Fluid Networking for the Activist

Nick Knouf

December 6, 2007

This project straddles two disciplinary boundaries: that of computer science and science and technology studies. On the one hand, part of the work is the active questioning and critiquing of modern network theory, especially as it applies to humans. These ideas suggest ways in which the abstracting nature of nodes and edges tends to smooth away messy questions of politics and ethics. However, once we begin to look at networks with a political lens, we can see where more nuanced views of network structure and representation provide openings for positive responses. The second part of the work, then, is the development of a mobile phone application for activists that relies on fluid interpretations of networks for its main effect. The entire project can be seen as a case study of how integration of these disparate areas can suggest novel technical undertakings while remaining true to conceptual concerns.

CRITIQUES AND REINTERPRETATIONS OF NETWORK VIEWS OF REALITY

I went into some depth in my reaction paper regarding the questions that have been raised regarding network views of reality, so I will not repeat all of those arguments here. Briefly, the critiques that are most important to this project can be summarized in three points:

- **Who is represented in the network?** When we decide to make explicit representations of network structures that ultimately refer to people, we are making political choices about who and what to represent. This can have severe consequences when we are not aware of attempts (conscious or unconscious) to marginalize people or groups through these representations.

- **Is the network taken to be a metaphor, or the real thing?** Do we use talk of networks as a way to describe complicated aspects of reality, as in actor-network theory from science and technology studies, or do we assume the world is structured in a network fashion, as much work in computer science seems to do? Choosing the later makes a strong ontological commitment that might be very difficult to stick to once we being to understand the messy and fluid nature of reality.

- **How can we talk about suffering within network views of reality?** This is a specific critique but quite relevant to this work. Thomas Berker, in combining my concerns, writes, "How can suffering and conflict be described in a meaningful way within a relational ontology" (Berker 2007, p. 182)? When we see things as nodes and edges, we tend to remove the specific qualities of lived experience from our description. Opening ourselves to that understanding might suggest different ways to conceptualize networks, but it might have to do away with structure and rigidity but provide us with ways to recognize and respond to suffering.

Taken together, the critiques suggest that if we see the networks of people as fluid, and unable to be represented within a rigid framework, we can be more responsive to the idiosyncrasies of lived life. Additionally, once we see people as more than nodes, and thus unable to be represented entirely, we turn away from attempting this impossible task and investigate rather how we can use the actions of people for political purposes. And finally, when we question who is represented by networks, and take a somewhat literal interpretation of the question of "suffering", we can identify potential users who might otherwise not be considered.

All of these reinterpretations suggest the present project, a fluid networking system that enables activists in countries where there is severe repression to use Bluetooth enabled phones to pass messages surreptitiously. This work pulls heavily from recent work in *pocket-switched networking*, which I briefly review next.

POCKET-SWITCHED NETWORKING

In general, *delay-tolerant networking* (DTN) refers to protocols that do not assume that data transmission can without severe latency. This can be due to a number of causes, most often because of structural or physical limitations, as in interplanetary internet (Burleigh, Hooke, Torgerson, Fall, Cerf, Durst, Scott, and Weiss 2003). Unlike in traditional IP networking protocols, where smaller and smaller latencies are the goal, DTN removes this as a criteria.

Related to this is the idea of *pocket-switched networking* a play on traditional IP *packet-switched networking*. Pocket-switched networks assume highly mobile nodes that may or

may not have connectivity to outside networks like the Internet, and assume that data might be best passed hop-to-hop, versus end-to-end (Hui, Chaintreau, Scott, Gass, Crowcroft, and Diot 2005; Chaintreau, Hui, Crowcroft, Diot, Gass, and Scott 2005). This work builds on prior considerations of mobile devices as data mules (Shah, Roy, Jain, and Brunette 2003) or message ferries (Zhao, Ammar, and Zegura 2004). Much work has gone into collecting real-world data for the study of pocket-switched networks, most often based on simple traces of the temporal appearance and disappearance of Bluetooth devices over a given time frame, leading to characteristic distributions of nodes. This work has lead to the creation of a new networking architecture called Haggle that supports mobile users (Scott, Hui, Crowcroft, and Diot 2006; Su, Scott, Hui, Upton, Lim, Diot, Crowcroft, Goel, and Lara 2007; Su, Scott, Hui, Crowcroft, Lara, Diot, Goel, Lim, and Upton 2007).

