Privacy-enhanced social-network routing

Iain Parris[1,*], Tristan Henderson[1]

School of Computer Science
University of St Andrews
St Andrews, Fife KY16 9SX, UK

Abstract

Opportunistic networking — forwarding messages in a disconnected mobile ad hoc network via any encountered nodes — offers a new mechanism for exploiting the mobile devices that many users now carry. Forwarding messages in such a network often involves the use of social-network routing— sending messages via nodes in the sender or recipient's friends list. Simple social-network routing, however, may broadcast these friends lists, which introduces privacy concerns.

This paper presents a threat analysis of the privacy risks in social-network routing. We introduce two complementary methods for enhancing privacy in social-network routing by obfuscating the friends lists used to inform routing decisions. We evaluate these methods using three real-world datasets, and find that it is possible to obfuscate the friends lists without leading to a significant decrease in routing performance, as measured by delivery cost, delay and ratio. We quantify

[*]Corresponding author.
Email addresses: `ip@cs.st-andrews.ac.uk` (Iain Parris), `tristan@cs.st-andrews.ac.uk` (Tristan Henderson)
URL: `http://www.cs.st-andrews.ac.uk/~ip/` (Iain Parris), `http://www.cs.st-andrews.ac.uk/~tristan/` (Tristan Henderson)
[1]Phone: +44 1334 463253
Fax: +44 1334 463278

the increase in security provided by this obfuscation, with reference to classes of attack which are mitigated.
Keywords: social networks, opportunistic networks, privacy, social-network routing
2010 MSC: 68M10, 91D30

1. Introduction

Mobile devices, such as mobile phones, are commonly carried around by people during their daily lives. While most current communication using such devices takes place through infrastructure such as licensed GSM or UMTS networks, it may be possible to exploit these devices in an ad hoc manner. Messages may be directly exchanged between devices when they are in physical proximity to each other. In this way, an *opportunistic network* may be formed; where people send messages to each other via intermediary devices utilising a disconnected store-and-forward architecture.

One main challenge in opportunistic networks is routing: given episodic connectivity based on people's real-world movements, how can we send messages from source to destination? One approach is *epidemic routing* — flooding the network with messages, by sending messages during each and every encounter. This approach indeed ensures that, if a path exists between source and destination, the message will certainly find and follow this path to be delivered as quickly as possible. But sending large numbers of redundant messages, as epidemic routing is apt to do, is wasteful, and will drain the batteries of the mobile devices rapidly.

To reduce the cost of message delivery, messages should be selectively forwarded during encounters between members of the opportunistic network. An ideal routing algorithm would eliminate redundant messages, while still forwarding all other messages. In practice, there is no oracle with global knowledge of all current and future states of the network: people may move freely, and encounters are to some degree random. The routing problem, then, is: given only local knowledge, what is a good method of determining whether or not a message should be forwarded during an encounter?

One possible method for forwarding is *social-network routing*. Making the underlying assumption that encounters between mobile devices are more likely to occur within groups of people who are connected to each other, for instance through friendship or co-location, than between random strangers, messages may be "source-routed" — forwarded selectively only between "friends" of the original sender.

But one oft-overlooked problem with social-network routing is that of privacy. In social-network routing schemes, intermediate nodes forward messages based on whether the encountered node is in the original message sender's friends list. This may involve broadcasting the social network information — this friends list— in the clear; the information cannot be encrypted end-to-end, even in the presence of a public key infrastructure, because it is utilised by intermediate nodes to inform their routing decisions.

There exist potential privacy concerns when broadcasting friends lists. For example, users of the opportunistic network might have an embarrassing friend about whom they do not wish the world to know. Or a user may be happy with some of their friends list information being used to inform routing decisions, but not with their whole set of friends being world-viewable: it is one thing for a curious person to be able to infer some of the friends based on forwarded messages, but quite another to distribute the potentially-sensitive information freely. Such privacy concerns are not just theoretical. For example, social network links (in this case, frequent email contacts) were broadcast by default during the release of Google Buzz in February 2010, prompting user backlash.

We wish to mitigate these privacy concerns, while still retaining the advantages of social-network routing.

In this paper we:

- Analyse the potential privacy threats implicit in social-network routing, to present an attack tree.
- Investigate the effect on routing performance of obfuscating the social network information used for social-network routing, using three real-world datasets.
- Investigate hiding social network information using one-way hashing, via the probabilistic Bloom filter data structure.
- Quantify the social-network routing security increase obtained when applying the levels of friends list obfuscation which our performance experiments indicate are practically-achievable, with reference to classes of attack which are mitigated.

Our contributions are to provide one of the first analyses of threats in social-network routing, and the first schemes to attempt to enhance privacy in social-network routing without the use of key management. We augment our previous paper [1] with extended evaluations using additional datasets, and we discuss linkability and eavesdropping attacks against our schemes. The paper is outlined as follows. In the next section we introduce the concept of an opportunistic network. In Section 3 we analyse the possible privacy threats in these networks, followed by a discussion of our two social-network routing schemes in Section 4. Section 5 presents an evaluation of these schemes using two real-world traces. Section 6 discusses the security of our schemes. Section 7 discusses related work, and finally in Section 8 we conclude and discuss ongoing work.

2. Opportunistic networks

Opportunistic networks [2] have become increasingly popular and relevant as more people carry mobile devices. Essentially, an opportunistic network is a disconnected MANET (mobile ad hoc network) where mobile nodes can send messages in the absence of any knowledge about network topology. Nodes opportunistically make use of any other nodes that they encounter, as long as these encountered nodes are likely to help the message reach its destination. For instance, users carrying mobile phones can send messages via other mobile phone users that they meet. Such networks can be used to create new applications, such as social media or information dissemination [3], even in the absence of existing infrastructure.

The efficiency and performance of an opportunistic network critically depends on accurately determining which encountered nodes will be useful in forwarding. Early opportunistic network routing protocols, such as epidemic [4] or PRoPHET [5] routing, used network or mobility characteristics. But one fruitful method of improving forwarding is to use social network information. If we know that a node is a friend of the message sender or of the intended message recipient, then it may make sense to use that node for forwarding. Many forwarding schemes have been proposed that leverage social network analysis in this way. The Bubble Rap [6] scheme analyses encounter patterns to detect communities of likely encounters, and uses these communities to decide to which encountered nodes data should be forward. The SimBetTS [7] and SimBetAge [8] schemes use further metrics from social network analysis, namely centrality and betweenness, to dynamically develop utility functions for each encountered node to aid forwarding decisions. Habit [9] combines social network and location information to build an information-dissemination system for mobile devices. The PeopleRank [10] and SRSN schemes [11] take an alternative approach: instead of determining social network information from encounter patterns, they use pre-existing social ties (e.g., from social network sites [SNSes] such as Facebook).

In this paper, we refer to this class of opportunistic network forwarding protocols which leverage social network information as *social-network routing,* and further, the class of protocols which broadcast social network information in the clear as *simple social-network routing.* Following [12], we refer to two people connected through social network links as *friends*, and the complete set of such friends of a particular person as a *friends list*. Thus, when sending a message using social-network routing, the message is forwarded only between members of the original message sender's friends list. Also, in suitable contexts, we may refer to people as being *nodes* inside a social network — the edges in the network being the social links of a particular person (node), and the linked-to people (nodes) in turn are the original node's friends.

As motivated in Section 1, our focus is on enhancing social-network routing privacy.

3. Threat analysis

Before we can attempt to enhance privacy in social-network routing, it is necessary to understand the possible threats against privacy that may occur in opportunistic networks using such routing schemes.

We consider an attacker with certain, limited capabilities. From the attacker models enumerated in [13] against opportunistic networks, our interest is in the *local eavesdropper* (an attacker who can eavesdrop in the vicinity of a user), and the *partial eavesdropper* (an attacker who can place receivers in a number of hotspots and intercept traffic in the vicinity). We agree that a *global eavesdropper* is not a practical attack model in an opportunistic network — by the very nature of such a network, nodes are distributed over a very large area, and traffic is not routed through any central hub.

Therefore we do not consider attacks which would require global knowledge, such as an attacker studying overall network traffic patterns. We enumerate attacks based on intercepting some number of messages.

We choose to employ *attack trees*, as introduced by Schenier [14]. An attack tree is a type of *and-or tree*, used to enumerate attacks against a system. The root node of the tree is the overall attack goal, while nodes within the tree are subgoals. The children of a particular node are the steps required to achieve that node's subgoal. By constructing such a tree from the root node (overall goal) downwards, we now enumerate a structured threat analysis for attacks against an opportunistic network using social-network routing.

3.1. Goal: Discover structural information about the social network graph.

1. Learn whether a friendship link exists (or does not exist) between two users.
   OR

   (a) Discover communication (or lack of) between the users. OR
      i. Eavesdrop a message as it is forwarded user-to-user, from source to final destination (or any intermediary). OR
         A. In simple social-network routing, a message traced along such a path reveals social network links (or lack of) – because messages are forwarded if and only friendship links exist. Friendship links are the path traversed by the message.
      ii. Extract source/destination from an intercepted message to an intermediary.
   (b) Extract friendship links from an intercepted message to an intermediary.
2. Learn how many friendship links a particular user has.
   (a) Extract friendship links from an intercepted message to an intermediary.

3.2. Goal: Discover whether two individuals have been in proximity within a certain timeframe.

1. Follow one or both individuals for the time in question. OR
2. Infer proximity by sending a specially crafted message, and making inferences based on where the message is observed within the network. OR
   (a) Example: has Alice from New York recently met Bob from Los Angeles? To find out, an attacker Mallory in New York can inject a message addressed for colluding attacker Trudy in Los Angeles into the system, with Alice and Bob only as requested intermediaries. If Trudy receives Mallory's message, Mallory and Trudy have learned that Alice and Bob have met within the lifetime of this malicious message.

3. Infer proximity by noting that messages are not forwarded twice. OR
    (a) Example: if a message is not forwarded to a node known to be a requested intermediary, the message must already have been forwarded earlier. An attacker can infer that the nodes were in proximity before this time. This is a passive version of 2, not requiring message injection.
4. Wait in a common place and listen for message traffic. Message exchange, or message headers, may reveal the colocation of individuals to an attacker.

3.3. Goal: De-anonymise a social network to discover the presence of individuals within the network.

1. Follow individuals, and tie their network identifiers to their actual identities. OR
2. Infer identities from known portions of the social network.
    (a) Example: if five people are known to be mutual friends, and four are deanonymised with a fifth mysterious node, an attacker can infer that this unknown node is the last member of the clique.

