

OMiN - an Opportunistic Microblogging Network

Neil Wells & Tristan Henderson

ABSTRACT

Opportunistic networks are possibly the most difficult networks to work with – nodes are mostly disconnected and occasionally encounter other random nodes for a short period of time. This leads to a number of unsolved problems about securely and effectively routing packets. This report describes OMiN, a microblogging service which uses an opportunistic network of smartphones to transmit messages. It also presents a new cryptographic approach to security in opportunistic networks. I hope that, in the future, OMiN and similar networks will be deployed to allow easy, cheap communication in areas which are unable to connect to the Internet.

DECLARATION

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 11,984 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

CONTENTS

1	Introduction	5
2	Background.....	6
2.1	Microblogging	6
2.2	Opportunistic Networks	6
2.3	Routing Algorithms	7
2.4	Security	12
2.5	Similar Projects.....	16
3	Use Cases.....	19
3.1	Disaster Area.....	19
3.2	Metro	19
4	Threats	20
4.1	Disaster Area.....	20
4.2	Metro	20
4.3	All Threats.....	20
5	Objectives	22
5.1	Primary Objectives.....	22
5.2	Secondary Objectives	22
5.3	Tertiary Objectives	22
6	Requirements Specification.....	23
6.1	User Requirements.....	23
6.2	System Requirements	23
7	Design.....	25
7.1	Social Structure	25
7.2	Security Scheme.....	25
7.3	Routing.....	30
7.4	Message Buffer Eviction	31
7.5	Database Design.....	31
7.6	User Interface Design.....	32
8	Implementation	34
8.1	Mobile Platform	34
8.2	Programming Language.....	34
8.3	Message Passing Medium	34
8.4	Cryptography Scheme	34
8.5	PKG Server.....	35
8.6	Database Library	36
8.7	Logging.....	36
8.8	3 rd Party Libraries and Code	36
9	Software Engineering Process	38
9.1	Task Management.....	38
9.2	Storage.....	38

9.3	Implementation	38
9.4	Regression Testing.....	39
9.5	Problems Encountered While Implementing	39
10	Ethics.....	41
11	Evaluation and Critical Appraisal.....	42
11.1	Impact of Cryptography.....	42
11.2	Objectives	42
11.3	Requirements	42
11.4	Use Cases.....	43
11.5	Further Work.....	44
11.6	Overall Evaluation	46
12	Conclusion.....	47
13	Acknowledgements	48
14	Bibliography.....	49
15	Appendices.....	52
Appendix 1	Dictionary of Terms.....	52
Appendix 2	User Manual	53
Appendix 3	Maintenance Document.....	55
Appendix 4	Example Logging Data.....	56
Appendix 5	Cryptography Timing Data	56
Appendix 6	Objectives Met	58
Appendix 7	Requirements Met.....	58

1 INTRODUCTION

The world has become much more connected in recent years due to the ubiquity of the Internet, but there are still huge areas – such as developing countries and disaster areas – where the lack of Internet connection is a huge disadvantage. Bringing the internet to these areas requires a lot of infrastructure and money, but a developing technology – the opportunistic network – can provide connectivity to the outside world in an affordable way by using mobile devices (such as smartphones) which move about to transport messages. This report describes OMiN, a microblogging platform using an opportunistic network of Android smartphones and tablets to transfer messages using Bluetooth. This report also proposes solutions to the problems of routing messages effectively and distributing them securely.

2 BACKGROUND

The following provides a summary of opportunistic networks and the current state of opportunistic network technology. Only the most relevant subjects will be addressed in order to give the reader sufficient background information to fully understand the project.

2.1 MICROBLOGGING

A microblogging service allows users to post short messages (microblogs) which can be viewed by others [1]. They are often used as part of a social network – online networks of users, each with a ‘profile’ and set of connections to other users [2]. Twitter’s¹ 140-character tweets are a good example of microblogging in a social network environment. Microblogging services are rapidly gaining popularity because of their ability to spread news quickly and to distil updates into short summaries [1].

2.2 OPPORTUNISTIC NETWORKS

An opportunistic network is a network where connections between nodes are sparse and a direct path from source to destination is rarely possible [3]. Opportunistic networks have a key advantage over any other form of network – they do not require any infrastructure (cables, towers etc.) to work. While they are very slow compared to other forms of networks, they are often deployed in areas where other forms of network cannot be used because the infrastructure does not exist (such as disaster areas where the infrastructure has been wiped out and developing countries where the infrastructure is too expensive).

A common form of opportunistic network (and the form we will focus on) is the Pocket Switched Network (PSN) – a network of devices (normally smartphones) carried around by people [4]. Connections are made between devices in close proximity using a short range protocol such as Bluetooth. In this paper, the concepts of users and nodes are interchangeable.

For example (Figure 1), a message could be sent from sender S to destination D even when S and D never meet. S encounters node 2. The routing algorithm calculates that the message is more likely to reach D if it is sent through node 2, so node 2 receives the message. Later on, node 2 encounters D and passes on the message.

¹ <https://twitter.com>

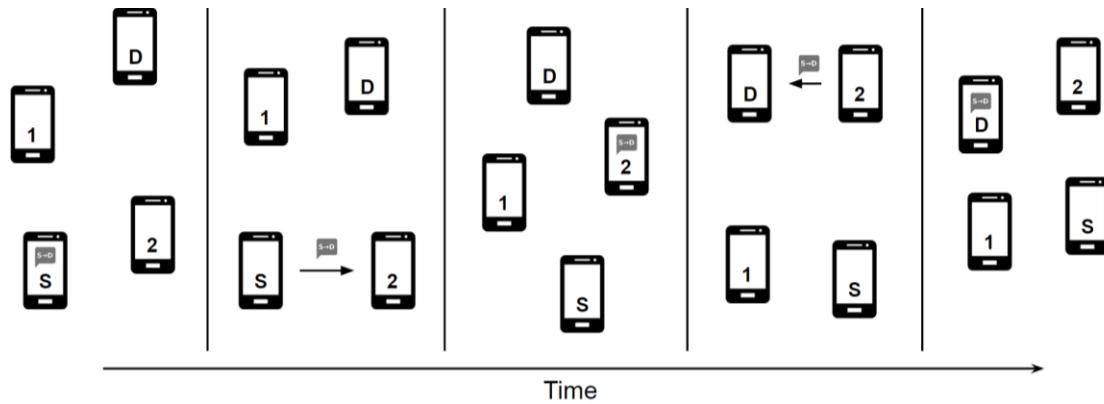


Figure 1: Example of message distribution in an opportunistic network.

Opportunistic network nodes must store and forward messages to other nodes whenever possible. Messages often take a significant amount of time to reach their destination: this makes it much more difficult to solve problems that have been solved in conventional connected networks (security, routing etc.), which assume near-instant message transfer. Because of the predictable nature of human behaviour, much research has been done to improve routing algorithms in pocket switched networks.

2.3 ROUTING ALGORITHMS

Routing messages in opportunistic networks is a non-trivial task because it is impossible to predict connections with any certainty.

Opportunistic networks can be viewed as a constantly changing graph. For this reason, many opportunistic routing algorithms are similar to graph search techniques. However, because the graph is constantly changing and is not necessarily random, such techniques are not necessarily the most effective [5].

For demonstration purposes, we will use the network shown in Figure 2 which is trying to pass a message from source S to destination D. The colour of a node represents some heuristic measure of utility (distance to D), where darker nodes are closer to the destination and lighter nodes are further away. Lines are drawn between nodes which encounter each other regularly. I have done my best to mirror the complex and unpredictable encounters in an opportunistic network.

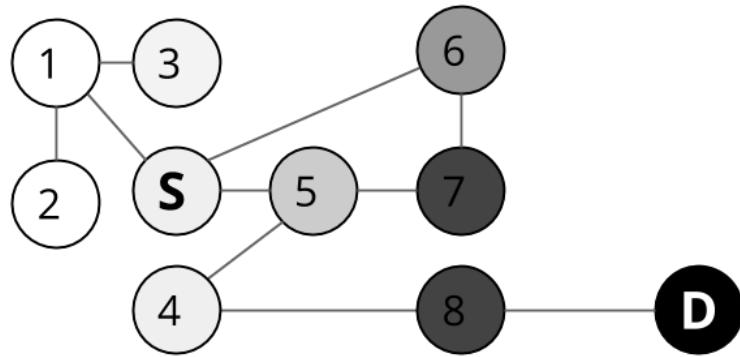


Figure 2: Example opportunistic network topology where connected nodes will encounter each other regularly and colour represents a heuristic measure of utility.