One of the most difficult questions in pocket-switched networks is the choice of forwarding algorithm. Since all data transmissions are based on hop-to-hop transmissions, and there is no way to know the network topology in advance, a decision has to made as to how and when to transmit data from one node to another. The most obvious solution is *epidemic forwarding*, where all messages that are currently stored at one node are forwarded to every other node that is seen. This has obvious problems of network congestion and high storage requirements, but still tends to perform quite well, at least in simulation (Hui and Crowcroft 2007*a*). A number of other forwarding algorithms have been suggesting, many using prior knowledge of community affiliation (Hui and Crowcroft 2007*b*; Hui and Crowcroft 2007*a*) or being based on models of chemical diffusion (Hui, Leguay, Crowcroft, Scott, Friedmani, and Conan 2006).

The main problem with these studies of different algorithms, at least when taken in the context of my project, is that they are all based on simulation and do not take into account the challenges of networking over Bluetooth in real-life environments. The design of the networking protocol described below tries to take this into account, suggesting a way in which we can do better than epidemic forwarding while also acknowledging the challenges of Bluetooth.

SELBAY

Selbay[1] is the name for my software system that attempts to both actualize some responses to the concerns I described earlier, while also suggesting a novel forwarding protocol that addresses and uses to its advantage some of the physical limitations of Bluetooth networking that would otherwise be lost within a simulation.

[1] The name is meaningless, other than my attempt to use a combination of phonemes that are perhaps known within many languages

USING STIGMERGY IN THE DESIGN OF THE FORWARDING PROTOCOL

We saw in the previous section how mass-forwarding of messages, while the most obvious way to approach the problem, is not necessarily the best given the limitations of our underlying network protocol. To understand this, we need to understand how we communicate over Bluetooth connections. Communication is a three step process:

1. **Device Discovery**: a device scans the local neighborhood, finding other devices that are advertising their presence over Bluetooth

2. **Service discovery**: once a particular device is found, this step queries the device for available services

3. **Message transmission**: once the relevant service has been found, the device transmits data or a message via the provided service. This is also the step where we would query whether or not a particular message *should* be sent.

Each step in this process can take anywhere from 10 to 60 seconds or more. Given that Bluetooth devices, at least when carried by humans, are likely to be highly mobile, any time we can eliminate from these steps increases the chance we will be successful in transmitting our message. If we could even eliminate parts of one of the steps we would be in better shape. To figure out which part of the chain we can cut out, I have to make a slight detour into ideas from swarm intelligence which has interesting parallels with the current problem.

It has long been an interest of researchers to better understand how groups of agents, each with simple abilities, can self-organize into a very complicated systems. In biological systems these types of behaviors often rely quite strongly on the leaving of perceptible markers of recent behavior. For example, ants will leave pheromones on the trail of food; as more ants follow one particular path, there will be more pheromones left, and thus later ants will follow the stronger scent. This is an example of *active stigmergy*. In general, *stigmergy* is where "individual behavior modifies the environment, which in turn modifies the behavior of other individuals" (Bonabeau, Dorigo, and Theraulaz 1999, p. 16). On the other hand, *passive stigmergy* is when agents modify parts of the environment in line with their normal behavior, but this modification sends signals to other agents to change their behavior. An example of this is the building of structures in certain shapes that cause other agents to choose one path over another.

Let's return to the earlier discussion of epidemic forwarding protocols. Many of the refinements to these transmission protocols relied on knowing some prior information about community affiliation (Hui and Crowcroft 2007*b*), "concentration" of files (Hui, Leguay, Crowcroft, Scott, Friedmani, and Conan 2006), or hierarchical rankings of communities

(Hui and Crowcroft 2007*a*). Each approach requires the imposition of some underlying structure on the network that, in our case, may not be there. Additionally, and perhaps more worryingly, is the fact that any lookup of data requires that we go through all three steps of the process described earlier: in order to get information about "concentration" of files, or to determine whether or not we should send a file, requires that we find the device (step one), find the right service (step two), and successfully transmit and receive data about server holdings (step three) before we can even transmit our desired message.