4. Privacy-enhanced social-network routing

In simple social-network routing schemes, the sender's friends list is transmitted in the clear along with each message. Intermediate forwarding nodes are able to read the sender's full friends list in plain text, facilitating most of the threats outlined in Section 3.

Encrypting the friends list end-to-end can ensure privacy, but we would then lose the advantages of social-network routing: intermediate forwarding nodes would no longer be able to exploit the sender's social network information to inform their routing decisions.

If friends lists could be encrypted using pair-wise keys shared between each pair of nodes, then at least an eavesdropper could not overhear the sensitive data. However, this has two problems. Firstly, we assume lack of a public key infrastructure (PKI) in opportunistic networks — due to the nature of such networks, we regard building a PKI as extremely difficult at best, and arguably impossible. Secondly, and more fundamentally, the sender may not wish to broadcast their social network information to all of their contacts — which would necessarily occur for this information to be used by these contacts as intermediaries for routing.

Inspired by [15], we attempt to target the privacy threats introduced by social-network routing by modifying and obfuscating each sender's friends list, on a per-message basis, at message generation time. By modifying the friends lists, we aim to introduce plausible deniability; each list transmitted is no longer a true copy of the friends list. By obfuscating the friends lists, we aim to make it more difficult for a person with a copy of a particular friends list to read out its contents. In this way, from our threat analysis in Section 3, we are specifically targeting threats 1b and 2a — recovering social network information from intercepted friends lists— which we regard as the most relevant privacy threats.[2] We now introduce two schemes for doing so.

[2] We discuss why we regard these as the most relevant threats in Section 6 — along with quantifying to what extent the threats are mitigated.

4.1. Statisticulated Social Network Routing

Named for a portmanteau of *statistical manipulation*[3], our first scheme is *Statisticulated Social Network Routing* (*SSNR*).

For each message transmitted, the sender makes changes to the message's copy of their friends list— adding or removing nodes. While the friends list sent along with the message will be based to some extent on the sender's true friends list, and so still useful for social-network routing, the friends list has been modified by the addition or removal of nodes. Any node seeing the friends list sent along with the message now cannot say with certainty whether a particular node is truly one of the sender's friends, or truly not one of the sender's friends.

The sender may in practice choose the level of manipulation of the friends list on a per-message basis. In our evaluation, however, we examine routing performance for a particular choice of modification degree of the sender's friends list. For instance, we may choose a $+50\%$ modification of the friends list. This notation signifies that the sender adds 50% more nodes to their friends list before message transmission. We thus determine average performance for a particular degree of friends list modification. For simplicity, we do not evaluate routing performance while simultaneously adding and removing nodes; only for either adding or removing nodes.

It would still be possible for a malicious person to average over the friends

[3] Huff coins the term *statisticulation* in his book *How to lie with statistics* [16], where he writes:

> "Misinforming people by the use of statistical material might be called statistical manipulation; in a word (though not a very good one), statisticulation."

lists included with many messages of one particular sender. But we have created much more work for this malicious person: many generated messages must be intercepted, rather than just one single message to reveal all. We quantify and discuss the nature of the improvement in security in detail in Section 6.

4.2. Obfuscated Social Network Routing

In our second scheme, *Obfuscated Social Network Routing* (*OSNR*), instead of transmitting the sender's friends list as a list of nodes, we embed the friends list within a Bloom filter.

A Bloom filter [17] is a probabilistic data structure which allows probabilistic querying for set membership. False negatives are not possible, but false positives are — with increasing probability as the Bloom filter becomes more full. After inserting each node in the sender's friends list into a Bloom filter, we may regard the Bloom filter itself as a non-trivially-reversible hash of the friends list.

To make a rainbow table attack[4] impractical, we create a per-message random salt, which is sent along with the message in the clear. The elements inserted into the Bloom filter are a concatenation of this random salt with a unique node identifier (any unique node identifier would suffice, e.g., a lower layer construct such as MAC address, Bluetooth address, IMEI, or some higher level identifier tied to the user rather than the device). In our evaluation, we choose to use Bluetooth addresses as the identifier.

Given the Bloom filter and the random salt (transmitted in the clear with the message) and an encountered node's identifier, it is easy to make a routing decision: query for set membership of the random salt concatenated with the candidate

[4] A rainbow table is a precomputed lookup table of hash value to hash input [18].

node identifier. A positive result — guaranteed if the candidate node is inside the sender's friends list, but also possible with low probability if not — means to forward the message, since the encountered node is most likely in the original sender's friends list. A negative result means that the candidate node is not in the sender's friends list, and so not to forward the message.

Since we are not using encryption (we assumed no PKI), it still may be possible for an attacker to reverse engineer the Bloom filter by brute force — the attacker can iterate through all the node identifiers, concatenating each with the plain-text salt and testing for a Bloom filter match. This is orders of magnitude more work than a rainbow table lookup, however, and the brute force step must be repeated for every message. So using the Bloom filter (with salt) does not provide perfect security, but does make the attacker's job very much more difficult. We elaborate and quantify this attack further in Section 6.

It is possible to combine *OSNR* and *SSNR*: the friends list may be modified as in *SSNR* prior to hashing the social network information in a Bloom filter as in *OSNR*. In our evaluation, we refer to this combined scheme as *SSNR-OSNR*.

We note that Bloom filters are fixed-width — a convenient property for scalability. In pure *SSNR*, packet headers may grow arbitrarily large as the sender's friends list grows; this is potentially a problem for a sender with a very large social network (and compounded if the social network is grown further using *SSNR*). *OSNR*, and *SSNR-OSNR*, have no such scaling problem due to the fixed size of the Bloom filter.

5. Performance evaluation

We now present an evaluation of our two schemes to determine their impact on opportunistic network performance. We use trace-driven simulation with three real-world datasets.

5.1. Datasets

We chose three real-world datasets to evaluate our routing schemes. While there are a variety of available datasets including encounter and social-network information, the three datasets used were chosen for their different scale and structure.

5.1.1. St Andrews mobile sensor network

We collected the first dataset from a deployed sensor network system, the St Andrews Sensing SYstem (SASSY), in a previous experiment [11]. We hereafter refer to this dataset as the *SASSY* dataset. 25 participants were equipped with 802.15.4 Tmote Invent sensor motes and encounters were tracked for a period of 79 days, although for efficiency we chose to use only the first 30-day section of this trace for our simulations.

The original dataset was very sparse due to hardware limitations which meant that many encounters may have been missed. Inspired by [19] and [20], we augment the collected traces using a working-day and augmented random-waypoint model. Nodes randomly select a waypoint from a set of points of interest and walk according to predetermined paths (such as roads) to reach these points. Nodes moved at 0.5–1.5ms^{-1}. At each waypoint the nodes could stop for 0–120s. Each node was additionally randomly assigned a home location, and the nodes would travel to this location to "sleep" for 8 hours in every 24. Each node had an additional 10% probability of either visiting the Computer Science departmental buildings (since our participants were mainly Computer Science students) or their "home" at any waypoint selection.

To obtain social network information for the *SASSY* dataset, we use the self-reported social network information provided by the 25 participants at the start of the experiment: their Facebook "friends". Many participants knew each other: the mean friends list size (i.e., number of Facebook friends also participating in the experiment) was 9.8, with a standard deviation of 5.0.

5.1.2. MIT Reality Mining

The second dataset used was the well-known *Reality Mining* dataset [21] collected at MIT [22]. This dataset comprises Bluetooth encounter traces from 97 mobile phone users over the course of an academic year. To obtain social network information for this dataset, we use the participants' address book information — if a pair of nodes encounter one another, and at least one has the other in their address book, then the pair of nodes is said to be friends; each node has the other in their friends list. Unlike the *SASSY* dataset, few participants knew each other: 52 participants had at least two other participants in their friends lists (and were thus candidate nodes for social-network routing in our simulations). Of these 52 participants, the mean friends list size is 3.7, with a standard deviation of 2.0.

Because of differing lengths of participation in the experiment[5], we could not treat the dataset as one contiguous trace — it would not be meaningful to simulate

[5]Some participants carried mobile phones for the full nine months of data collection, while others participated for much lower amounts of time — as low as one month.

message-passing between people no longer participating in the study. Therefore, at the beginning of each simulation run, we draw out a random[6] 30-day segment of the trace.

5.1.3. NUS student contacts

The third dataset used is a student contact pattern dataset comprising the class schedules of 22 341 students at the National University of Singapore [23] (hereafter referred to as the *NUS* dataset).

The dataset describes contacts between students in recurring weekly sessions. As in the original paper describing the dataset [24], we regard an encounter between two students as occurring when these students share a session — that is, when the students are in physical proximity in the same classroom. We regard two students as having one another as social contacts, so having one another in each of their respective friends lists, if they share at least one session in the week. Defining social links in this way, the mean friends list size is 561, with a standard deviation of 396.

Due to memory constraints, we do not perform simulations using the full *NUS* dataset: the full dataset contains 12.3M encounters and 6.2M social network links. Instead, we sample a subset of students from the full dataset in two different ways, to derive two new, smaller datasets. In each case, after extracting a subset of the students, we preserve all encounters and social network links between students

[6] The only constraint placed on the selection of the random 30-day segment which we draw out is that at least three participants, each with at least two other participants in their friends list, must be carrying phones throughout the 30 day period — otherwise meaningful social-network routing could not occur (since there would be no message-passing intermediaries).

within this subset.

1. We randomly select 500 students from the full 22 341 students in the *NUS* dataset. We call this derived dataset *NUS-R*. Results from this dataset may reflect a real-world opportunistic network deployment, where only a proportion of students participate in the opportunistic network.
2. A downside of randomly selecting students from the full *NUS* dataset is that doing so leads to a relatively sparse social graph. Therefore, we adopt the approach of Liu and Wu [25] to sample the *NUS* dataset in a second way, which avoids the extremes of sparsity (as occurs when randomly sampling students) or over-connectedness in the new, derived, size-reduced dataset. We select the first student randomly, and then, to select the k^{th} student, we randomly divide the previously-selected $k-1$ students into two equal-sized groups S_1 and S_2, and select the k^{th} student as that student with the highest $\sum_{s_1 \in S_1} sim(s, s_1) - \sum_{s_2 \in S_2} sim(s, s_2)$ where sim is the number of common class sessions in which two students are enrolled. Using Liu and Wu's approach, we sample 500 students from the full *NUS* dataset. We call this derived dataset *NUS-L*.