2.3.1 Context-Based Routing

Conventional routing algorithms work by passing a single packet between routers until it reaches its destination. When this approach is taken by an opportunistic network, it is known as context-based routing.

Context-based routing is a form of greedy best-first search, where a single message is continually passed to the node most likely to reach the destination (the node with the highest utility). There are a variety of methods to compute the utility of a node, including CAR [6] and MobySpace [7].

In Figure 3, a single copy of the message is always passed to the neighbouring node closest to the destination. Five nodes are visited by the message (6-7-5-4-8), which is far from the optimal path (three nodes – 5-4-8) but correctly avoids node 1. Node 6 is chosen over node 5 because it is (wrongly) perceived to have a better utility than node 5.

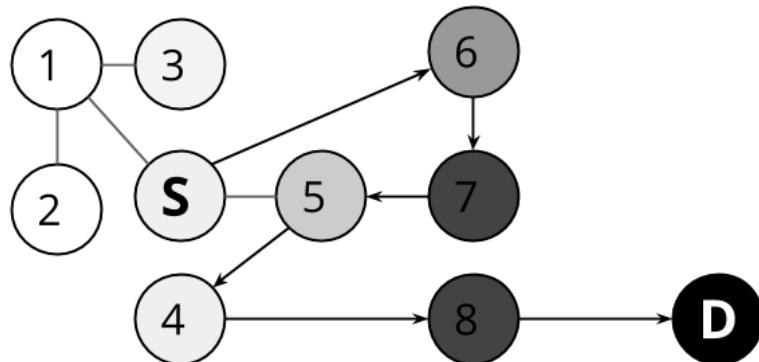


Figure 3: Message path in context-based routing algorithm. Note that node 6 is wrongly chosen because the heuristic utility function gives it a better utility.

As with all heuristic algorithms, there are pathological cases – if S was closer to D than node 4 then the message would get stuck in an infinite loop (S-6-5-7-S-6...). Similarly, if the only path to D was through node 1, the algorithm would never find it. While it is not guaranteed to find the optimum path (or any path) to the destination, context-based routing algorithms use very few resources as the message is never copied, so is a good choice for networks with a good heuristic for node utility (which is why a similar technique is used in conventional networks with a predictable structure).

Context-based routing algorithms are very susceptible to blackhole attacks, where a node (known as a blackhole) accepts a message but refuses to pass it on. In this case, the message will never reach its destination. There are heuristics for identifying blackholes such as IRONMAN [8], but none of them can totally protect the network against blackholes, because the only way to identify a blackhole node is for it to drop a message.

If a message has multiple destinations, the sender will have to explicitly create a copy of the message for every destination – this is impossible if there are an unknown number of destinations (e.g. in a microblog).

2.3.2 Dissemination-Based Routing

Dissemination-based routing is a paradigm designed to offset the weaknesses of context-based routing in opportunistic networking environments. Instead of passing one copy of the message around, nodes in a dissemination-based network create copies of the message. This is akin to a breadth-first search of the network – if the sender is the root node, copies of the message are passed to the children (nodes who connect directly to the sender) of the sender, who pass it on to their children in turn and so on.

This approach effectively searches multiple possible paths at the same time. If messages are passed on to every node that is encountered, it is guaranteed to find the shortest path if one exists.

While this is a very useful property, it means that every message will be passed to every node on the network and every node must store every message ever sent. This approach is not scalable because nodes in large networks would have to store a large number of messages. For this reason, dissemination-based routing algorithms normally include a heuristic for deciding when a message should be passed on.

This approach also creates another problem – when a message reaches its destination, all copies of the message are now pointless. But how does a node know that it can discard its copy? At present, the best known solution is some heuristic which will discard messages which are likely to have reached their destination.

2.3.3 Comparison of Techniques

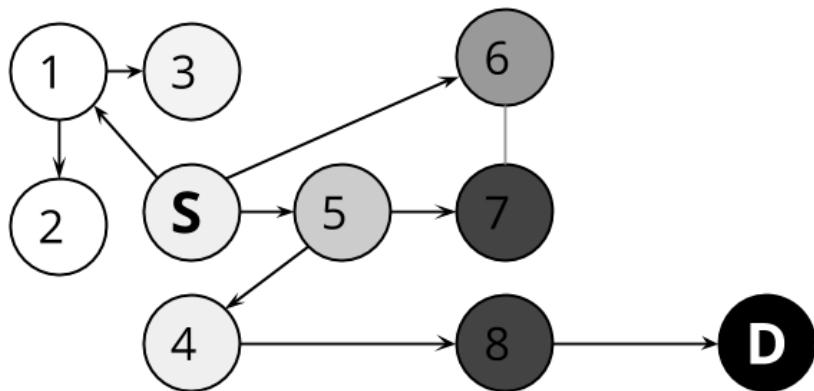
Given a good heuristic for node utility, context-based routing algorithms are much more efficient than their dissemination-based counterparts as only one copy of a message is stored at any point in time, and the message is removed from the network when it reaches its destination. However, they are very susceptible to blackhole attacks and often fail to find an efficient path to the destination as the structure of an opportunistic network is very chaotic and cannot be predicted well by heuristics. In addition, they cannot be used in networks with an unknown number of destinations because every message can only be delivered to one sender.

Dissemination-based counterparts are much better at finding a path to the destination and are much less susceptible to blackhole attacks, but do require every node to store many more messages than a context-based algorithm would.

2.3.4 Epidemic Routing

Epidemic routing is the simplest form of dissemination-based routing [9]. Copies of the message are passed at every opportunity until it saturates the network – a method guaranteed to reach the recipient. This is often likened to the spread of a virus.

In the example below, the message reaches the destination by the shortest path, but is also passed to every other node in the network. This approach is undesirable in most cases because of the high resource usage, which is why more advanced dissemination-based algorithms are often used.



*Figure 4: Message propagation using an epidemic routing algorithm.
The message is sent to all nodes.*

2.3.5 PRoPHET

The Probabilistic Routing in Intermittently Connected Networks (PRoPHET) algorithm [10] is a common dissemination-based algorithm where messages are only passed on to nodes with a higher utility than the current node.

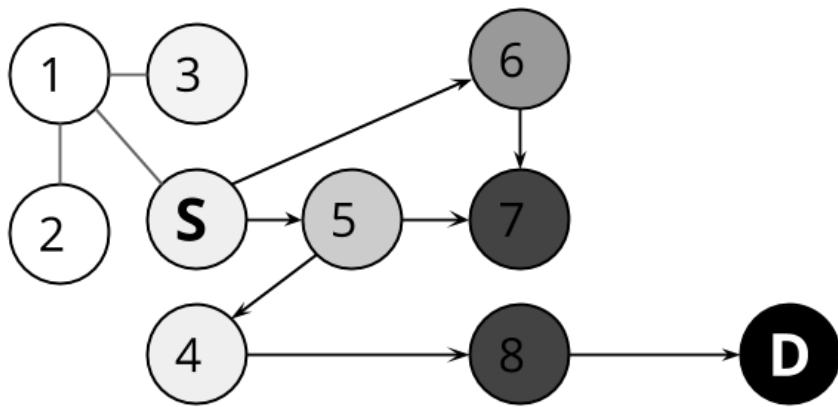


Figure 5: Message propagation using the PRoPHET algorithm. The message is not passed to node 1 because the utility of node 1 is lower than the utility of S.

In PRoPHET, a node's utility is defined as the probability that it can pass a message on to the destination. The destination's utility is 100%, and every other node's utility is slightly less than the utility of recently encountered nodes – so nodes which have recently encountered the destination have a high utility and nodes which are a long way from the destination have a low utility.

PRoPHET assumes that, if a node has been encountered recently, it is likely to be encountered again. Messages follow a path which would recently have brought them to the destination – which is likely to work again.

PRoPHET is considered to be a favourable routing algorithm as it will normally find a good path while only sending the message to a subset of the network, making it much more scalable than naïve epidemic routing.

2.3.6 Bubble RAP

Data collected on PSNs suggests that algorithms that treat routing as a generic graph search problem are often unsuited to PSNs [11]. Bubble RAP [12] works on the idea that a social connections graph has a tree like structure, where closely related nodes form a community (a bubble) and highly connected, high ranking nodes near the root can forward messages between these communities. In order to send messages to a different community, the message is sent towards the highly connected nodes with a high ranking, and then towards the destination community and sub-community and, eventually, the destination node.