Yet if we use ideas from stigmergy, we can eliminate the third step entirely. To see how, we have to go into depth how we advertise services using Bluetooth.

Listings 1 and 2 show how we use Bluetooth service advertising as a means of stigmergy. This is the second step of the three-step data transmission chain described previously. The lists of services were made from commands entered at a Python prompt[2]. Each entry in the list is a tuple of three elements: Bluetooth address, service port, and service name. We can see in Listing 1 the Selbay service is active at port number 4. This is the service and port that clients would connect to in order to send data to the server. More interesting, for our purposes of exploring stigmergy, is the service at port number 5. This service does not have any methods that listen for connections; rather, the service is there to advertise the hash[3] of a recent message that was accepted by the server[4]. Instead of having to query the "Selbay" service at port number 5 in order to determine whether or not to transfer a particular message, a client can merely query for the list of services, comparing the discovered hashes with local ones and streamlining the data transmission process.

Listing 1: List of services before acceptance of new item

```
>>> findservices(psont)
 [('00:02:EE:6B:86:09', None, u'SDP Server'),
  ('00:02:EE:6B:86:09', 1, u'Hands-Free Audio Gateway'),
  ('00:02:EE:6B:86:09', 10, u'OBEX File Transfer'),
  ('00:02:EE:6B:86:09', 11, u'SyncMLClient'),
  ('00:02:EE:6B:86:09', 12, u'Nokia OBEX PC Suite Services'),
  ('00:02:EE:6B:86:09', 9, u'OBEX Object Push'),
  ('00:02:EE:6B:86:09', 2, u'Dial-Up Networking'),
  ('00:02:EE:6B:86:09', 3, u'AppleAgent'),
```

[2] All Bluetooth code, on the phone and for testing on the laptop, used the open-source lightblue (`http://lightblue.sourceforge.net/`) library

[3] The hash is the md5 sum of the title and data of a message; this gives a unique fingerprint of any message we transmit or receive

[4] The hash has an additional ":" given as a prefix in order to enable quick programmatic differentiation of this service from the other advertised services.

```
 ('00:02:EE:6B:86:09', 4, u'Selbay'),
 ('00:02:EE:6B:86:09', 5, u':f51ffdfc71bc943e9ac1d6515b3e7c31')]
```

This is a form of active stigmergy, where the server explicitly leaves traces of previous interactions for other clients (mobile phones) that might interact with the server. Because of the multi-step process of the Bluetooth protocol, and the fact that devices are going to be in constant motion, using this active leaving of traces of previous interactions at the second step of the process cuts down on unnecessary traffic and hopefully will lead to more data transmissions occurring. Using stigmergy we have eliminated the third step entirely when deciding whether or not to transmit a message.

Listing 1 shows another example of stigmergy, after we have successfully transmitting a different message to the same server. We see that two hashes are now advertised. If a new client communicated with this server, the client would know not to send messages with those hashes, if the client were in possession of them.

Listing 2: List of services after acceptance of new item

```
>>> findservices(psont)
[('00:02:EE:6B:86:09', None, u'SDP Server'),
 ('00:02:EE:6B:86:09', 1, u'Hands-Free Audio Gateway'),
 ('00:02:EE:6B:86:09', 10, u'OBEX File Transfer'),
 ('00:02:EE:6B:86:09', 11, u'SyncMLClient'),
 ('00:02:EE:6B:86:09', 12, u'Nokia OBEX PC Suite Services'),
 ('00:02:EE:6B:86:09', 9, u'OBEX Object Push'),
 ('00:02:EE:6B:86:09', 2, u'Dial-Up Networking'),
 ('00:02:EE:6B:86:09', 3, u'AppleAgent'),
 ('00:02:EE:6B:86:09', 4, u'Selbay'),
 ('00:02:EE:6B:86:09', 5, u':063ab2111a1c5499ab4fc128bcdb490e'),
 ('00:02:EE:6B:86:09', 6, u':f51ffdfc71bc943e9ac1d6515b3e7c31')]
```

We can now describe the networking protocol in detail for both the server and the client.