For *NUS-R*, the mean friends list size is 12.0, with a standard deviation of 8.54. For *NUS-L*, the mean friends list size is 119, with a standard deviation of 39.9. The wide difference in mean friends list size (while the absolute number of nodes is the same at 500) is an expected outcome from the different natures of the two *NUS* sampling methods, and provides an initial illustration of the differing properties of the derived datasets.

Dataset	Number of nodes		Clustering coefficient	Social links	Encounters
	Total	≥1 edge			
SASSY	25	25	0.771	254	29 909
Reality Mining	97	75	0.318	107	32 359
NUS	22 341	22 340	0.536	6 261 458	12 320 946
NUS-L	500	500	0.634	29 743	71 819
NUS-R	500	476	0.506	3 001	6 109

Table 1: Dataset statistics. All fields refer to properties of the social network, except for the number of encounters.

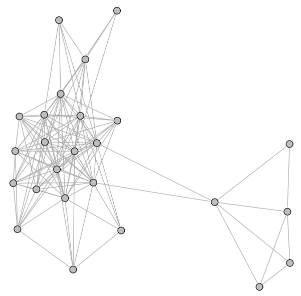
5.1.4. Dataset statistics

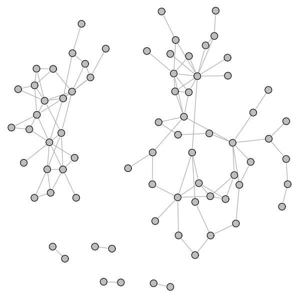
Table 1 provides an overall description of the datasets used for evaluation. It can be observed that we have a chosen a variety of datasets which vary in size, timescale and density. This is borne out by visualisations of the various social networks within these datasets (Figure 1) and the degree distributions of these social networks (Figure 2), which indicate variety in the network structures. Thus we are confident that these datasets provide suitable test cases for the performance of our proposed schemes.

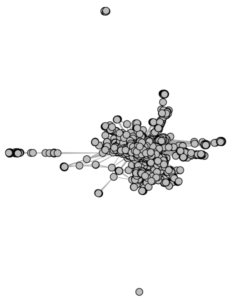
5.2. Simulation parameters

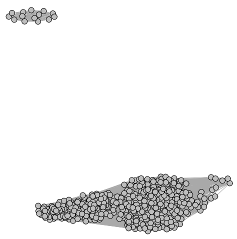
We performed trace-driven simulations using these datasets with the following parameters:

- 900 messages generated per simulation. Each message was unicast from a random sender to a random recipient from that sender's contacts list.


(a) *SASSY* dataset.


(b) *Reality Mining* dataset.


(c) *NUS* dataset.


(d) *NUS-L* dataset.

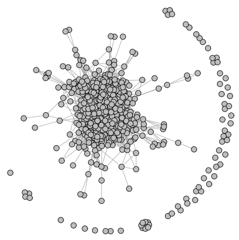

(e) *NUS-R* dataset.

Figure 1: The social networks from the three datasets used for evaluation. The *NUS* dataset, Figure 1(c), was sampled in two different ways to derive two different datasets — Figure 1(d) and Figure 1(e).

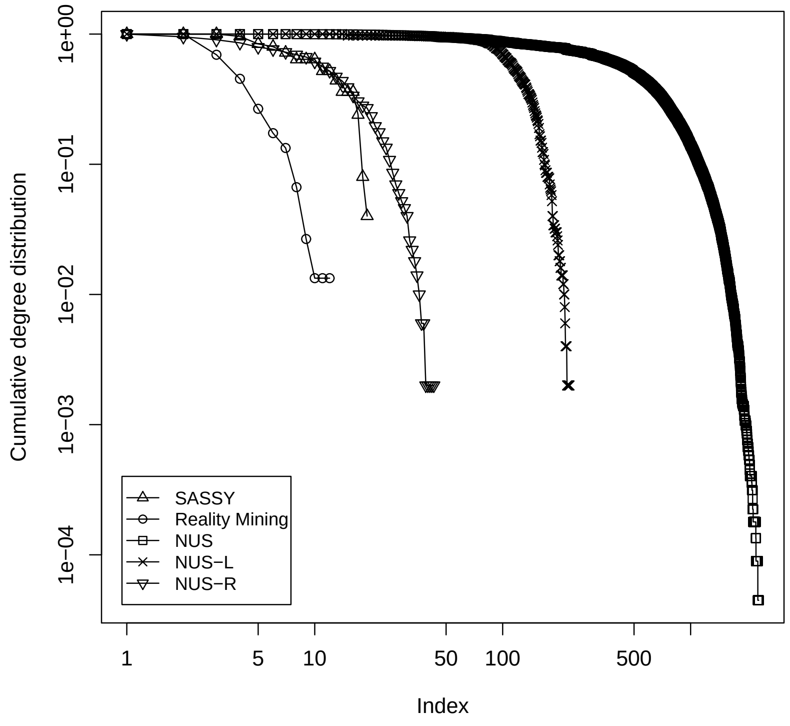


Figure 2: Cumulative degree distributions for the datasets (log-log plot). Our datasets have a wide variance in their cumulative degree distributions.

  - *SASSY* & *Reality Mining*: We simulate 30 days (choosing a random 30 days in the case of *Reality Mining*), and generate 30 messages per day.
  - *NUS-R* & *NUS-L*: Exploiting the cyclic nature of the dataset, for each simulation we select one full week of six business days, each with 13 business hours[7], beginning each simulation at a random time throughout the week. That is, we duplicate the first week to obtain an identical second week, then select out a random one-week period from these two weeks for each simulation run. We simulate 150 messages for each of the six business days; total 900 messages.

- Each message has a TTL of one day:
  - *SASSY* & *Reality Mining*: 24 hours.
  - *NUS-R* & *NUS-L*: 13 business hours, reflecting one business day in the compressed business time representation of the original *NUS* dataset.

- 10 runs for each set of parameters

- *SSNR* obfuscation from -80% to +200% at 20% intervals.[8] When adding nodes to the sender's contact list, the nodes added are chosen from the pool

[7]The original, raw dataset is "compressed" to "business hours", not real time. [24]

[8]For some messages, it may not be possible to continue adding or removing nodes to reach the target modification — if we reach the upper bound of all nodes in the dataset added, or the lower bound of only one node remaining in the sender's social network, we stop adding or removing nodes for this message. For simplicity, we do not simultaneously add and remove nodes from the social network.

of nodes present within the dataset (and within the given time slice from the dataset, in the case of *Reality Mining*).

For the *SASSY* dataset, which contains location information, we used a modified version of the ONE simulator [26], which included our augmented random waypoint model, to generate ns-2 traces. For speed, we used ns-2 rather than ONE for all of the simulations. For the *Reality Mining* and the two *NUS* datasets, which have no location information, we could not use ns-2, so instead used a Python program to parse the encounters and simulate message-passing.

5.3. Performance metrics

To evaluate our simulations, we use the following widely-used metrics from [6]:

- *Delivery ratio:* the proportion of messages that were delivered, out of the total number of unique messages created.
- *Delivery cost:* the total number of messages (including duplicates) transmitted, normalised by the total number of unique messages created.
- *Delivery delay:* the length of time taken for a message to reach its destination: the time between the time at which the message is first sent, and the time at which the message first arrives.

When computing these metrics, we disregard messages which were directly transmitted from original sender to final receiver. In the *NUS* derived datasets, there is a high rate of such encounters — because we derive social network information from the encounters — and so leaving in these messages obscures the performance of (non-trivial) social-network routing, where messages reach their destination via at least one intermediary. For the other datasets, although incidence is not so high, we also disregard such messages so as to allow comparisons across datasets. The performance of social-network routing is therefore underestimated: higher delivery ratios and lower delivery delay would be achievable if we allowed such messages in our analysis.

5.4. OSNR implementation

Since the false positive rate of a Bloom filter depends on the length of the Bloom filter, and the number of elements in the Bloom filter[9], but the number of elements in the Bloom filter greatly varies between datasets (since the sizes of the friends lists vary depending on the scale of the dataset), we choose the Bloom filter length on a per-dataset basis. We aim for a false positive rate of approximately 1% for the unmodified social-network routing case (0% *SSNR*) in each dataset.

Figure 3 shows the actual *OSNR* false positive rate for each dataset, based on the average sizes of the routing friends lists and the lengths of the Bloom filters in each dataset (32 bits for *Reality Mining*, 128 bits for *SASSY* and *NUS-R*, and 1024 bits for *NUS-L*). The higher the average size of the unmodified friends lists, the greater the length of the Bloom filter required in order to target an initial 1% false positive rate.

To insert each element (string representation of a node ID concatenated with a random salt, as described in 4.2) into the Bloom filter, the element's 128-bit MD5 hash[10] was divided into four 32-bit portions, each interpreted as a 32-bit integer.

[9] The Bloom filter false positive rate ε is approximately $(1-e^{-kn/m})^k$, where k is the number of hash functions (in our case, $k = 4$ since we split the 128-bit MD5 hash into four 32-bit integers); n is the number of elements in the Bloom filter; and m is the length of the Bloom filter (in bits).

[10] MD5 is not collision-resistant, but we are only using the uniformity and one-way properties

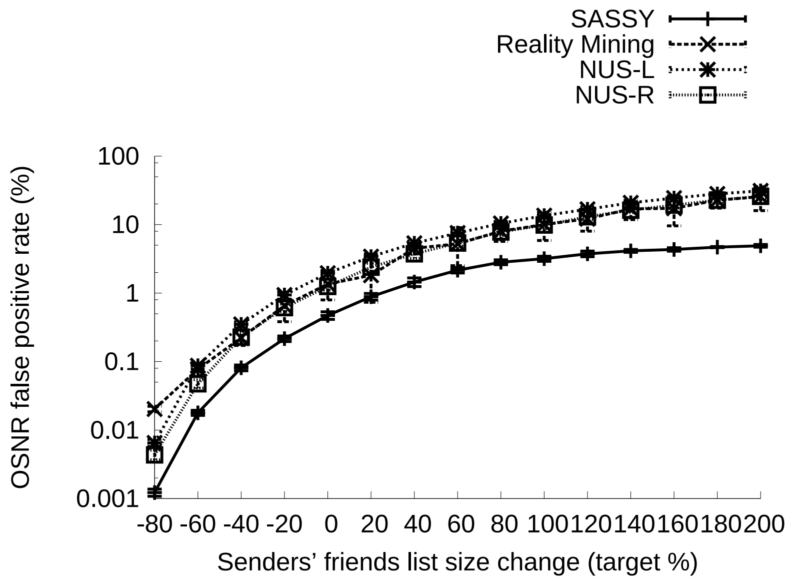


Figure 3: *OSNR* false positive rate for each dataset. The Bloom filter lengths were selected such that for pure *OSNR* (no *SSNR* modification), the false positive rate would be approximately 1%.