This has been shown using the data collected from Haggle to be much more effective than algorithms like CAR and PRoPHET for sending messages to a known recipient [12].

2.4 SECURITY

Security is a very hard problem in opportunistic networks. Conventional network security relies on fast communication between nodes (allowing complex handshakes) and a central trusted accessible server. These approaches are infeasible for an opportunistic network as it takes so long to deliver a message. Instead, different paradigms are needed to enforce security in opportunistic networks.

Security can be compromised in an opportunistic network by controlling a node or by intercepting messages during transmission. Common attack types include:

- Sybil attacks: impersonating another node in order to send messages that appear to be from that node or to receive messages intended for the node.
- Majority attack: by controlling a large number of nodes, an attacker can control a network which assumes that the majority of nodes can be trusted.
- Eavesdropping: gathering information such as message metadata to discover private information such as message contents and user location.
- Denial of Service: saturating the network with unwanted messages.
- Black hole attack: failing to pass on messages to either reduce resource usage or as part of another attack.

2.4.1 *Trust-Based Security*

Trust-based security mechanisms depend on generating a list of trusted or untrusted nodes. This is commonly based on trusting connections within a social network [13] or distrusting nodes exhibiting strange behaviour [8].

2.4.2 *Cryptographic Security*

Conventional cryptographic security mechanisms often use a single trusted authority to verify the identity of a user and distribute certificates.

This is infeasible in a scalable opportunistic network because as the network grows, the time to communicate with the central server increases. Some mechanisms, like the one proposed by Shikfa et al [14] do use a central server, but only require it to be available for nodes joining the network. Other mechanisms split the responsibility over a number of nodes. Mechanisms for doing this in a distributed manner require some level of trust in network nodes. For example Capkun et al's approach [15] does this by building a graph of certificates determining who trusts who. Abnormalities in the trust graph may indicate foul play.

2.4.3 Asymmetric Key Cryptography

Asymmetric key cryptography is a commonly-used method for encrypting data and signing it to verify its origin and integrity. Every user has two cryptographic keys – a public key known to the world and a private key (also known as a secret key) known only to them.

The public key can be used to encrypt a message so that it can only be decrypted by the matching private key. Since only the intended recipient knows the private key, only they can decrypt it.

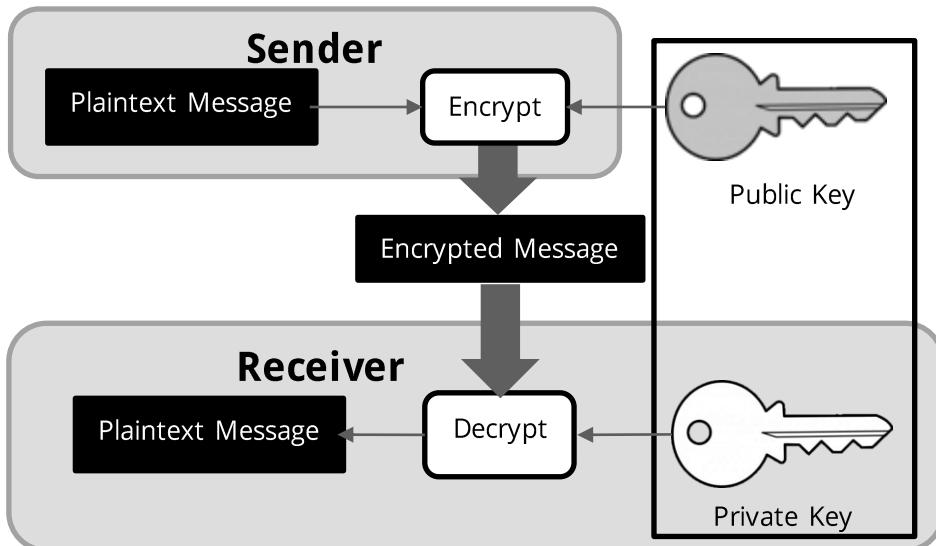


Figure 6: Message encryption/decryption using asymmetric keys. The message is encrypted using the receiver's public key. It can then only be decrypted using the matching Private key.

Similarly, the private key can be used to create a message signature. This signature can be used to verify that the message creator knows the private key and that the message has not been modified after it was created (data integrity).

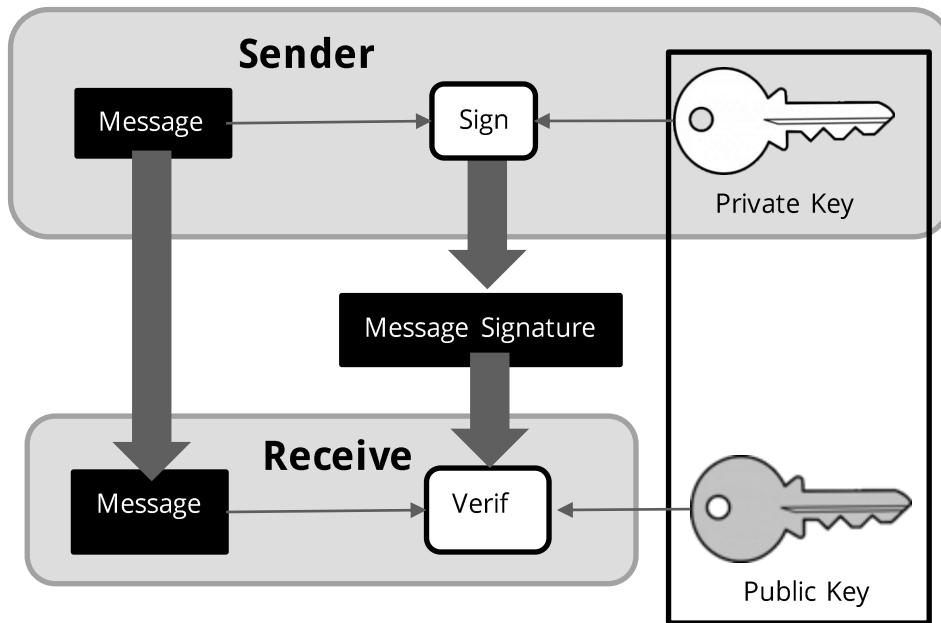


Figure 7: Message signing/verification using asymmetric keys. The message is signed using the sender's private key. The signature is verified using the sender's public key. The receiver now knows that the message must have been signed using the matching key.

2.4.4 Identity-Based Cryptography

Identity-based cryptography (IBC) is an increasingly common form of asymmetric key encryption where a user chooses a short unique identifier (such as an email address) as their public key, and a private key is generated by a central private key generator (PKG) and sent securely to the user [16].

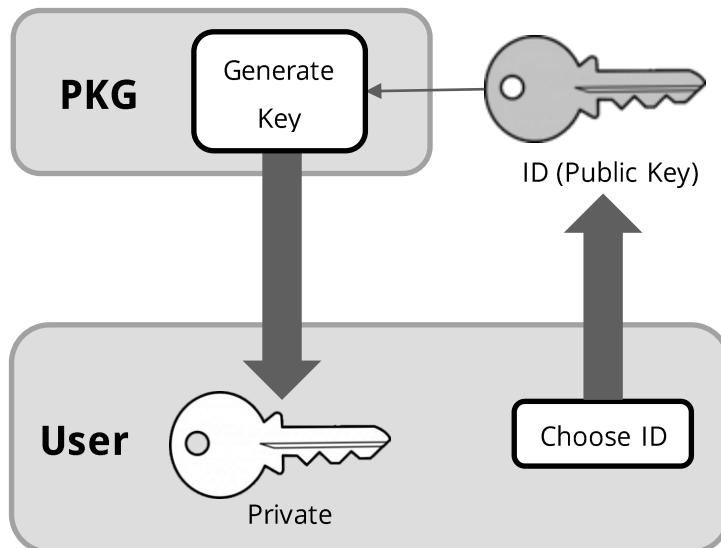


Figure 8: Private key generation in an ID-based scheme. The PKG generates a private key for the user's ID and sends it to the user.

IBC algorithms capable of signing messages are referred to as ID-based signature (IBS) algorithms.

When applied to opportunistic networks, this approach has similar problems to certificate-based approaches - a central server is needed. Some security frameworks assume that there is a central PKG that can and will be accessed occasionally [17]. Others split up the PKG into multiple nodes, some of whom must collaborate to generate a private key [18]. The advantage of the identity-based approach is that it is no longer necessary to distribute public keys – it is still necessary for the PKG to distribute private keys, but this can happen less frequently (e.g. when a central PKG on the Internet is available).