SERVER PROTOCOL

1. On startup, advertise hashes that are have been stored in a special table called `SelbayStigmergy`

2. Loop:

    a) Accept connections from clients

b) Add new data to the database and to the list in the UI

c) Add the hash of the new data to the `SelbayStigmergy` database and advertise the hash

As far as I can tell from the Bluetooth Specification (SIG 2007), there is no limit as to the number of services that any one device can advertise. Whether advertising services that do not accept connections is a violation of the protocol or not is something I still need to determine. But this re-appropriation of service advertisement should significantly cut down on the amount of time to transmit a large number of messages.

CLIENT PROTOCOL

1. Loop:

    a) Search for available devices

    b) For each available device

        i. Get list of services available on device

        ii. Use discovered hashes as a blacklist of hashes to not send

        iii. Until device is not in range anymore

            A. Send items from the `SelbayOutgoing` database

            B. Send items from the `SelbayData` database

The client protocol actively uses the provided stigmergy of the server to create a "blacklist" of messages hashes to *not* send, thereby improving on epidemic forwarding algorithms, as well as cutting down the amount of time that would be needed to find out this information in standard ways.

SELBAY SYSTEM DESIGN

The system design for Selbay is given in Figure 1. There are three software components. A UI program, described in detail below, that enables browsing and editing of messages that are to be sent or of messages that have been received. The server program, described above, listens and responds to incoming connections, as well as advertises recent hashes for stigmergetic reasons. The client program, also described above, connects to server programs and sends messages, avoiding those with hashes that match those advertised by the server, and choosing first messages from the `SelbayOutgoing` database, and then in sequence from the `SelbayData` database, starting with the most recent ones.

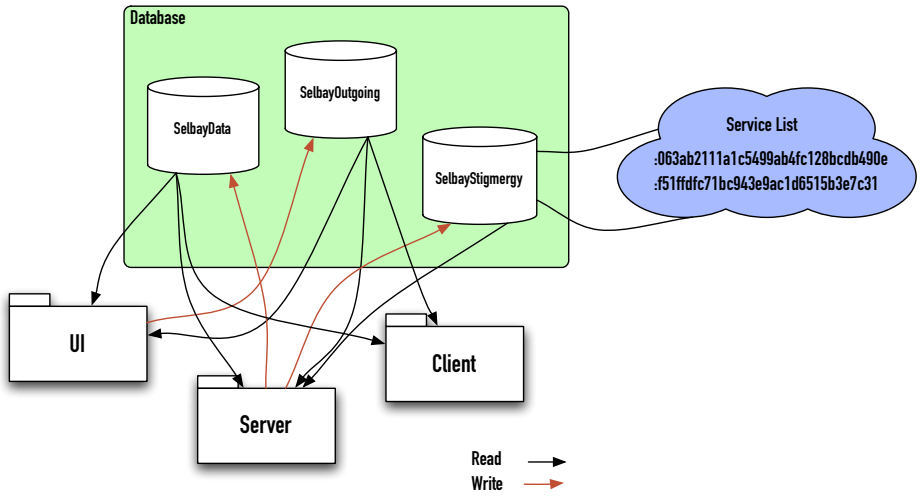


Figure 1: Selbay system design, with connections between the software components and the database

Each software component speaks to a database[5], chosen for simple persistent storage. The diagram details the types of read and write connections each component makes to the database. The `SelbayStigmergy` database provides the list of hashes that are to be advertised by the server and that provide a trace of recent interactions with the software.

(non-)simulation of present protocol

In my proposal for this project I had suggested that I would write a multi-agent simulation of the propagation protocol, trying to take into account some of the idiosyncrasies of the Bluetooth protocol. In the end, I decided that this was not worth the effort of trying to simulate all of these parameters, and here I would like to go further into this decision.