Taking each of these four integers mod the Bloom filter length L resulted in four values in range $0..(L-1)$. These four values were interpreted as indices for bits in the Bloom filter; and the corresponding bits were, if not already 1, set to 1.

5.5. Results

Figures 4–15 show our trace-driven simulation results for our routing schemes, for each of our four datasets (*SASSY*, *Reality Mining*, *NUS-L*, *NUS-R*).

— not the collision-resistance property — of MD5. A maliciously-generated collision does not affect the security of our system, because the ability to generate a collision would merely result in another false positive in routing. Such false positives already occur with Bloom filters, and can more easily be triggered by manually setting more bits of the Bloom filter to 1 than by maliciously crafting an MD5 collision.

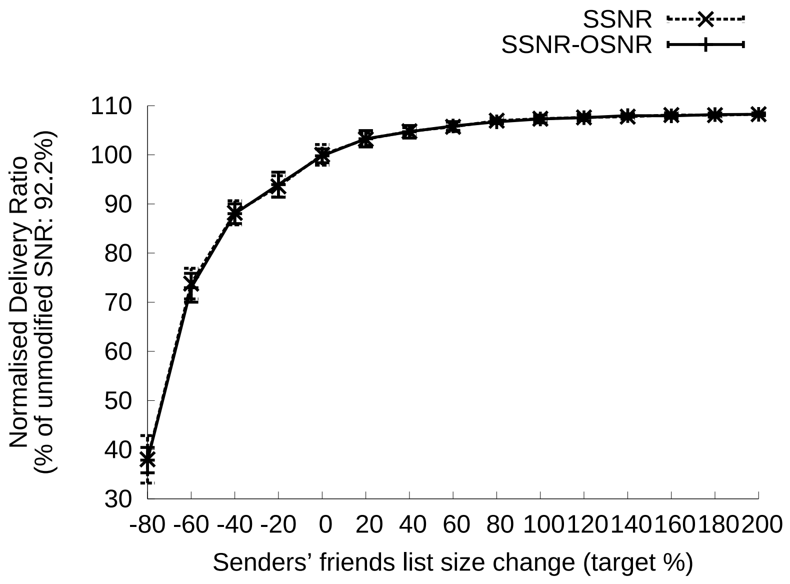


Figure 4: *SASSY* dataset. Message delivery ratio vs target modification of the size of the sender's friends list. Error bars indicate 95% confidence intervals. It is possible to remove 40% of the sender's friends list each message while still retaining high message delivery ratios. Relative to unmodified social-network routing, 88% of messages arrive after removing 40% of the source node's friends list each message.

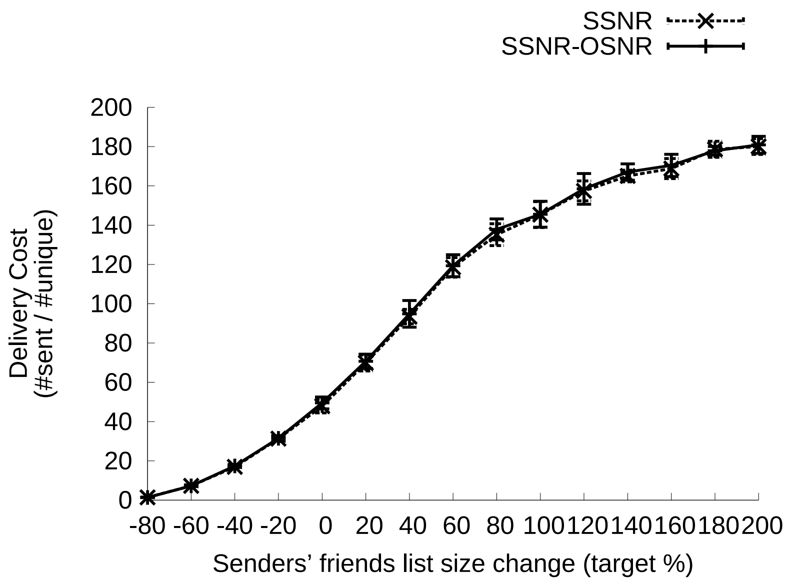


Figure 5: *SASSY* dataset. Message delivery cost vs target modification of the size of the sender's friends list. As we obfuscate the sender's friends list by adding links, the delivery cost increases.

5.5.1. OSNR performance

We note that for every set of friends list size *reductions* for each of our three metrics, *the OSNR scheme did not significantly impact routing performance.*

Figure 3 offers an insight into why this may be: for pure *OSNR* (no *SSNR* modifications), the false positive rate was set — by choosing the length of the Bloom filter — to approximately 1%, as we discussed previously. This 1% false positive rate did not significantly affect routing performance, by any of our metrics. When removing nodes from the senders' friends lists, the false positive rate further decreases — and the decreased false positive rates also do not significantly affect performance by our metrics.

OSNR thus only ever had a noticeable impact on routing performance when *increasing* the size of the senders' friends lists. Even then this impact was often

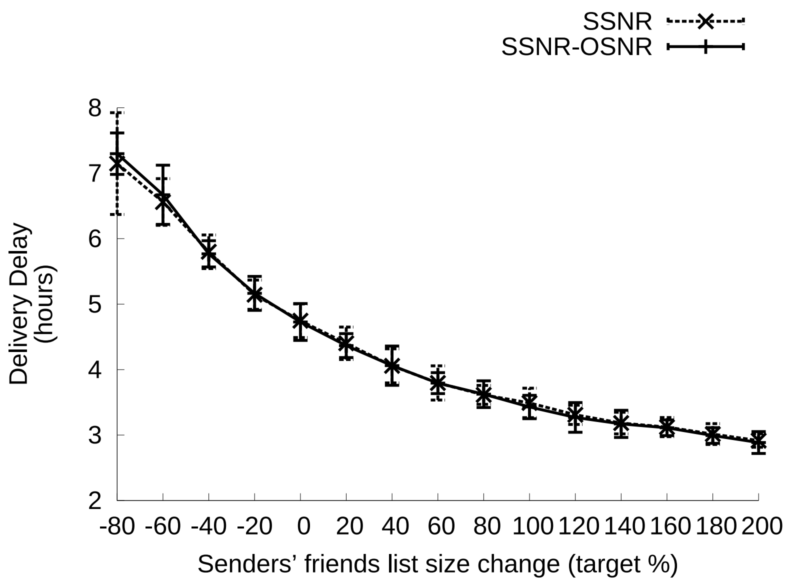


Figure 6: *SASSY* dataset. Message delivery delay vs target modification of the size of the sender's friends list. As we remove from the sender's friends list, delivery delay increases — but only from about 5 to 6 hours for simple social-network routing compared to *SSNR* with -40% change in the sender's friend list.

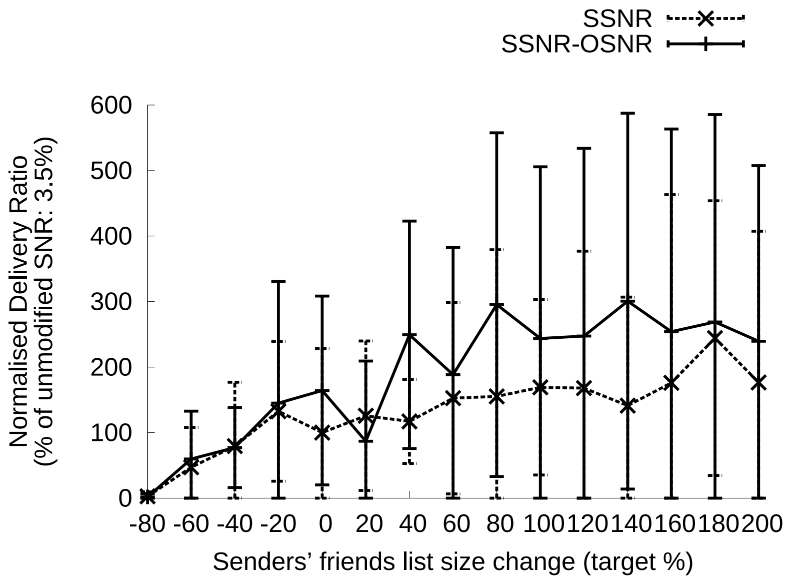


Figure 7: *Reality Mining* dataset. Message delivery ratio vs target modification of the size of the sender's friends list. It is possible to change the sender's friends list size by -40% without significantly reducing the delivery ratio.

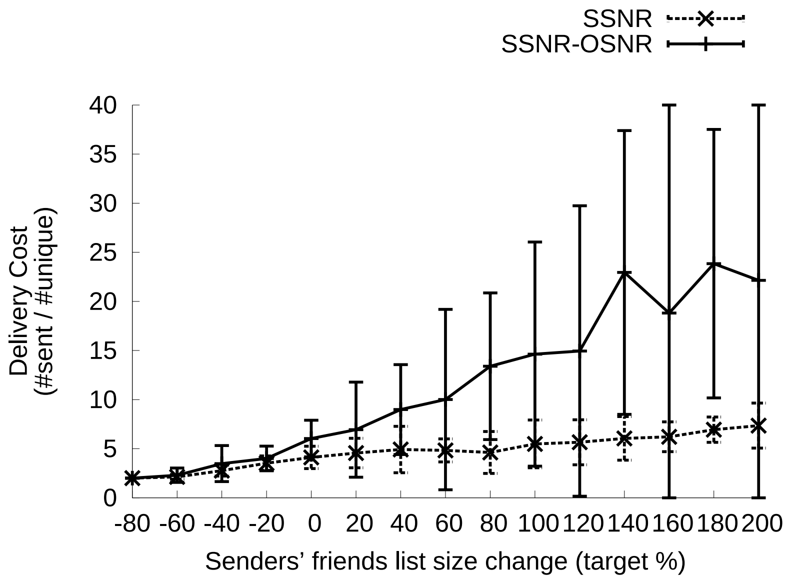


Figure 8: *Reality Mining* dataset. Message delivery cost vs target modification of the size of the sender's friends list. As we obfuscate the sender's friends list by adding fake friends, the delivery cost increases. When using *OSNR*, the false positives associated with using a Bloom filter also lead to an increase in delivery cost.