2.4.5 Hierarchical Identity-Based Cryptography

Hierarchical identity-based cryptography (HIBC) is a form of IBC where any node with a private key can generate a private key for another node (known as delegation) [19]. This creates a tree hierarchy where the central PKG is the root. A user's public key is a combination of their own ID and the public key of the node they received a key from – in other words, a user's public key is the path from the user to the root node (the PKG). For example, the central PKG generates a private key for a user ID A. User A can now delegate a private key for users B and C with public keys A→B, A→C respectively.

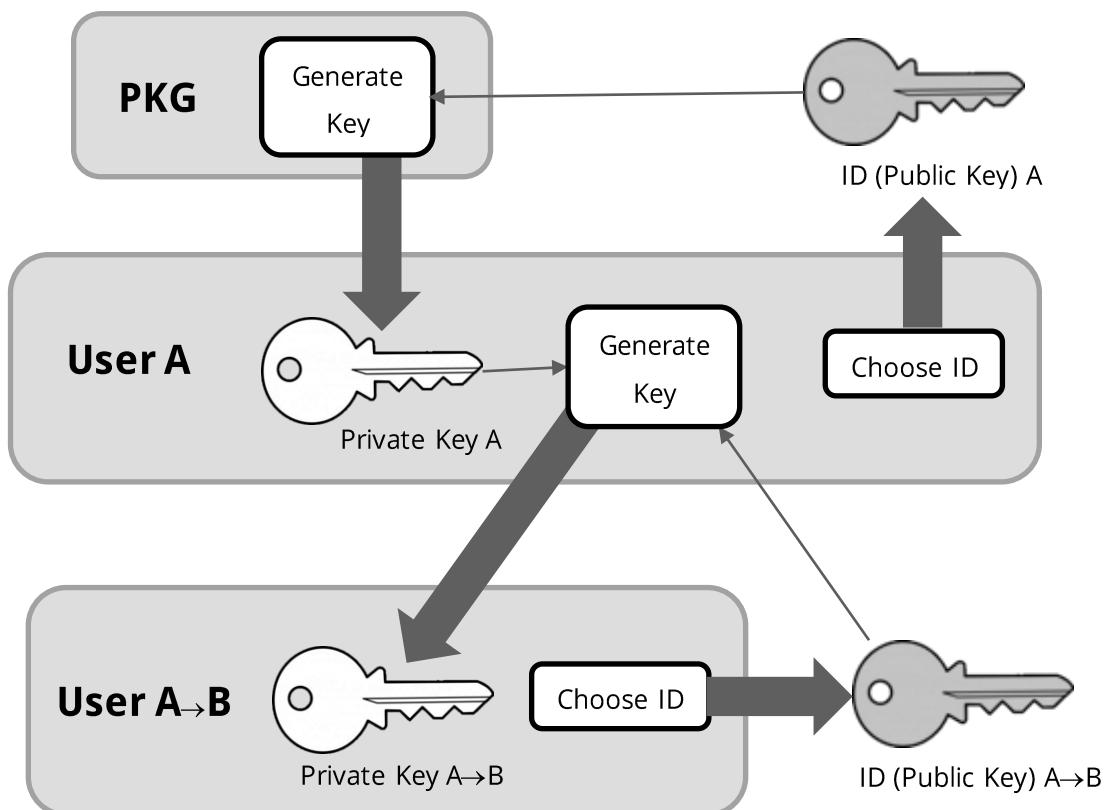


Figure 9: Private key delegation in a hierarchical ID-based scheme.
User A creates a private key for user A→B and sends it to the user.

Again, HIBC schemes which can sign messages are referred to as HIBS (hierarchical identity-based signature schemes).

There have been no known applications of HIBC to opportunistic networking, although Seth & Keshav present a working solution for delay tolerant networks [20] which could be adapted for opportunistic networks.

2.4.6 SSNR-OSNR Obfuscation

Parris & Henderson introduce two mechanisms for obscuring a user's list of friends while making it accessible for routing purposes [21]. The same ideas can be applied to similar data being used for routing.

In statisticulated social network routing (SSNR), random friends are added to the friends list when it is advertised to other nodes. This does not affect the routing algorithm much because, in the worst case, a few unnecessary messages are passed on. This introduces plausible deniability – if user A is in my advertised friends list, it does not imply that user A is in my actual friends list. By occasionally omitting a friend from the advertised friends list, we can prevent an attacker from inferring that I do not follow a user just because they are not in my friends list.

In obfuscated social network routing (OSNR), the friends list is encoded in a Bloom filter – a small fixed size data structure representing a set which can tell if an object is probably in the set or definitely not in the set [22]. Again, this introduces plausible deniability – an attacker cannot derive my friends list from my advertised friends list. Also, if an attacker wants to derive a set of my possible friends, this would require a brute-force attack where they check if every possible username is in the filter.

Combining these techniques makes it significantly harder for an attacker to ascertain a user's friends list and, even then, the list will not be 100% accurate.

2.5 SIMILAR PROJECTS

There are a number of projects utilising opportunistic networks and similar technologies. I have listed the most relevant ones here.

2.5.1 Haggle

Haggle² [5] - a pocket switched network designed to run on smartphones - is one of the largest opportunistic networks. There are implementations for a number of clients including Android³ and Windows Mobile.

² <http://www.haggleproject.org>

³ <http://play.google.com/store/apps/details?id=org.haggle.kernel>

By monitoring use of the platform, the authors discovered trends in inter-contact times and contact durations, showing that conventional opportunistic routing algorithms are poorly suited to real world pocket switched networks [4].

2.5.2 FireChat

FireChat⁴ is a smartphone application used for off-the-grid messaging between nearby users. It has been used to circumvent government restrictions in Iraq [23] and during the Hong Kong protests [24].

However, the app mostly relies on an Internet connection and its simple protocol is insecure [11] and unable to implement the store-and-forward functionality of a proper opportunistic network.

2.5.3 Daknet

Daknet [25] is an opportunistic network for rural villages in India. Each village has a kiosk which can connect wirelessly to Mobile Access Points (MAPs) on buses and cars. This access point travels between villages and towns, carrying communications between them. As well as carrying communications between villages, a MAP can also relay requests to download something from the Internet from an Internet access point.

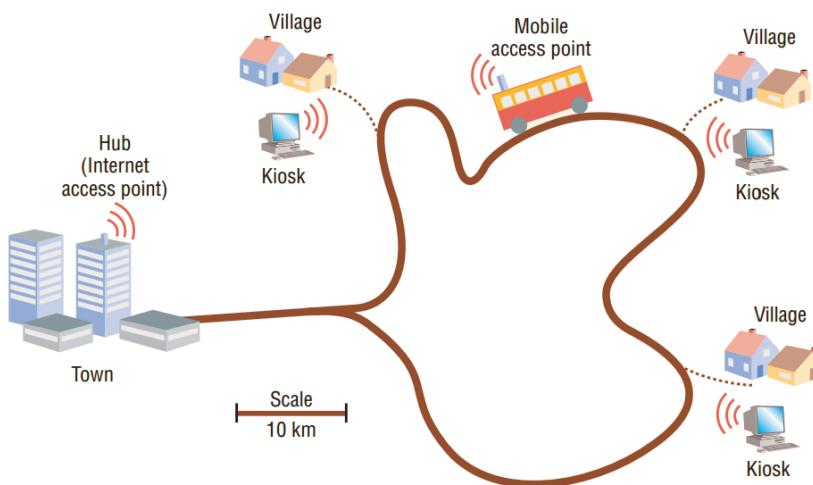


Figure 10 [25]: Physical transport, in this case a public bus, carries a mobile access point (MAP) between village kiosks and a hub with Internet access. Data automatically uploads and downloads when the bus is in range of a kiosk or the hub [25].

⁴ <http://opengarden.com/firechat>

2.5.4 SWIM

The Shared Wireless Infostation Model (SWIM) is a proposed opportunistic network to monitor whales [26]. Small nodes are attached to the whales, which record data such as location and interactions with other whales. Connected nodes transfer this data between each other. Whenever data is transferred to a base station (the paper proposes using seabirds), it can be collected and stored.

SWIM makes it possible to get information about a whale without having to physically find it to read data from its sensor – data about the whale will have been relayed to sensors on other whales. This is a perfect example of the power of opportunistic networks in an environment with very limited connectivity.

3 USE CASES

There are lots of cases where an Internet connection is unavailable and opportunistic networks could be used to enable communication. Below are two very different scenarios which could both benefit from an opportunistic network. OMiN is designed to work in these scenarios but to be flexible enough to be applied to other scenarios.