Much of the literature on message forwarding algorithms is, at one level, based on empirical data, but on another level, completely divorced from it. Specifically, empirical data has been extensively collected on the time distribution of the appearance and disappearance of nodes. Using *post-hoc* analyses, many have developed various types of forwarding algorithms that take into account the dynamics of this empirical data (Hui, Leguay, Crowcroft, Scott, Friedmani, and Conan 2006; Hui and Crowcroft 2007*a*; Hui and Crowcroft 2007*b*). Importantly, however, the testing of these algorithms happens in simulation; to date, I have

[5] This is implemented using the built-in `e32db` module of Python Series 60. The module uses the built-in Symbian database facilities that offer a minimal subset of standard SQL.

been unable to find published work that tests these algorithms within the messy realm of real-life, where data transmission is not instantaneous, where Bluetooth device discovery takes longer than expected, where a device might only be within range for less than a minute. Some of these variables could have been implemented within a simulation but would have been based on little more than wild guesses and thus had little correspondence to reality. While Hui, Leguay, Crowcroft, Scott, Friedmani, and Conan (2006) go into some detail as to their choice of network simulation parameters, there is no test like putting the device and algorithm in the wild.

My concern in this work is not to better understand node distributions within mobile, ad-hoc networks, nor is it to mathematically model these networks. Rather, I am most interested in developing a network application that can be used by activists for surreptitious information transmission. This goal requires does need *some* of the results of the networking literature discussed so far, but it does not need to replicate it. In the end, the application will succeed based on its applicability to real-life situations, not that of a simulation. Time is better spent developing a robust program that is likely to hold up well when thrown into the wild, with an iterative design process improving on the program structure through repeated real-life tests.

With that said, I have not quite made it to the point of having a releasable application that can be tested by others; however, I should make it to that point shortly. Additionally, the lack of an IRB-approved human-subjects protocol would have prevented me from testing this on people at this stage. Within the next month I should be able to begin looking at how the application works when used by others, as well as opening up the source code for other developers to use and modify.

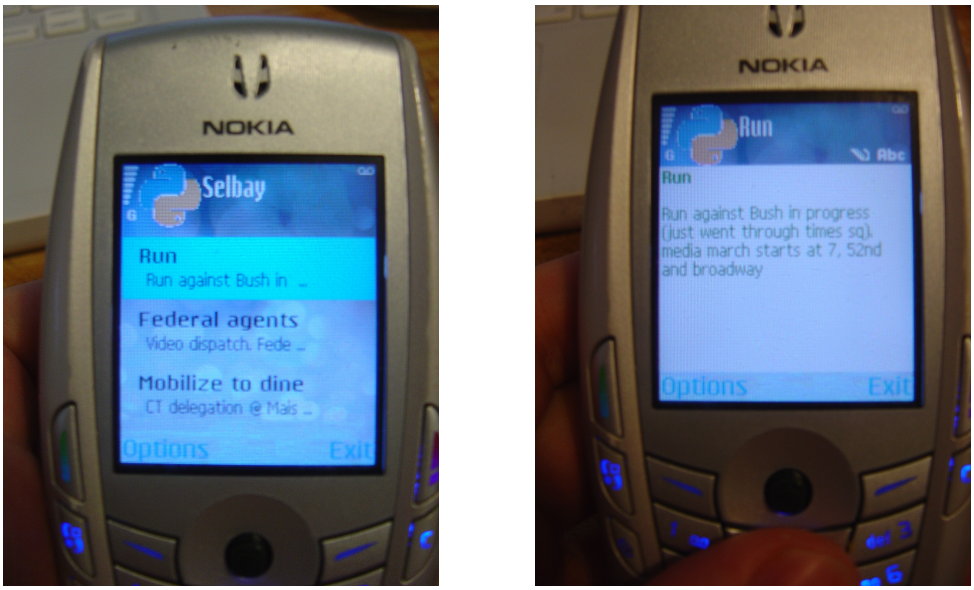
Additionally, while I have tested the protocol with two to three devices, it is still an open question as to whether or not it will scale. Yet this depends on the factors that I just described, so it too will have to wait until the (near) future.

Selbay UI Design

Given that Selbay is going to be used by a variety of different end users, the user interface design for browsing and sending data should be straightforward and easy-to-use. At the moment I cannot claim to have tested the UI on a large number of potential users, but I want to present the ideas as they now stand.