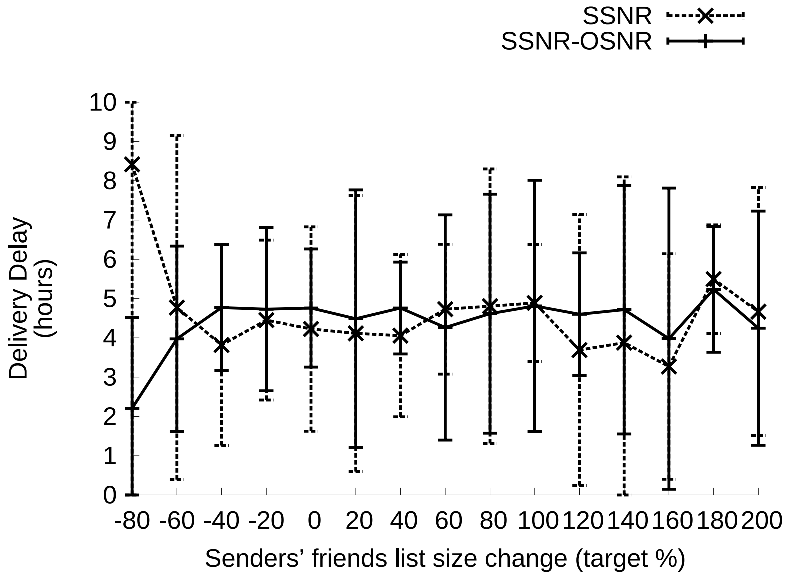


Figure 9: *Reality Mining* dataset. Message delivery delay vs target modification of the size of the sender's friends list. The impact on delivery delay when modifying the sender's friends list size is insignificant.

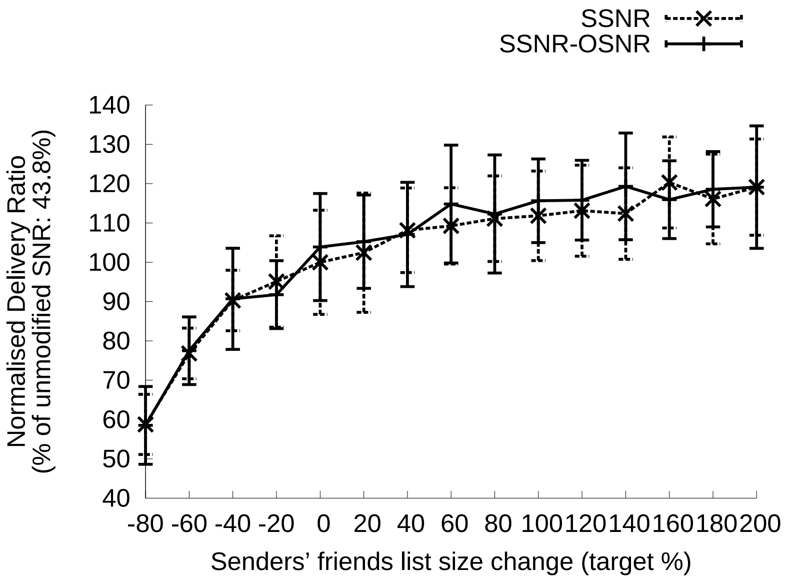


Figure 10: *NUS-L* dataset. Message delivery ratio vs target modification of the size of the sender's friends list. It is possible to remove 40% of the sender's friends list each message while still retaining a 90% message delivery ratio relative to simple social-network routing.

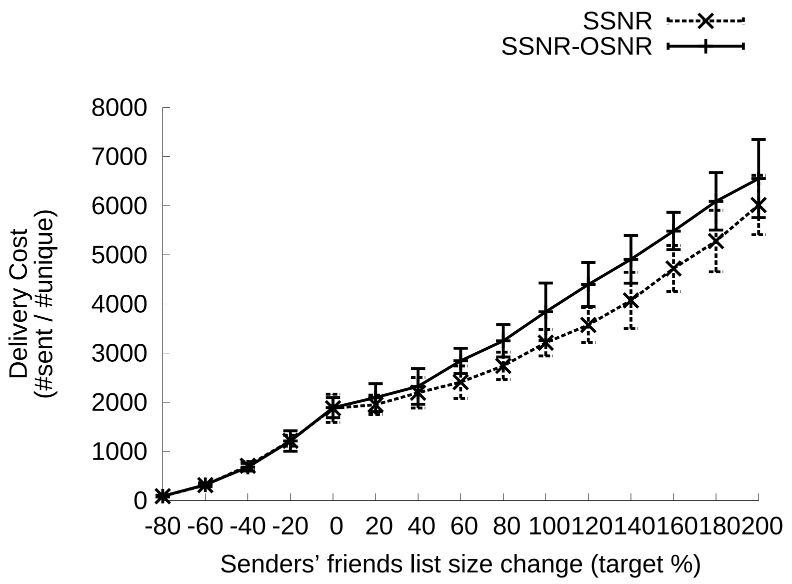


Figure 11: *NUS-L* dataset. Message delivery cost vs target modification of the size of the sender's friends list. As we obfuscate the sender's friends list by adding links, the delivery cost increases.

not significant (as for the *SASSY* dataset, shown in Figures 4–6).

On the few occasions when a significant difference between *SSNR* and *SSNR-OSNR* was visible — such as the upper end of friends list size increases for the *Reality Mining* dataset as shown in Figure 8 — we note from comparison to Figure 3 that the false positive rate for the Bloom filter had grown very high (30% for *Reality Mining* +200% *SSNR*).

5.5.2. SSNR performance

Delivery ratio. Figure 4 shows that for the *SASSY* dataset, the delivery ratio is high for all tested social network size target modifications. It is possible to remove 40% of the nodes from the senders' friends lists, while still retaining a good delivery ratio: almost 90% of the ratio when not modifying the social network at all.

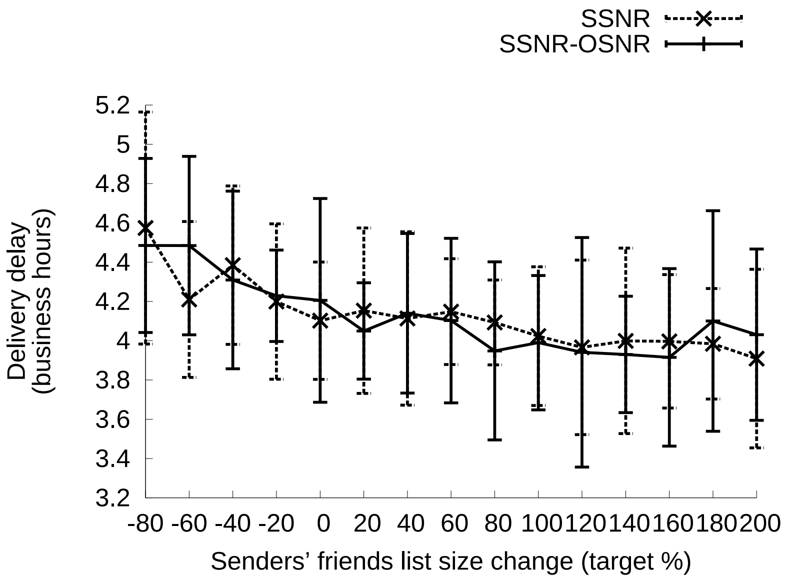


Figure 12: *NUS-L* dataset. Message delivery delay vs target modification of the size of the sender's friends list. As we remove from the sender's friends list, there seems to be a slight trend towards increasing delivery delay — but the increase is slight both in absolute terms, and in relative terms compared to the 95% confidence interval error bars.

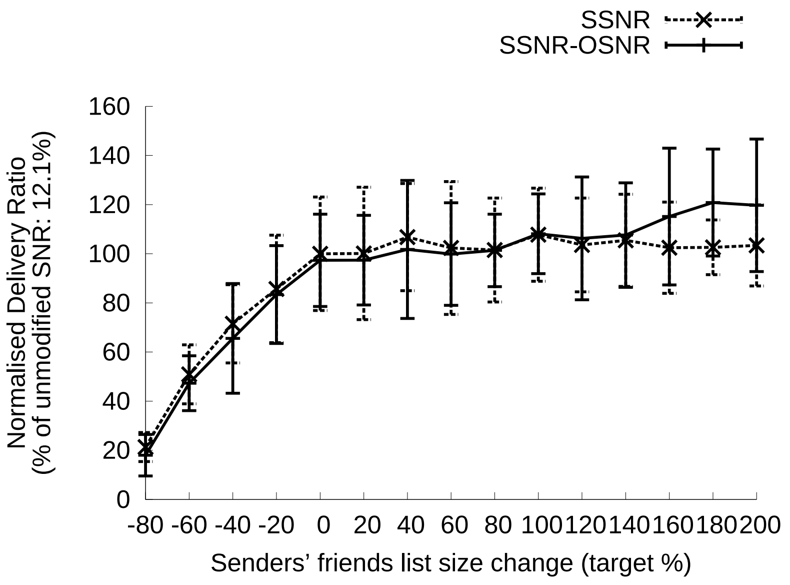


Figure 13: *NUS-R* dataset. Message delivery ratio vs target modification of the size of the sender's friends list. After removing 40% of the sender's friends list each message, the error bars still overlap when compared to simple social-network routing— although the means differ, the difference is smaller than the noise.