3.1 DISASTER AREA

A tsunami has wiped out all communications infrastructure in the area and injured a lot of people. Our opportunistic network is the quickest way to contact medical teams and inform them of injured people who need help. In this scenario, most people have smartphones and will be moving about regularly. Alex is injured and must contact the nearest free medical team so that they can help her. She uses her smartphone to publish a message with her location and her status to all nearby nodes. The message is distributed in this manner until it reaches Doctor Brian. He sends a reply message to indicate that help is coming and goes to help Alex.

3.2 METRO

Many people use underground railways to commute to work. Because it is underground there is no mobile phone signal and no way of connecting to the Internet. Commuters use the OMiN network as a social network where conventional Internet-based networks will not work. As well as messages from individual users, Internet-connected base stations at subway stations send travel updates from the transport operator and breaking news from news sites to commuters and message distribution nodes on the trains. Commuters choose which people and updates they wish to receive messages from. Messages from unpopular users will not spread far but a number of users will become very popular for their amusing or novel messages – these messages will reach the whole network.

4 THREATS

From the use cases, it is possible to construct a model of potential motives for attack and attack vectors. This is shown by a threat tree – a representation of possible goals for an attacker along with a breakdown of techniques the attacker could use to reach that goal.

4.1 DISASTER AREA

In a disaster area, people tend to act altruistically. Therefore they are unlikely to attempt to subvert the network. However, people may act selfishly by attempt to conserve battery or memory space on their phone.

- Goal: Selfishly reduce personal resource usage
 - Avoid accepting messages or passing them on (black hole attack)

4.2 METRO

In this scenario, an attacker could disrupt the transport network by sending messages that appear to be travel updates from the transport operator or breaking news from news sites. It is very important to ensure that these popular users cannot be compromised.

- Goal: Spread false information appearing to be from a reputable source (Sybil attack)
 - Exploit trust-based security mechanism (if it exists)
 - Discover private key of reputable source
 - Hack source
 - Hack PKG
- Goal: bring down whole network
 - Prevent all messages from being disseminated
 - Control most influential network nodes (majority attack) and turn them into black holes
 - Gain control of most message dissemination nodes in trains
 - Gain control of most base stations in subway stations
 - Flood the network with messages (Denial of Service)

4.3 ALL THREATS

Considering these scenarios and threats, the network must protect against the following attacks:

- Sybil (impersonating another user)
- Message modification
- Black hole (failing to pass on messages)

- Denial of Service (overloading the network with messages)
- Snooping (using metadata to infer private information)

5 OBJECTIVES

In order to implement an opportunistic network for the use cases, the following objectives should be met:

5.1 PRIMARY OBJECTIVES

- Design and implement protocol for discovering nodes in close proximity and passing messages and necessary metadata between them.
- Create core library to manage message storage and routing.
- Implement epidemic routing algorithm to send messages to all available nodes.
- Design routing algorithm using user metadata to route messages while disguising message content and metadata.
- Design scheme to allow verification of the origin and integrity of a message.

5.2 SECONDARY OBJECTIVES

- Create user interface.
- Implement more advanced routing algorithm.
- Implement message verification scheme.
- Evaluate impact of message verification scheme.
- Evaluate performance of the implemented routing algorithms.

5.3 TERTIARY OBJECTIVES

- Compare real world vs. simulated performance of the routing algorithms.

6 REQUIREMENTS SPECIFICATION

From the objectives, use cases and threats, I have formulated a set of requirements which the system should meet in order to fulfil the objectives and be useful in the given use cases.

6.1 USER REQUIREMENTS

6.1.1 *Functional Requirements*

- High: The user shall be able to create a unique identity.
- High: The user shall be able to send plain text messages to all others who follow the user.
- High: The user shall be able to ‘follow’ any user and receive messages sent by that user.
- High: The user shall be able to send messages without requiring an Internet connection.
- Low: The user shall be able to send messages via the Internet to Internet-connected nodes.

6.1.2 *Non-Functional Requirements*

- High: The network shall be usable on a large number of mobile devices.
- Medium: The user shall be able to be confident in the origin and integrity of a message.
- Low: The user shall be able to use the network with minimal training.

6.2 SYSTEM REQUIREMENTS

6.2.1 *Functional Requirements*

- High: The system shall work on portable electronic devices such as smartphones or tablets.
- High: The system shall allow creation of user identities with a unique cryptographic identity.
- High: The system shall automatically connect to nearby nodes and pass on relevant information.
- High: The system shall pass on messages until they reach their destination.
- Medium: The system shall ensure that messages cannot be modified in transit or that any such modifications can be detected.
- Medium: The system shall ensure that nodes cannot send a message that appears to be from another user.
- Medium: The system shall restrict the size of the message store.
- Medium: The system shall protect against Sybil attacks.
- Medium: The system shall prevent messages from being modified while in transit.

- Medium: The system shall protect user metadata from all other nodes.
- Medium: The system shall be scalable to an arbitrary number of nodes.
- Low: The system shall block attempts to prevent message propagation.
- Low: The system shall have mechanisms to mitigate Denial of Service attacks.

6.2.2 Non-Functional Requirements

- High: The system shall work in an unstructured environment with random encounters between nodes.
- High: The system shall not require a connection to any other network (such as the Internet).
- Medium: The system shall be robust and able to continue functioning when it encounters an unexpected state such as a malfunctioning or untrustworthy node.
- Medium: The system shall minimise the number of messages lost before they reach their destination.
- Medium: The system shall route messages effectively given a semi-predictable set of encounters between nodes.
- Medium: The system shall deliver messages as quickly as possible.

7 DESIGN

A significant amount of time was spent at the start of the project researching current technologies and carefully designing the routing algorithms and security systems to fulfil the requirements.

7.1 SOCIAL STRUCTURE

Both use cases revolve around users viewing short messages sent by others. For this reason, it seems practical to structure the network as a microblogging service similar to Twitter, where users send short messages and are “followed” by other users.

7.2 SECURITY SCHEME

7.2.1 *Ensuring Message Integrity*

Steps have been taken to prevent Sybil attacks (impersonation of users), message modification and majority attacks. Many network protocols [12] [13] use heuristic algorithms to determine which nodes in a network to trust. This is often good enough for small networks, but heuristic algorithms cannot guarantee security – they are often based on detecting attackers after their first offence (attackers will not be caught if they change their identity when they are detected) or trusting the majority of nodes in a network (vulnerable to a majority attack). For these reasons, I believe that a more secure approach is necessary for the Metro use case (a large network with lots of possible attackers).

I have chosen to take a cryptographic approach where users use an asymmetric key pair to sign messages and verify their origin and integrity. There must be some mechanism for associating a user’s public key with their identity and distributing this throughout the network. This must take place without any handshakes – where a message is sent over the network to a node which then replies over the network – because of the high latency of opportunistic networks (this approach is often used in conventional networks where asymmetric keys are used to securely negotiate a shared secret key). This cryptographic approach is much more secure than a trust based approach, but exposes two attack vectors which allow an attacker to assume a user’s cryptographic identity:

- Discovering a user’s private key.
- Manipulating the public key distribution mechanism to associate a user’s identity with an attacker’s public key.

In a normal asymmetric key cryptography, a user’s private key is generated by the user’s device and never leaves the device – it is secure from most attacks (except from hacking a user’s device, which would compromise the user in any network). However, most

cryptographic opportunistic networks use a trust-based approach to distribute public keys [15]. I have already shown that trust-based approaches are vulnerable to attack, and there is no known method for securely distributing conventional public keys in an opportunistic network – I suspect that it is impossible. Therefore, conventional asymmetric key cryptography within an opportunistic network can never be cryptographically secure.

OMiN will use ID-based asymmetric key cryptography, where a user's username is also their public key. These usernames are short and memorable enough to be distributed by word-of-mouth in the same way that email addresses are, removing the need for a public key distribution system.

In ID-based cryptography, private keys are generated and distributed by a central PKG. This must be done in a cryptographically secure manner in order to prevent attackers discovering a user's private key. This must also be done in an opportunistic network environment, where there is no Internet connection and messages could take a long time to reach their destination, if they arrive at all. There are a number of solutions to this problem:

- Seth & Keshav [20] propose using USB drives to distribute one-time keys which are used to securely request a private key from the PKG over the network. However, their solution is aimed at delay tolerant networks where round trip times are more reasonable and messages are more likely to be received – it will not scale well because it requires sending a message to the central PKG and receiving one back.
- Kong et al [18] use multiple PKGs where one or more PKG must collaborate to generate a private key. This removes the bottleneck of a central server, but requires more PKGs to be created and managed as the network scales.
- The simplest solution, taken by Kamat et al [17], is to assume that every node can access the Internet when they start using the network – at this point they can securely communicate with the PKG over the network to receive their private key.