(non-)interface to server and client

Importantly I made the decision to not provide an interface to the server and client parts of the software. While these components are vital to the functioning of Selbay, they are not



(a) Message list (opening screen)

(b) Message content of selected message

Figure 2: Selbay UI for browsing existing messages

important to the end-user. He or she is able to inspect them if desired, but the front-facing software should hide the implementation details. Thus Selbay starts the client and server parts as separate processes upon program start. Periodically the main loop checks to make sure the processes are still running and restarts them if necessary.

At the moment the client and server processes do not run independently of the UI, meaning that when the UI program exits, the client and server processes do as well. This is partly due to the limitations of the Series 60 Python implementation, which does not provide any hooks for registering programs to be run upon phone boot. Later versions of the software will work to overcome this problem, providing menu options for installing the client and server programs to run independently of the UI.

BROWSING CURRENT MESSAGES

On starting Selbay the user first sees the screen for browsing through the list of current messages, shown in Figure 2(a). The list shows all of the messages, sorted by time with most recent first, that are currently in the database, providing a title and a short snippet of text. Selecting a message will open up a new screen that provides the complete text of the message, shown in Figure 2(b). Pressing the right button under the screen returns you to

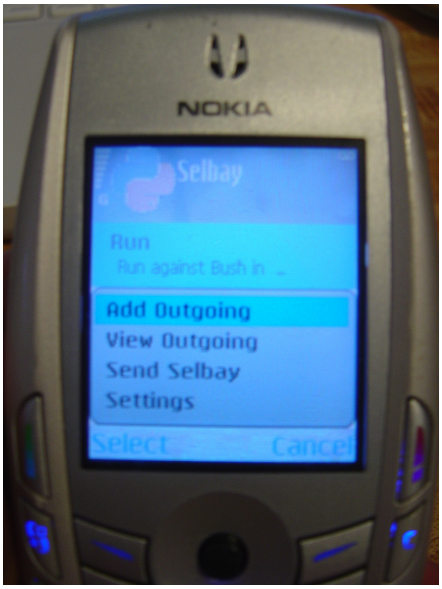


Figure 3: Selbay menu items

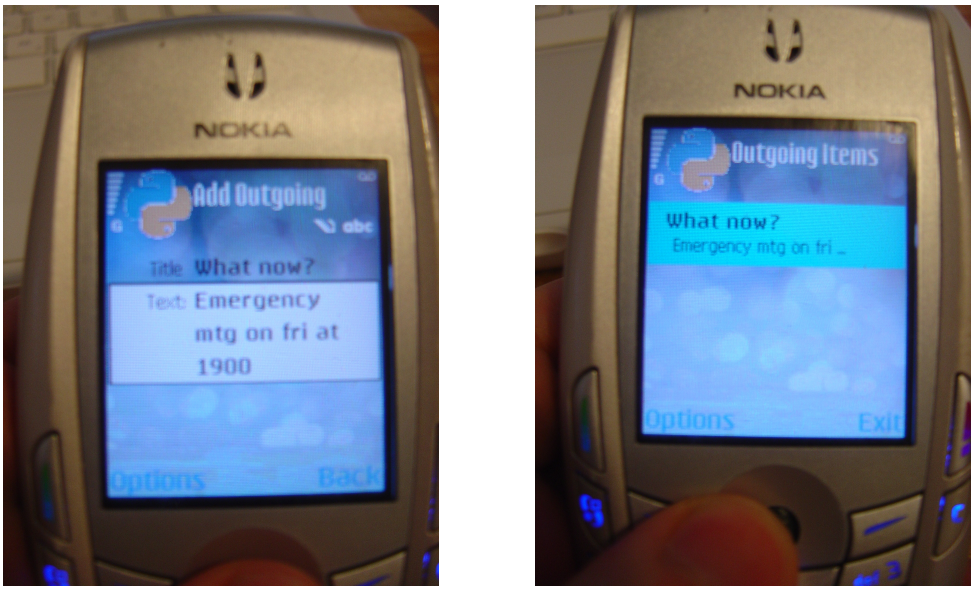
the list of messages in the opening screen.

SELBAY MENU OPTIONS

The opening screen offers a number of menu options, as shown in Figure 3. The first two options, "Add Outgoing" and "View Outgoing" will be described in the next section. The third option is "Send Selbay", enabling the user to send the application to any other user accessible over Bluetooth. This option is important to the dissemination of Selbay; given that its use is most likely in locales where access to external networked sites (such as the Internet) might be intermittent or unavailable, having a local source for the software is vitally important. The "Send Selbay" option turns any user of the software into a distribution point and eliminates the need for external network connectivity.