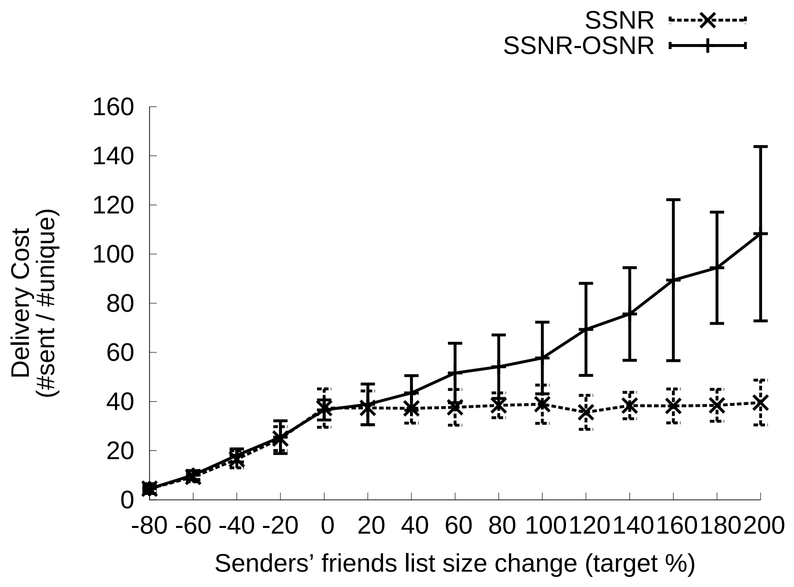


Figure 14: *NUS-R* dataset. Message delivery cost vs target modification of the size of the sender's friends list. As we obfuscate the sender's friends list by adding fake friends, the delivery cost does not significantly change when using pure *SSNR*, but does increase when using combined *SSNR-OSNR*. This may be because of the high false positive rate, c.f. Figure 3. When removing from the friends list, the delivery cost decreases for both schemes (pure *SSNR* and *SSNR-OSNR*).

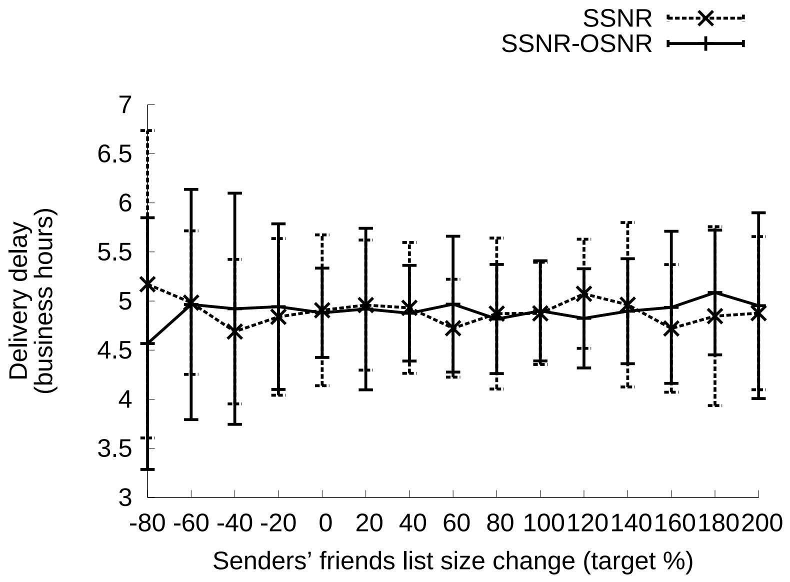


Figure 15: *NUS-R* dataset. Message delivery delay vs target modification of the size of the sender's friends list. The impact on delivery delay when modifying the sender's friends list size is insignificant.

Figure 10 and Figure 13 show similar results for the two datasets derived from *NUS*. Although the absolute figures for delivery ratio are different due to the differing relative connectedness of *NUS-L* and *NUS-R*, the trend for the normalised delivery ratios are similar. Picking out the same -40% *SSNR* modification of the senders' friends lists, we note that, although the means differ (90% of unmodified social-network routing for *NUS-L*; 70% of unmodified social-network routing for *NUS-R*), the difference is smaller than the noise: the error bars overlap.

Although noisier, and with much lower absolute delivery ratios, than the other datasets, we see a similar result holds again for the delivery ratios in the *Reality Mining* dataset in Figure 8. It is possible to make large modifications to the sizes of the senders' friends lists without significantly affecting the delivery ratio, relative to the error margins.

Delivery cost. Figure 5 shows that for the *SASSY* dataset, delivery cost is quite significantly affected by modifying the sender's target friends list size: the fewer nodes in the modified sender's friends list, the lower the cost of sending a message. Compared to simple social-network routing, with 50 data messages per unique message, a -40% change in sender friends list results in only 20 data messages: fewer than half as many data messages. This corresponds to the result from Figure 4, where we still retain a high delivery ratio. By applying *SSNR*, we have actually improved performance by this metric, by reducing the delivery cost, but while simultaneously retaining a good delivery ratio — and increasing the sender's privacy by not revealing some of their true friends.

Figure 11 and Figure 14 show similar results (in relative, not absolute) terms for delivery cost for the *NUS-L* and *NUS-R* datasets, when removing from the senders' friends lists. A -40% change in the size of these lists results in a more than halving of data messages for both datasets. The relative differences are large, though: for *NUS-L* the change in cost is from 1900 to 700, while for *NUS-R* the change in cost is from 37 to 17. The absolute difference is again presumably due to the differing degrees of connectivity of the datasets, as illustrated earlier in this section. The differing connectivity presumably also accounts for the differing performance of pure *SSNR* when increasing the senders' friends lists for these datasets: for the highly-connected *NUS-L* dataset encounters with the "fake" new friends are likely, increasing delivery cost, while for the less-connected *NUS-R* dataset such encounters do not occur so much, keeping delivery cost about constant. We note that combined *SSNR-OSNR* does show an increase in delivery cost for *NUS-R*, because extra encounters do occur (and with high false positive rates for the Bloom filter result in message forwarding) — but the number of such extra encounters is high relative to the relatively-small size of the senders' friends lists (as shown in Figure 2), and hence the absolute number of extra forwarding opportunities in pure *SSNR* is low, keeping the delivery cost unchanged.

Figure 8 shows that delivery cost for the *Reality Mining* dataset stays fairly constant (since it is so low in absolute terms) in applying *SSNR* which reduces the friends lists sizes: the delivery cost falls from five messages to three messages on applying -40% SSNR. A similar effect is seen on applying *SSNR* which increases the friends lists sizes as for the *NUS-R* dataset. The senders' friends lists are, in absolute terms, small, as shown in Figure 2. So increasing the relative size of the friends lists does not dramatically change the delivery cost with pure *SSNR*, since few encounters occur with the added fake friends. However, when adding these fake friends on applying combined *SSNR-OSNR*, the false positive rate ends up high (as shown in Figure 3) — about 30% at the upper end of the scale. This means that extra messages are forwarded, as triggered by this high false positive rate. Hence the delivery cost increases.

Delivery delay. Figure 6 shows that for the *SASSY* dataset, delivery delay increases somewhat when removing nodes from the sender's friends list. This increase is only from about 4.8 to 5.8 hours when using -40% *SSNR* compared to unmodified social-network routing. Indeed, if delivery delay is a concern, we may also reduce delivery delay by adding nodes with *SSNR*.

For the other datasets, the difference is not significant: Figure 12 (*NUS-L*); Figure 15 (*NUS-R*); and Figure 9 (*Reality Mining*) all show little (if any) correlation between delivery delay and modification of the senders' friends lists. If such a difference exists, it is smaller than the noise — the error bars overlap within each dataset for each set of *SSNR* parameters.

Summary. Finally, we observe that for all the datasets, it is possible to significantly modify the size of the sender's friends list (for example, by -40%), thus increasing the privacy of the sender, and yet to still retain good routing performance. Indeed, removing nodes may significantly reduce delivery cost — a beneficial side effect while enhancing privacy. If delivery delay or ratio is more of a concern, conversely, *SSNR* allows adding nodes to improve performance by these metrics, again while enhancing privacy, though at the expense of increased delivery cost in this case.

We quantify the improvement in security in the next section.

6. Security discussion

The simulation experiments in Section 5 demonstrate that our schemes are practical with respect to performance: we are able to obfuscate the friends lists used for routing without a large impact on opportunistic network performance. We now consider the practicality of our schemes with respect to security — we discuss the privacy gains in using our schemes, with reference to classes of attack which are mitigated.

6.1. Security of OSNR

We first consider the *OSNR* scheme — where we “hash” the friends list of the sender to a Bloom filter.

In simple social-network routing, an attacker may read the sender’s friends list in plain text from one single eavesdropped message, as might an intermediate node who has legitimately received a message for forwarding. By hashing the sender’s friends list to a Bloom filter, we raise the bar for a curious, casual observer — such as one of the sender’s friends who legitimately receives a message as part of social-network routing. Our scheme keeps honest people honest. But we also increase the effort required by a malicious attacker. By how much?

6.1.1. OSNR with single intercepted message

In this attack an attacker attempts to reverse the Bloom filter, i.e., deduce the sender’s original friends list from the Bloom filter. The attacker does so by iterating through the universe of elements that may be contained within the Bloom filter, and testing the Bloom filter for membership of each of these elements. For example, if a Bloom filter contains (salted) elements concatenated with Bluetooth address, then to reverse the Bloom filter one should test each of the 2^{32} (similarly salted) Bluetooth addresses, for presence in the Bloom filter.

We tested how long it might take to reverse the Bloom filters used in Section 5 on one of our compute servers — a machine with two Intel Xeon L5320 processors

(2x quad core at 1.86GHz). We were able to test approximately 2^{14} Bluetooth addresses (concatenated with salt) for presence in a Bloom filter per CPU core per second. Iterating through the universe of Bluetooth addresses took 58 CPU-hours.

Since Bloom filters guarantee no false negatives, all of the addresses encoded inside the Bloom filter would be found by such iteration through all possible elements Bloom filters produce false positives, however, with a known rate ε — e.g., we targeted $\varepsilon = 1\%$ in choosing the Bloom filter length in our experiments. Therefore, the addresses truly encoded in the Bloom filter would be lost in the sea of false positives: with 4.3B Bluetooth addresses, we would expect 43M false positives. Thus an attacker would find it difficult to accurately deduce a node's friends list from eavesdropping a single message.

6.1.2. OSNR with multiple intercepted messages

We now consider an attacker who can intercept multiple messages. In our *OSNR* scheme, each subsequent intercepted message would allow the attacker to reduce the set of false positives (size f) to a new subset of size $\approx \varepsilon \cdot f$, each round.

Therefore, for Bluetooth addresses (2^{32} possible addresses) and a false positive rate $\varepsilon = 1\%$, the expected number of false positives, f, after intercepting n distinct messages is $f = 2^{32} \cdot 0.01^n$.

To recover the original friends list of the sender, the attacker must intercept sufficiently many messages, n, that $f < 1$. Rearranging the previous equation, this is $n = log_{0.01}(2^{-32}) \simeq 4.8 \simeq 5$.

So under the assumptions above, the attacker must intercept approximately five distinct messages in order to recover the sender's original friends list.