My cryptography scheme use a version of Kamat et al's scheme [17] with a modification to allow for the case where the PKG is not accessible: instead of IBC, we use a HIBC scheme where every node with a private key is capable of delegating private keys to other users.

If user A cannot access the PKG, they can still be authenticated by the first user they meet, user B (giving them the identity B→A). Every user must either be authenticated by the PKG or another user, so there will always be a chain back to the master PKG. In this way it is possible to create a chain of key generators where the master PKG (accessible via the Internet) delegates PKG responsibilities down the chain (Figure 11).

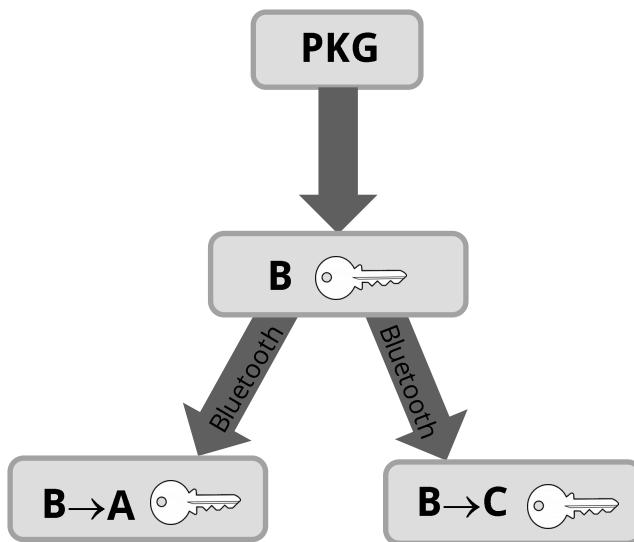


Figure 11: Key delegation chain. The PKG creates a private key for B and sends it over the internet. B can now delegate a private key to $B \rightarrow A$, $B \rightarrow C$ and so on.

It is wise to keep this chain as short as possible because a node's parents and ancestors are, by definition, capable of deriving their descendant's private keys. Again, this introduces the notion of trust – a node must trust its parents and ancestors not to use its private key.

It is impossible to completely eliminate the notion of trust in IBC – every node must trust the PKG – but it is possible to reduce the number of nodes that must be trusted. In this scheme every user can assume multiple identities, each with their own private key: Figure 12 has a user with identities $B \rightarrow A$ and $C \rightarrow A$ for example. Messages are signed with every one of the user's private keys – each message now has a signature for each of the user's identities. Now an attacker must discover all of the user's private keys to assume a user's cryptographic identity and send messages on their behalf. In the example shown in Figure 12, both users B and C must collaborate to compromise the user's identity. As users gain more private keys, they become harder to compromise. The ultimate in security is to have a private key from the central PKG – falling back to the IBC scheme, which only requires trust in the PKG. In this way, it is possible to minimise the number of parties which have to be trusted in all cases where at least one node has had access to the Internet at some point.

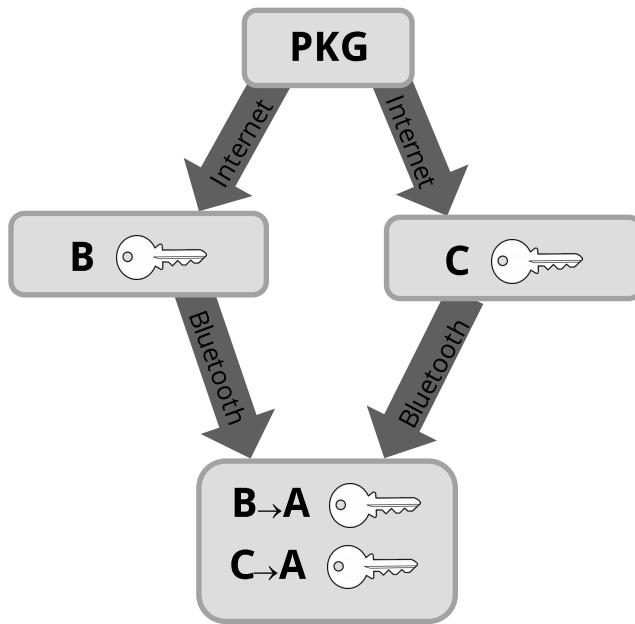


Figure 12: Private key delegation chain where users can have multiple IDs and private keys. Both B and C must now collaborate to discover $B \rightarrow A$ / $C \rightarrow A$'s private key set.

Because of the cost of cryptography (see section 11.1 on page 42), nodes should seek a small set of IDs while reducing trust (i.e. keeping short ID chains and using many different ancestors).

7.2.2 Alternative to HIBS-Based Approaches

In practice (see section 8.4.3 on page 35), there is no useable implementation of HIBS – although such a scheme is presented in theory by Yuen & Wei [27]. It is still possible to use Kamat et al's approach [17] (a central PKG accessed over the Internet) but it is desirable to deal with the case where the nodes cannot access the Internet to obtain their private key. Users who cannot access the Internet can send unsigned messages, but there is no foolproof way of determining the message's origin and authenticity because any node between the sender and receiver could maliciously modify the message – they must all be trusted.

To reduce the number of nodes that must be trusted, nodes with a private key can sign the message on behalf of the sender, guaranteeing that it cannot be modified for the rest of its journey to the sender, as shown in Figure 13.

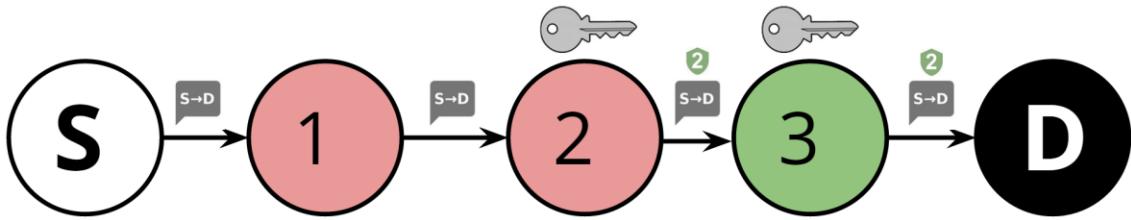


Figure 13: Distribution of an unsigned message from sender S to destination D. The message is vulnerable to nodes 1 and 2 until it is signed by node 2. Node 3 cannot modify the message.

It is possible to encounter multiple copies of an unsecured message that have been signed by different nodes. Since both copies are identical, it does not matter which version should be passed on. Nodes should choose a random copy to prevent attackers from taking advantage (e.g. if nodes always choose the version with the lexicographically lowest signature, an attacker could modify the message and use a lexicographically low signature to increase the chances that their version is chosen to be redistributed).

In the example in Figure 14, Node S sends an unsigned message intended for node D. Nodes 1, 2, 4 and 5 must be trusted because they are both capable of maliciously modifying or blocking the message. When nodes 2 and 5 receive the message, they sign it on behalf of S. Node 3 receives both signed copies and takes a random one – in this case, 2. This is then delivered to the destination, D.

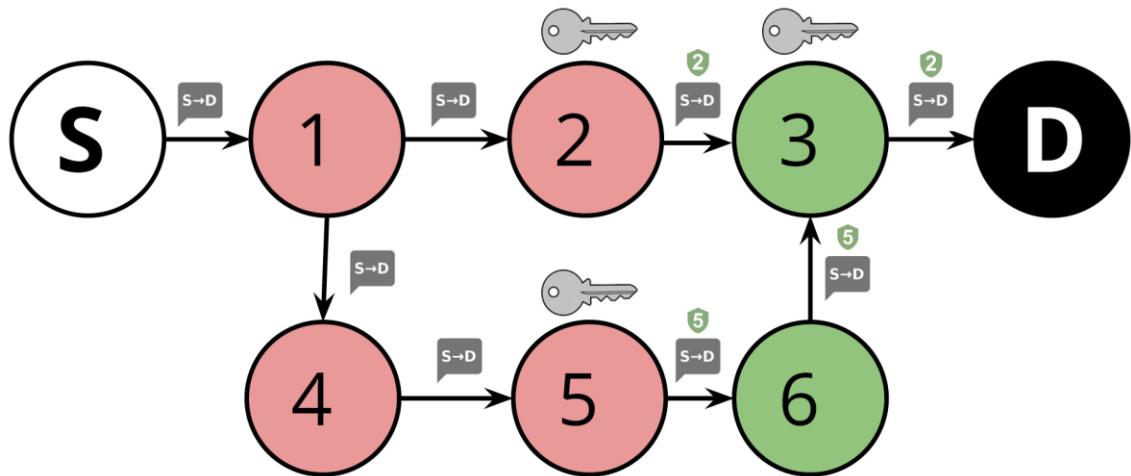


Figure 14: Distribution of an unsigned message from sender S to destination D. The message is vulnerable to nodes 1, 2, 4 and 5 until it is signed by nodes 2 and 5. Node 3 encounters multiple signed versions of the message and picks one at random.

