Homework 4, CSE 461 Fall 2009

Instructor: Wetherall, Out: 11/18, Due: 12/2

Part A – Photo Sharing

In this part of the homework, you will collectively design, implement and run a photo sharing application. You will develop your protocol designs with the rest of the class starting in section, since this is an application will run across the class. You will code your designs with your partner and run your application at likely locations (section, class, the lab, etc.) so that it can automatically find and share photos. The goal of the exercise is to give you experience with 1) socket programming and network applications; 2) issues of interoperability, specification and robustness; 3) ideas in social networking, disruption tolerant networks and content-based naming on which the application draws. *We recommend that you do this part with a partner. You can work with any of your classmates.*

Photo Sharing Application

The photo sharing application should work as follows at a high-level. It should maintain a pool of photos on disk that you can look at, and you should be able to add photos to the pool locally. The photos should be in the JPEG file format and have an associated short text description. The app should be a program that you run and can leave running on your computer. It should look for nearby instances of the application, e.g., running in the same classroom or coffee shop. When it finds another instance, it should exchange photos, adding any new photos that it gets to the local pool. Eventually, everyone should have all photos as you go about your regular day.

The user interface is up to you and will not be standardized; it is the network protocols you will standardize. A simple interface would be to add files to the local pool by copying them to the right place in the right format with the right name, and to view files with a Web browser that opens the directory in which they are stored. It would also be straightforward to have your program generate an index page that shows all the photos and descriptions on one page. You would then point your browser at that page. It would look better. There are many other possibilities too.

Note that your application explores ideas around a few “hot topics” as follows:

Social networking. This is a social network application for the class. Information flows over a path defined by social contacts, in this case multiple people who are in the same place at the same time. You will have to work out how instances of the application find each other when they happen to be “nearby”.

Disruption Tolerant Networking (DTNs). There is no central server, and there is no time at which the whole class is necessarily connected, i.e., the connectivity is disrupted. Nonetheless, the photos should get to everyone in the class eventually as long as information propagates when there are connectivity opportunities.

Content-based naming. You should identify photos by their content so that everyone gets all the different photos; there was nothing written above about the names of photos like “beach.jpg”. Read about cryptographic hashes (message digests) to see how you can construct short names that reflect the content to make it easy find different pictures.

Photo Sharing Protocol

Here is a sketch to get you started. This protocol is missing all the details, which you will fill in as part of the assignment.

Photo objects. A photo in the system must have the following attributes (and it may have more depending on your design). The photo itself, in JPEG file format. A short text description, e.g., "Josh and Brent go skiing". And a movement trace, e.g., "Created on Josh-MAC on 17 Nov 2009; Moved Josh-MAC to Brent-PC on 18 Nov 2009". The first two attributes are fixed on creation. The last attribute gets updated as apps move photos (or, better, learn where photos have moved). It will help you see how the app is moving photos around.

Photo transfer. Your app must support at least two operations to exchange photos efficiently. Do this on top of TCP. The first is INDEX. It is used by one application instance to query another instance and find out what photos it has. The second is GET. It is used by one instance to retrieve a photo object from another instance. These two operations should be sufficient, but you can add more features if you like.

Application Discovery. Your app must support at least one operation for finding nearby instances. It is DISCOVER. It is used by one application instance to broadcast a local query to which other instances respond. Once two instances find each other, transfer can take place. Send messages on top of UDP to do this.

Your assignment

Your job is to work as a class to complete the design, implement the application with your partner, and run it to share photos. We will work on the design in section. Each section should come up with their preferred design in the form of "RFCs". (The class can implement a single design if you can all agree, or we can have splinter factions if there is a group of at least 10 people who want to implement a different design. But please start with a default of one design per section.)

Josh will help to facilitate the design by leading discussion in section. However, this assignment asks you, collectively, to fill it the details – we will provide feedback but we do not want to simply tell you what to do, as working through the problem is much of the learning and fun.

Each section (or group) will produce a short specification in the form of "RFCs". Pick a handful of people to write notes to capture the current state of the design. Keep it short, likely a page or two. We will provide a way to put these up (maybe a discussion board) so that each group can see the RFCs and continue to define and improve them.

Your section should produce RFCs as follows:

1. Give details for each of the categories above (photo objects, photo transfer, and application discovery).
2. For each operation, describe the parameters with which it is invoked (e.g., the content name of a photo to be retrieved) and the messages it causes to be exchanged across the network between instances (e.g., a request and a response?).
3. Give the formats for each message or other information that is exchanged.
4. Describe how these messages are sent using the services of TCP or UDP.

Keep in mind that these designs are only about the network portion of the application. They should be kept short and be used for interoperability. They should not specify details that are up to implementers, like how often to try to discover nearby instances, or which order to get files. There will likely be multiple versions as you fill in or change details, so you should consider what will happen when the version changes. Another tip is that formatting of messages is often the source of much complexity. Prefer simple formats that are easy to implement in Java and do not worry about bandwidth efficiency.

You and your partner should implement the application and run it in section and class to share photos. You should use sockets. A good starting exercise that you can do immediately is to write code to transfer a photo in a file from one application to another application that then writes it to disk. We also suggest that you implement discovery last. You can manually point your instance at another instance to test photo transfer without getting discovery right. Another tip is that a source of complexity in distributed applications is that the program does not fail all at once. Your application may be fine, but the implementation it is talking to may crash halfway through. You want to be robust to this sort of problem so that problems elsewhere don't cause you to fail. The fewer assumptions you make about what another implementation will do, the safer you are.

After your group has the basic functionality specified and working, you may want to extend your application. You could define a way to exchange and merge traces even when you already have the photo so that nodes learn more about where photos have moved. You might add other types of data to the system, e.g., the name of the content creator so that you can show photos grouped by the creator. You might provide a way to delete photos, though this is hard: to provide decent security only the creator of a photo should be able to delete it. These extra features should be optional so that everyone in the group does not need to implement them.

Questions and Turn-in

Question 1: Turn in your photo transfer and application discovery logic (as code). Assume we have read the specification on the website and describe any implementation choices for how your code works that are not in the specification.

Question 2: Turn in the traces of the photos you have that tell us how they moved through the system. (We're curious!) Tell us how many other application instances you connected to and tell us the longest movement trace you found.

Question 3: Give one example of a decision that is significant from the point of view of how well the application works but which is not (and should not be) part of the protocol specifications.

Question 4: P2P networks have widely been used to share content. Content owners have sometimes put "decoys" in the system to block illegal transfers, e.g., a movie called "Harry Potter" that really contains an anti-piracy message. How does your application prevent decoys? (If it doesn't, then it should!)

Question 5: The Internet Robustness principle in RFC 791 says "Be conservative in what you do; be liberal in what you accept from others". It makes protocols less fragile. Give one example of how you followed this principle in your implementation. (You might learn this when you *don't* interoperate with someone else when you both think you have it right.)

Part B – Problems

1. P&D 5.54.
2. P&D 6.4, plus sketch the “power” curve. (See 6.1.3)
3. P&D 6.10
4. P&D 6.7 (See 6.1.3)
5. P&D 9.2