The bulk of the computational burden on the attacker is reversing the first intercepted message's Bloom filter. After that, the attacker need only test the exponentially-decreasing number of elements from the previous rounds against newly-intercepted Bloom filters.

6.1.3. Implications of combined SSNR-OSNR

The combined *SSNR-OSNR* scheme is able to mitigate the eavesdropping attack, but how many distinct messages must an attacker intercept in order to recover the original friends list of a sender who is employing *SSNR*?

Section 5 shows that *SSNR* allows us to randomly remove 40% of the sender's true friends list per message without a major degradation in social-network routing performance. Using -40% *SSNR*, the probability of each member in a friends list appearing in a given message is $1-0.4=0.6$. As the attacker intercepts a number of messages n, the number of messages x in which a given member of a friends list appears (with appearance being random per message) is therefore binomially distributed, $x \sim B(n, 0.6)$.

Figure 16 shows the probability of the attacker identifying each friends list node as n increases, according to different threshold values of x, again using -40% *SSNR*. Using pure *SSNR*, the threshold is $x \geq 1$ — there are no false positives. To identify 95% of friends list nodes, four messages must be intercepted.

In practice, though, we combine *SSNR-OSNR*. The false positive rate is now defined by the Bloom filter. Using a $\varepsilon = 1\%$ as in our previous discussion, a suitable threshold might be $x \geq 3$ — the attacker may be confident that a friends list node is truly identified if the node appears in three or more intercepted distinct messages. Using this threshold, Figure 16 shows that the attacker must intercept eight messages in order to identify 95% of friends list nodes.

Moreover, when combining *SSNR-OSNR*, the optimisation in Section 6.1.2 (discarding all addresses except those that were flagged up in previous rounds;

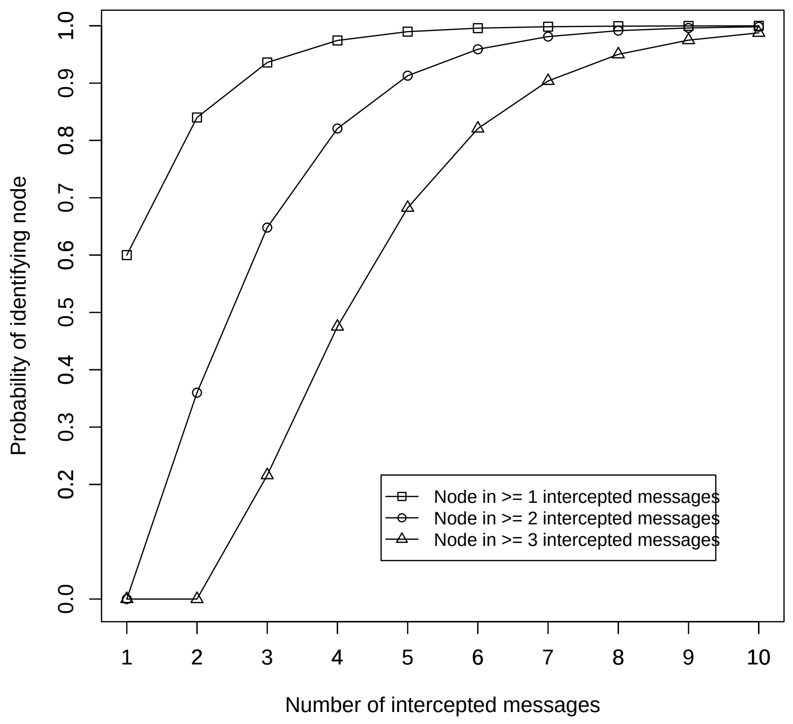


Figure 16: Probability of identifying each node within the sender's original social network after applying *SSNR* (−40%), as a function of the number of distinct messages intercepted. Using *SSNR-OSNR*, we consider the attacker as identifying a friends list node if that node appears in three or more distinct intercepted messages. To identify 95% of nodes, the attacker must intercept eight distinct messages.

initially mostly false positives but with the true addresses mixed in too) is also defeated since false negatives are now possible, further increasing the computational burden on the attacker.

6.1.4. Burden on the attacker

If an attacker must collect approximately eight messages in order to deduce the original sender's friends list, then how practical an attack is this?

Firstly, these messages need to be distinct. In *OSNR*, the Bloom filter is added by the original sender at the time of message generation, and is not altered en-route during routing. Therefore, distinct messages must be intercepted in order to obtain messages with different Bloom filters — it is not useful to capture the same message as it is routed through the network since the Bloom filter will be unchanged. Most opportunistic network implementations, however, are likely to employ bundle protocols which aggregate many application-layer data units into few network-layer data units for forwarding [27], thus hindering the eavesdropping of multiple messages.

To collect these messages, the attacker could shadow the sender, but if this were possible, then the attacker could directly observe the sender's interactions with other nodes and directly measure the sender's social network, rendering the attack redundant. An alternative strategy is to eavesdrop constantly in a well-known busy spot. Again, if this were possible, then an attacker could directly observe the social networks of many nodes.

Our schemes, therefore, are not infallible, but instead serve to raise the amount of effort required for an attack. Instead of being able to discover a sender's friends list by intercepting a single message and then reading off the data in plain-text, the attacker must now intercept multiple messages, and then devote multiple days of

CPU time to the attack.

6.2. Linkability

While the bar for an attacker has been raised significantly for reversing a single sender's friends list, so too has the bar been raised much more for obtaining even a relatively small portion of the whole social network.

The structure of the social network itself is sensitive information. For example, Narayanan and Shmatikov have shown that it is possible to link individuals who are members of different online social networks, based on no more information than anonymised node-edge graphs of both social networks. [28] Anonymity of social network participants is thus not sufficient for privacy, since the participants may be linked to another social network in which they also participate.

Narayanan's deanonymisation algorithm is described as being "robust to mild modifications of the topology such as those introduced by sanitization". This is because it deanonymises nodes by starting out at known seeds whose with positions known in both networks, and then crawling outwards from those seeds to find corresponding nodes in the two networks.

Thus, to be able to use this algorithm against an opportunistic network, an attacker would now have to be able to deduce accurate friends lists for a significant proportion of nodes close together in the social network. Crucially, the algorithm cannot "jump the gap" between disconnected subgraphs, so deducing the friends lists of some isolated nodes is not sufficient: the attack would only succeed if large-scale deduction of *all* the friends lists for nodes within a connected subgraph of the social network were achieved by the attacker.

Such an attack would be difficult, but feasible, for a simple social-network routing routing scheme. A single eavesdropped message would reveal the sender's complete local friends list information. Thus, by sniffing a sample of messages, the attacker may be able to gain enough information to reconstruct a fairly sizeable connected subgraph of the complete social network.

Our *SSNR-OSNR* scheme prevents this attack from being successful — or, at least, raises the bar very much higher for a potential attacker. The number of messages that must be sniffed is increased of the order of tenfold, since, as discussed in the previous section, approximately eight distinct messages from each sender must be intercepted to obtain the sender's local social network neighbours (their friends list) with some reasonable confidence. This must be repeated for each sender.

It therefore appears that our scheme may make these linkability attacks difficult, and, we believe, impractical.

7. Related work

If nodes are to trust their data with any other nodes that they encounter, then privacy is paramount. Few researchers, however, have studied privacy in opportunistic networks. The HiBOp scheme [29] proposes a key management solution, where users are divided into communities and public-key cryptography is used to secure communication within a community, while some nodes are chosen to forward messages between communities. Key distribution and management in such a scheme is very difficult in a mobile ad hoc environment, however, and may impede the very feature which makes opportunistic networking so appealing — the fact that nodes may forward to *any* node that they encounter. Thus we have aimed to build privacy-enhancing schemes that do not require a public-key infrastructure. If nodes have to choose communities and register to obtain keys, as in

HiBOp, then this may limit the number of potential nodes that may participate in the network, and so limit its usefulness. Moreover, even within a scheme such as that proposed for HiBOp, members of a community can observe the routing tables of all other members of the community, and so many of the attacks that we describe in this paper are still possible.

Shikfa et al. [30] also propose group-based cryptography for opportunistic networks, using multiple levels of cryptography to prevent data from being accessed by different groups. They concentrate on protecting the application-layer data payload, rather than the routing information as we do here.

Lilien et al. [31] explore privacy in opportunistic networks, but their definition of an opportunistic network is akin to finding helper nodes in an ad hoc network, rather than the mobile opportunistic network definition that we use here. They list a number of privacy challenges and proposed solutions, including maintaining a list of trusted devices, intrusion detection systems, and the use of public-key infrastructure. Again, we believe that these systems would be impractical in a pure opportunistic network and therefore investigating schemes which do not require PKI has merit.

Aad et al. [32] present methods to improve anonymity within an ad hoc network. These include using Bloom filters to compress and obscure a packet's routing list, and a technique for combining multicast and onion routing. However, they assume global routing information is available for the ad hoc network, which we do not; and they do not carry out simulations to evaluate performance, as we do here.

Another popular mechanism for enhancing privacy in networks is onion routing [33], where packets are routed through a group of collaborating nodes, thus mingling connections and making it difficult to determine the source of a communication. Le et al. [13] propose such a scheme for opportunistic networks. Onion routing, however, does not prevent eavesdropping attacks from intermediate nodes as we have described here, and Le et al.'s scheme still requires a PKI.

Obfuscation is widely used for enhancing privacy in Internet applications. TrackMeNot [34] adds noise to a user's search engine queries by generating fake search queries to make it harder for a search engine provider to profile individual users. Viejo and Castellà-Roca extend this idea by sending search queries through an onion-like network of users comprised of a user's social network friends [35]. Guha et al. [36] uses obfuscation to enhance privacy in social network sites. Users' information is mixed with information from other SNS users to make it difficult for an attacker to accurately profile an individual user. Dóra and Holczer [37] extend this to opportunistic networks by obfuscating users' interests in an opportunistic publish-subscribe application, in an attempt to prevent attackers from identifying a user's specific interests.

We have already discussed Narayanan and Shmatikov's work on de-anonymising social networks through comparison of an anonymised dataset to another available social network dataset [28]. Wondracek et al. [38] demonstrate that it is also possible to deanonymise social network data if group membership information is public, as is common on many social network sites.

8. Conclusions

Social network information is commonly used for improving routing performance in opportunistic networks. In this paper we have presented two schemes for enhancing privacy in such social-network routing systems. We find that it is possible to obfuscate a sender's friends list by removing up to 40% of the nodes from the social network, while still maintaining a typical mean delivery ratio that is approximately 90% that of unaltered social-network routing. In addition we have demonstrated that, by using Bloom filters, we can prevent eavesdropping of social network information with a minimal effect on network performance.