7.2.3 Preventing Black Hole Attacks

A black hole attack is where a node fails to store or pass on a message. This can be done for selfish reasons (to reduce storage usage) or to prevent a message from being distributed (in a context-based routing scheme). Schemes such as IRONMAN [8] and

RADON [28] store metadata about recent connections in order to find nodes which are failing to pass on connections and decrease their reputation (for example; A sends a message through B then B connects to C but does not forward the message as expected: when A later connects to C they can figure out that B is a black hole). Disreputable nodes will not be sent new messages, effectively isolating them from the network.

OMiN uses a dissemination-based routing algorithm where many copies of the message are spread through the network. While black hole attacks are a serious threat to context-based routing (a single black hole can stop a message), it is a less significant threat in our network - many black holes are needed to prevent a message from being disseminated. For this reason, protection against black holes is a low priority in the network and has not been implemented. If it were to be implemented, an algorithm similar to IRONMAN or RADON would be used to detect and punish black hole nodes.

7.2.4 Preventing Snooping

Snooping is the use of metadata (like location) to infer private information (like a user's identity). The routing algorithm has been designed to use very little metadata - any metadata that is used is disguised using the SSNR-OSNR algorithm [21].

7.2.5 Protecting the PKG

The PKG is the only party which must be trusted by all nodes. If it is compromised then the attackers could gain access to the master private key, which could be used to generate private keys for all users and compromise the whole network.

For this reason, the PKG must be built securely. It must be hosted on a secure system, transfer private keys securely (using SSL) and be invulnerable to injection attacks and unexpected input. Section 8.5 on page 35 describes the measures taken to build a secure PKG.

7.3 ROUTING

A routing algorithm takes all available information about a message and uses that information to decide where to send it. In trust-based networks, this information is freely available to all trusted nodes (recipients, previous paths etc.). However I have opted for a more secure model where all nodes are considered untrustworthy – information must be hidden or obscured while still allowing a routing algorithm to use it.

The Bubble RAP algorithm has been shown to be very effective for pocket switched networks [12] but it relies on sending a message to a known destination. OMiN cannot use this approach because messages can be sent to multiple, possibly unknown, users.

For the same reason, context-based routing algorithms cannot be used because every message must have a single, known destination.

The PRoPHET routing algorithm [10] is a very good dissemination-based routing protocol which can be adapted to minimise the use of sensitive metadata. In PRoPHET, all nodes calculate and advertise the probability that they will be able to deliver the message to its destination based on previous encounters. If node A has communicated with the destination recently, it has a high probability. When node B communicates with node A, it calculates that its probability is lower than A's.

OMiN messages are microblogs without a specific destination – they are sent to all users following the sender. Instead of distinguishing between destination nodes and carrier nodes, all destination nodes can advertise that they have a 100% probability. The SSNR-OSNR algorithm [21] described in section 2.4.6 (page 16) describes a method of obscuring these probabilities in a way that attackers cannot definitively decide who a user is following.

In future, this could be combined with Bubble RAP's idea of communities to increase performance.

7.4 MESSAGE BUFFER EVICTION

When the message buffer is too large, it must evict a message. Ideally, this message will already be close to the destination. Nodes cannot know this information, but they can use heuristics to infer it - messages that have been forwarded to many nodes are likely to be widely distributed throughout the network and are therefore closer to the destination than the current node is. The message that has been forwarded the most should be evicted.

When a message is evicted, a node must ensure that it is not received again - this could result in loops where a message is forwarded, evicted re-received and re-forwarded. Nodes should use a bloom filter - a small fixed size data structure representing a set which can tell if an object is probably in the set or definitely not in the set [22]. When a message is seen, it should be added to the bloom filter. Nodes will only accept messages if they are not in already in the bloom filter - they have definitely not been seen.

In practice, this scheme was not implemented because it relies on a routing algorithm which does not send messages indiscriminately to every node (i.e. is not epidemic). I instead chose to implement the simpler heuristic of evicting the oldest message.

7.5 DATABASE DESIGN

Every node must store a record of messages and users encountered. The network can be seen as a way to synchronise these records – when a node is encountered, messages and other relevant metadata are taken from the database, converted to the JSON format and passed to the node, which deserialises the relevant messages and metadata and stores it in its database. This process encompasses the majority of the work of the network nodes.

Nodes store whether they are following a **user**. Every user can have multiple **user IDs**, which consist of a username and possibly a parent user ID. The user of the node also stores a **secret key** (a private key) for their **user IDs**. A **message** consists of a message body, source user, time when it was sent from the source, time when it was last sent by the node, a security rating (verified, unverified, unsigned) and one or more source user IDs, each with a signature.

This is shown in a normalised entity-relationship diagram in Figure 15.

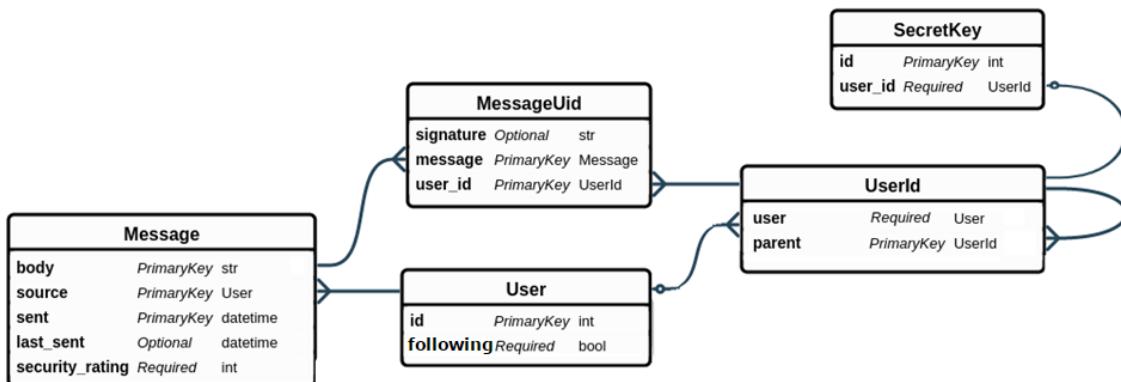


Figure 15: Database ER diagram showing the relationships between users, user IDs, messages and secret keys.

Note the use of the terminology “secret key” instead of “private key”. These are interchangeable, but while this report uses “private key”, the implementation normally uses “secret key” for the simple reason that it allows unambiguous use of the “PK” and “SK” acronyms.

7.6 USER INTERFACE DESIGN

Modern Android apps are encouraged to use Google’s Material Design language, a set of guidelines for creating intuitive, aesthetic interface. Android’s appcompat v7 library⁵ allows Material Design apps on most versions of Android. I chose to use it because it seems to be the best way to create an intuitive native app on Android.

There are six core things that the user must be able to do when interacting with OMNiN:

- Set username.
- View messages from followed users.
- Send a message.
- View followed users.
- Add followed user.
- Remove followed user.

⁵ <https://developer.android.com/tools/support-library/features.html#v7-appcompat>

The app can be divided into two tabs for messages and following users. The Android action bar, which otherwise has no use, is a good place to show and edit the username, which is always visible.

The messages tab can be implemented as a list of messages with a popup button for writing a new message. The ‘pull to refresh’ UI pattern would also be useful here.

The following tab can also be implemented as a list, with the ‘swipe to delete’ UI pattern to delete users and a popup button to follow new users.

Figure 16 shows a mockup of the user interface design.