CREATING MESSAGES FOR DISTRIBUTION

Users of the software need the ability to input messages for sending to other activists; these options are provided by the "Add Outgoing" and "View Outgoing" options on the main menu screen. Selecting the "Add Outgoing" option opens up a form for the input of a "Title" and "Text", as shown in Figure 4(a). Exiting the form causes the message to be saved in the `SelbayOutgoing` database; as described earlier, the client transfers messages queued in this



(a) Form for entering a new message

(b) Viewing the added message

Figure 4: Selbay UI for entering and viewing added messages

database first. Added messages can be viewed in a list similar to what was already described (Figure 4(b)), with the possibility of edit and deletion as well.

CONCLUSION

In this project I have described Selbay, a mobile phone application that is in its early stages of providing ad-hoc, fluid networking for activists. While I have not been able to extensively test the networking protocol I developed, it at least appears that it will perform better than epidemic forwarding as it re-appropriates Bluetooth service advertising to eliminate one step of data transmission. I hope that in the near future the software will be stable enough to be tested by activists in a number of different scenarios.

REFERENCES

Berker, Thomas (2007). "Networks and Suffering". In: *New Network Theory*. 182–191.

Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York, NY, USA: Oxford University Press.

Burleigh, S., A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss (2003). "Delay-tolerant networking: an approach to interplanetary Internet". In: *Communications Magazine, IEEE* 41.6. 128–136.

Chaintreau, Augustin, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott (2005). Pocket Switched Networks: Real-world mobility and its consequences for opportunistic forwarding. UCAM-CL-TR-617. Tech. rep. University of Cambridge.

Hui, Pan, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot (2005). "Pocket switched networks and human mobility in conference environments". In: *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. Philadelphia, Pennsylvania, USA: ACM. ISBN 1-59593-026-4. DOI: http://doi.acm.org/10.1145/1080139.1080142. 244–251.

Hui, Pan, and Jon Crowcroft (2007*a*). Bubble Rap: Forwarding in small world DTNs in ever decreasing circles. UCAM-CL-TR-684. Tech. rep. University of Cambridge.

— (2007*b*). "How Small Labels Create Big Improvements". In: *Proeedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07*. 65–70.

Hui, Pan, J. Leguay, J. Crowcroft, J. Scott, T. Friedmani, and V. Conan (2006). "Osmosis in Pocket Switched Networks". In: 1–6.

Scott, James, Pan Hui, Jon Crowcroft, and Christophe Diot (2006). "Haggle: A Networking Architecture Designed Around Mobile Users". In: *Proceedings of IFIP WONS 2006*.

Shah, R. C., S. Roy, S. Jain, and W. Brunette (2003). "Data MULEs: modeling a three-tier architecture for sparse sensor networks". In: *Proceedings of the First IEEE Sensor Network Protocols and Applications, 2003*. 30–41.

SIG, Bluetooth (2007). *Specification of the Bluetooth System*. Bluetooth SIG.

Su, Jing, James Scott, Pan Hui, Jon Crowcroft, Eyal de Lara, Christophe Diot, Ashvin Goel, Meng How Lim, and Eben Upton (2007). "Haggle: Seamless Networking for Mobile Applications". In: *Proceedings of the Ninth International Conference on Ubiquitous Computing (UbiComp 2007)*.

Su, Jing, James Scott, Pan Hui, Eben Upton, Meng How Lim, Christophe Diot, Jon Crowcroft, Ashvin Goel, and Eyal de Lara (2007). Haggle: Clean-slate networking for mobile devices. UCAM-CL-TR-680. Tech. rep. University of Cambridge.

Zhao, W., M. Ammar, and E. Zegura (2004). "A message ferrying approach for data delivery in sparse mobile ad hoc networks". In: *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*. Roppongi Hills, Tokyo, Japan: ACM. ISBN 1-58113-849-0. DOI: http://doi.acm.org/10.1145/989459.989483. 187–198.