We have evaluated these two schemes using three opportunistic network datasets: one collected by ourselves, and the widely-used MIT *Reality Mining* and *NUS* datasets. Although these datasets vary widely in many properties (including scale, location and connectivity), our findings appear to hold for all three datasets, which gives us confidence that our schemes would be deployable in a real-world opportunistic network. We have also considered attacks against our schemes, and demonstrated the classes of attack which we may mitigate.

Acknowledgements

The authors thank Greg Bigwood for his help in collecting the SASSY dataset. This work is supported by the UK Engineering and Physical Sciences Research Council and the Technology Strategy Board as part of the Privacy Value Networks project, EP/G002606/1.

References

[1] I. Parris, G. Bigwood, T. Henderson, Privacy-enhanced social network routing in opportunistic networks, in: Proceedings of the IEEE International Workshop on SEcurity and SOCial networking (SESOC), IEEE, Los Alamitos, CA, USA, 2010, pp. 624–629. `doi:10.1109/PERCOMW.2010.5470511`.

[2] L. Pelusi, A. Passarella, M. Conti, Opportunistic networking: data forwarding in disconnected mobile ad hoc networks, IEEE Communications Magazine 44 (11) (2006) 134–141. doi:10.1109/MCOM.2006.248176.

[3] J. Crowcroft, E. Yoneki, P. Hui, T. Henderson, Promoting tolerance for delay tolerant network research, ACM SIGCOMM Computer Communication Review 38 (5) (2008) 63–68. doi:10.1145/1452335.1452345.

[4] A. Vahdat, D. Becker, Epidemic routing for partially-connected ad hoc networks, Tech. Rep. CS-200006, Duke University (Apr. 2000).
URL http://issg.cs.duke.edu/epidemic/epidemic.pdf

[5] A. Lindgren, A. Doria, O. Schelén, Probabilistic routing in intermittently connected networks, in: Proceedings of the 1st International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR), 2004, pp. 239–254. doi:10.1007/b99076.

[6] P. Hui, J. Crowcroft, E. Yoneki, Bubble rap: social-based forwarding in delay tolerant networks, in: Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), ACM, New York, NY, USA, 2008, pp. 241–250. doi:10.1145/1374618.1374652.

[7] E. M. Daly, M. Haahr, Social network analysis for information flow in disconnected delay-tolerant MANETs, IEEE Transactions on Mobile Computing 8 (5) (2009) 606–621. doi:10.1109/TMC.2008.161.

[8] J. A. Bitsch Link, N. Viol, A. Goliath, K. Wehrle, SimBetAge: utilizing temporal changes in social networks for pocket switched networks, in: Proceedings of the 1st ACM workshop on User-provided networking: challenges and

opportunities (U-NET), ACM Press, New York, NY, USA, 2009, pp. 13–18. doi:10.1145/1659029.1659034.

[9] A. J. Mashhadi, S. B. Mokhtar, L. Capra, Habit: Leveraging human mobility and social network for efficient content dissemination in delay tolerant networks, in: Proceedings of the 10th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2009, pp. 1–6. doi:10.1109/WOWMOM.2009.5282467.

[10] A. Mtibaa, M. May, C. Diot, M. Ammar, PeopleRank: Social opportunistic forwarding, in: Proceedings of the IEEE INFOCOM 2010 Mini-Conference, IEEE, Los Alamitos, CA, USA, 2010. doi:10.1109/INFCOM.2010.5462261.

[11] G. Bigwood, D. Rehunathan, M. Bateman, T. Henderson, S. Bhatti, Exploiting self-reported social networks for routing in ubiquitous computing environments, in: Proceedings of the 1st International Workshop on Social Aspects of Ubiquitous Computing Environments (SAUCE), Los Alamitos, CA, USA, 2008, pp. 484–489. doi:10.1109/WiMob.2008.86.

[12] D. Boyd, N. B. Ellison, Social network sites: Definition, history, and scholarship, Journal of Computer-Mediated Communication 13 (1) (2008) 210–230. doi:10.1111/j.1083-6101.2007.00393.x.

[13] Z. Le, G. Vakde, M. Wright, PEON: privacy-enhanced opportunistic networks with applications in assistive environments, in: Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assis-

tive Environments (PETRA), ACM Press, New York, NY, USA, 2009, pp. 1–8. doi:10.1145/1579114.1579190.

[14] B. Schneier, Secrets and Lies: Digital Security in a Networked World, Wiley, 2004.

[15] S. K. Belle, M. Waldvogel, Consistent deniable lying: Privacy in mobile social networks, in: Proceedings of the Workshop on Security and Privacy Issues in Mobile Phone Use, 2008.
URL http://www.pervasive2008.org/Papers/Workshop/w1-03.pdf

[16] D. Huff, How to Lie With Statistics, W. W. Norton & Company, 1954.

[17] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (7) (1970) 422–426. doi:10.1145/362686.362692.

[18] P. Oechslin, Making a faster cryptanalytic time-memory trade-off, in: Proceedings of Advances in Cryptology (CRYPTO 2003), Springer, Berlin, Germany, 2003, pp. 617–630.
URL http://www.springerlink.com/content/U9GXWD29P2TNX3WL

[19] F. Ekman, A. Keränen, J. Karvo, J. Ott, Working day movement model, in: Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models (MobilityModels), ACM, New York, NY, USA, 2008, pp. 33–40. doi:10.1145/1374688.1374695.

[20] R. Friedman, M. Gradinariu, G. Simon, Locating cache proxies in MANETs, in: Proceedings of the 5th ACM international symposium on Mobile ad hoc

networking and computing (MobiHoc), ACM, New York, NY, USA, 2004, pp. 175–186. doi:10.1145/989459.989482.

[21] N. Eagle, A. S. Pentland, CRAWDAD data set mit/reality (v. 2005-07-01), Downloaded from http://crawdad.cs.dartmouth.edu/mit/reality (Jul. 2005).

[22] N. Eagle, A. S. Pentland, D. Lazer, Inferring friendship network structure by using mobile phone data, Proceedings of the National Academy of Sciences 106 (36) (2009) 15274–15278. doi:10.1073/pnas.0900282106.

[23] V. Srinivasan, M. Motani, W. T. Ooi, CRAWDAD data set nus/contact (v. 2006-08-01), Downloaded from http://crawdad.cs.dartmouth.edu/nus/contact (Aug. 2006).

[24] V. Srinivasan, M. Motani, W. T. Ooi, Analysis and implications of student contact patterns derived from campus schedules, in: Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MobiCom), ACM, New York, NY, USA, 2006, pp. 86–97. doi:10.1145/1161089.1161100.

[25] C. Liu, J. Wu, Routing in a cyclic mobispace, in: Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc), ACM, New York, NY, USA, 2008, pp. 351–360. doi:10.1145/1374618.1374665.

[26] A. Keränen, J. Ott, T. Kärkkäinen, The ONE simulator for DTN protocol evaluation, in: Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools), ICST, Brussels, Belgium, 2009, pp. 1–10. doi:10.4108/ICST.SIMUTOOLS2009.5674.

[27] K. Fall, A delay-tolerant network architecture for challenged internets, in: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), ACM, New York, NY, USA, 2003, pp. 27–34. doi:10.1145/863955.863960.

[28] A. Narayanan, V. Shmatikov, De-anonymizing social networks, in: Proceedings of the 30th IEEE Symposium on Security and Privacy, IEEE, Los Alamitos, CA, USA, 2009, pp. 173–187. doi:10.1109/SP.2009.22.

[29] C. Boldrini, M. Conti, A. Passarella, Exploiting users' social relations to forward data in opportunistic networks: The HiBOp solution, Pervasive and Mobile Computing 4 (5) (2008) 633–657. doi:10.1016/j.pmcj.2008.04.003.

[30] A. Shikfa, M. Onen, R. Molva, Privacy in content-based opportunistic networks, in: Proceedings of the 2nd IEEE International Workshop on Opportunistic Networking (WON-09), IEEE Computer Society, Los Alamitos, CA, USA, 2009, pp. 832–837. doi:10.1109/WAINA.2009.73.

[31] L. Lilien, Z. H. Kamal, V. Bhuse, A. Gupta, The concept of opportunistic networks and their research challenges in privacy and security, in: Mobile and Wireless Network Security and Privacy, Springer US, Boston, MA, USA, 2007, Ch. 5, pp. 85–117. doi:10.1007/978-0-387-71058-7_5.

[32] I. Aad, C. Castelluccia, J.-P. Hubaux, Packet coding for strong anonymity in ad hoc networks, in: Proceedings of the 2nd International Conference on Security and Privacy in Communication Networks (SecureComm), IEEE,

Los Alamitos, CA, USA, 2006, pp. 1–10. doi:10.1109/SECCOMW.2006.359571.

[33] D. Goldschlag, M. Reed, P. Syverson, Onion routing, Communications of the ACM 42 (2) (1999) 39–41. doi:10.1145/293411.293443.

[34] D. C. Howe, H. Nissenbaum, TrackMeNot: Resisting surveillance in web search, in: I. Kerr, V. Steeves, C. Lucock (Eds.), Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society, Oxford University Press, Oxford, UK, 2009, Ch. 23, pp. 417–436.

[35] A. Viejo, J. Castellà-Roca, Using social networks to distort users' profiles generated by web search engines, Computer Networks 54 (9) (2010) 1343–1357. doi:10.1016/j.comnet.2009.11.003.

[36] S. Guha, K. Tang, P. Francis, NOYB: privacy in online social networks, in: Proceedings of the 1st Workshop on Online Social Networks (WOSN), ACM, New York, NY, USA, 2008, pp. 49–54. doi:10.1145/1397735.1397747.

[37] L. Dóra, T. Holczer, Hide-and-Lie: enhancing application-level privacy in opportunistic networks, in: Proceedings of the 2nd International Workshop on Mobile Opportunistic Networking (MobiOpp), ACM, New York, NY, USA, 2010, pp. 135–142. doi:10.1145/1755743.1755767.

[38] G. Wondracek, T. Holz, E. Kirda, C. Kruegel, A practical attack to de-anonymize social network users, in: Proceedings of the 31st IEEE Symposium on Security and Privacy, IEEE, Los Alamitos, CA, USA, 2010.
URL http://www.iseclab.org/papers/sonda-TR.pdf