDTag, replacing the strawman protocol. *We strongly advise you to develop your protocol in stages, testing that each stage you add works (and presumably improves performance) rather than coding and testing performance at the end.*

You are free to design your protocol as you like but should act plausibly by at least observing these rules, which are intended as a reasonable model of the scenario:

1. The reader has no a priori knowledge of the number of tags or their identifiers. You cannot assume the identifiers are sequential.
2. The reader and tags can communicate with each other only via the channel. This means that your tag cannot access any reader state directly or vice versa. You must not change RFIDChannel.
3. Tags cannot communicate directly with each other via the channel (or by accessing state). They can only exchange messages with the reader.
4. Tags can perform computation and keep a modest amount of state.
5. Tags can only reply with messages in response to a reader command; they cannot set timers to wake up later and take action.

As you design your protocol, you should draw on ideas we have discussed in class. The closest example we have covered is Ethernet for the multi-access aspects of the design. The situation bears some resemblance to N hosts each having one message to send (their identifier), but it is also different in that everything is driven by the reader. You will also have to think about reliability and errors.

You are also welcome to read up on RFID tags, including the EPC Class 1 Gen 2 RFID specification (see link on the Homework page) but realize that 1) you do not have to implement the standard; and 2) there is much detail you can read about RFID that is superfluous for this

exercise. [Sportsfans, also note that you may read the RFID assignment from previous versions of 461, but take care to realize that the scenario and design space are both different.]

Questions and Turn-in

1. Turn in you the code for your protocol. You need include only the relevant portions in the reader and tags (no GUIs, or general support code).

2. Turn in a description of how your protocol works at a high-level that points out the important features and *why* you designed it the way you did. One page maximum.

3. Give us a graph to show how well your design works versus the strawman. For each design, plot the average number of bytes sent/received and average missed tags versus the number of tags, where the number of tags ideally ranges from 1 to 128. Stop with fewer tags if the computational (hence simulated) time becomes too large. (You will want to generate these graphs as you experiment with your protocol, not simply at the end, to see how well it is doing.)

4. Compare your protocol with the strawman. Specifically, how is your design better than the strawman and why? Compare your protocol with the unrealistic theoretical minimum in which the tags simply send their EPCs in sequence. Specifically, how far is your protocol from this theoretical minimum and do the gaps seem fundamental or easy to fix?

5. Suppose that when two or more tags reply the reader does not always get a collision. If one tag has a stronger signal than the other that tag may be received, with no indication that other tags sent a message. This is called capture, and it is a more realistic physical model in some cases. How does capture affect your design problem? Speculate as to the impact it would have on missed tags (reliability) and bytes sent/received (performance).

Part B – Problems

1. P&D 2.49. (I suggest you look up "Birthday Attack" on Wikipedia or elsewhere and use the common exponential approximation.)

2. P&D 4.6

3. Following on from the above, consider two alternatives to the "send twice" models of part a) and part b). The first alternative called "c)" is to send an 11^{th} "parity fragment." The parity fragment is the modulo sum of all the 10 data fragments. By the properties of parity, one lost fragment can be reconstructed – it is simply the sum of all the other fragments. (Try an example for yourself.) The second alternative called "d)" is to send smaller packets, each with the same loss probability as a fragment, and retransmitted the packets that were lost. The questions are as follows:
    a. What is the probability of net loss of the packet with c)?
    b. What is the expected number of "small packets" you need to send to get 10 packets of data (equal to one "large packet") through?
    c. Which scheme or schemes do you argue are best and why?

4. P&D 4.15 and 4.17. Read the appropriate text sections for examples tables of the algorithms in action.