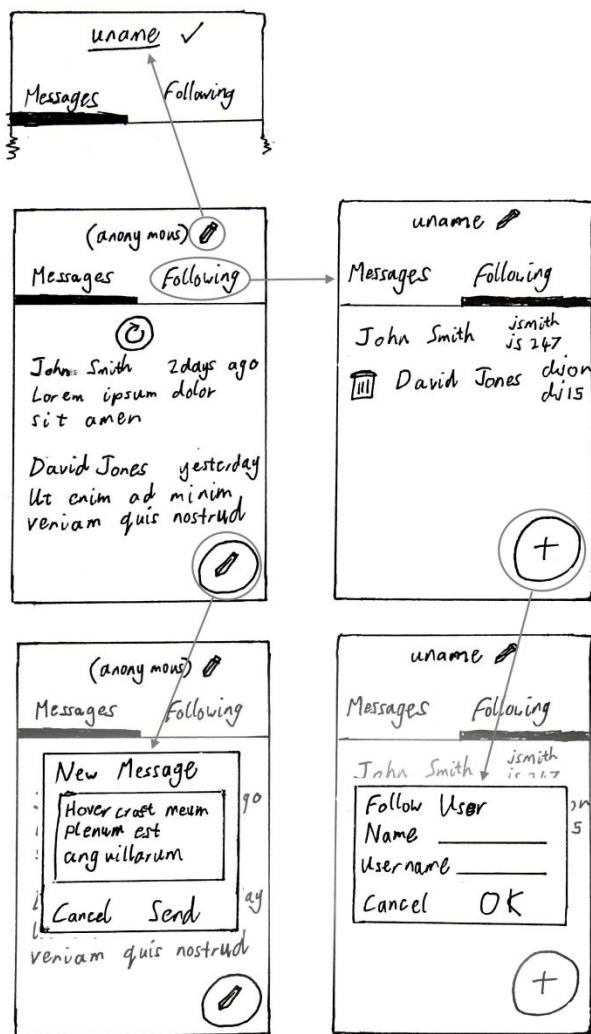


Figure 16: User interface design. Grey circles denote clickable elements and arrows point to the effects of clicking the element.

8 IMPLEMENTATION

There are a number of decisions to make about which technologies to use to implement the network.

8.1 MOBILE PLATFORM

One of the user requirements is that OMiN is useable on a large number of mobile devices. This not only increases the number of people who can use the network, but the network will work better as more people use it and more connections are formed. For this reason, it makes sense to use the most common platforms – Android and/or iOS. I chose to target the Android platform because there are fewer barriers to developing and releasing Android apps.

8.2 PROGRAMMING LANGUAGE

Android apps primarily use Java, but it is theoretically possible to use any language. However, only JVM languages (and C++ via the Android native development kit) have access to the Android application framework and libraries. Scala is a very flexible JVM language I have used to write android apps before, so I started writing the program in Scala. However, the build tools and libraries caused me a number of issues (see section 9.5.1 on page 39) which forced me to switch back to Java because it is so well supported.

8.3 MESSAGE PASSING MEDIUM

There are a number of methods for Android devices in close proximity to interact. The following mediums were considered:

- LAN communication - easy to implement but requires a LAN, which may not be possible for many use cases.
- Wifi-Direct - good range but requires that devices are not connected to a LAN and is only supported by Android API 14+ (about 93% of devices [29]).
- Bluetooth - well supported although limited connectivity.
- Bluetooth Low Energy - only supported by Android API 18+ (about 50% of devices [29]).

I have chosen to use Bluetooth because it is almost universally supported and does not rely on devices being connected or disconnected from a LAN.

8.4 CRYPTOGRAPHY SCHEME

The system should ensure that messages cannot be modified in transit and that nodes cannot send a message appearing to be from another user (a Sybil attack).

8.4.1 Disaster Area Scenario

In the disaster scenario, cryptography is not an important aspect – the main requirement here is that messages are distributed as quickly as possible. For this reason, the following attributes are desirable:

- Small public keys.
- No effect on message distribution.
- Users can start without having to contact a central server.

8.4.2 Metro Scenario

Cryptography is much more important in the Metro scenario, where the cryptography scheme should ensure that messages cannot be spoofed. The following attributes are important for the cryptography scheme:

- Verifiable source.
- Verifiable integrity.
- Unbroken cryptography scheme.

8.4.3 Cryptography Algorithm Choice

Any cryptography algorithm capable of signing a message can use the signature to verify the message's origin and integrity. Yuen et al. describe the ideal cryptography algorithm [27], a hierarchical ID-based scheme capable of signing messages. Unfortunately I cannot find an implementation of this, or any other HIBS. The best algorithm I can find with an existing implementation is described by Paterson & Schuldt [30], an ID-based signing algorithm. Paterson & Schuldt's algorithm has not received much attention so it is hard to determine how secure it is, but it should be sufficient for OMNiN. Section 7.2.2 (page 28) discusses a new design for the cryptography system taking into account that it is only possible to use an IBS scheme.

8.5 PKG SERVER

The PKG server is a single point of failure, so it has to be very secure. Existing web servers are more secure than building my own, so the OMNiN PKG server is written as a CGI script served by a web server. The server is written in Java because some code (like the implementation of the cryptography algorithm) must be shared between the server and client. Java is not particularly suited to CGI because a new JVM has to be created for every request (a relatively slow and costly operation), but there is a description of how to do it at <http://www.javaworld.com/article/2076863/java-web-development/write-cgi-programs-in-java.html>. The server maintains a list of users with private keys so it will only ever send the private key for a user once. Unix file locks are used to make sure that this file is only being read/written by one process at a time to prevent attackers taking advantage of race

conditions (e.g. if an attacker requests the same ID as a user at the same time as the user, both of them could receive the private key).

8.6 DATABASE LIBRARY

The app needs to store messages and other data in a database. In order to simplify the implementation, OMiN uses an Object Relational Model (ORM) library to allow database records to be treated as objects. My research showed that the Sugar ORM⁶ library provided the necessary functionality and was easy to integrate with the application.

8.7 LOGGING

As with all distributed projects, it is useful to maintain a central record of activities and errors encountered. OMiN uses Logentries⁷ to do this. Logentries provides a simple Android library which records and stores logging data and sends it to a central server when an Internet connection is available. OMiN uses Logentries because it is capable of dealing with situations where an Internet connection is unavailable – a common scenario in opportunistic networks. I was careful to ensure that all information sent to Logentries is anonymised in compliance with the ethics policy. Appendix 4 contains an example of the logging data stored by Logentries.

8.8 3RD PARTY LIBRARIES AND CODE

OMiN also uses the following libraries and code snippets:

- Sugar ORM: Object Relational Model for Android databases.
- Logentries: Sending logging data to a central server.
- Appcompat V7⁸: Android compatibility library for Material Design UI patterns.
- Guava⁹: Java utility library.
- Floating Action Button¹⁰: Material Design floating button.
- PrettyTime¹¹: Verbose description of time differences.
- Java Pairing-Based Cryptography Library¹²: Implementation of Paterson & Schuldt' ID-based signing algorithm [30].
- <http://stackoverflow.com/a/20072918>: method for showing toast notifications from non-UI thread.
- BluetoothChatService.java from Android BluetoothChatService example¹³.

⁶ <http://satyan.github.io/sugar>

⁷ <https://logentries.com>

⁸ <https://developer.android.com/tools/support-library/features.html#v7-appcompat>

⁹ <https://github.com/google/guava>

¹⁰ <https://github.com/makovkastar/FloatingActionButton>

¹¹ <http://www.ocpsoft.org/prettytime>

¹² <http://gas.dia.unisa.it/projects/jpb>

- Android library's implementation of Base64¹⁴ .
- ListViewAnimations¹⁵: Swipe-to-dismiss and other ListView animations.
- Example of CGI in Java.¹⁶

¹³ <https://developer.android.com/samples/BluetoothChat/src/com.example.android.bluetoothchat/BluetoothChatService.html>

¹⁴ <https://developer.android.com/reference/android/util/Base64.html>

¹⁵ <https://github.com/nhaarman/ListViewAnimations>

¹⁶ <http://www.javaworld.com/article/2076863/java-web-development/write-cgi-programs-in-java.html>

9 SOFTWARE ENGINEERING PROCESS

9.1 TASK MANAGEMENT

The project was divided up into a number of tasks to be completed, along with their importance, timescale and dependencies (what tasks need to be completed beforehand).

Figure 17 shows a snapshot of the task list during the software development process.

Number	Done	Status	Description	Type	Importance	Timescale
17 Y	Done	implement a simple epidemic routing algorithm to send message to all available nodes	objective	Primary	Days	
21 Y	Done	Design encryption mechanism	task	Primary	Days	
20 Y	Done	Design a routing algorithm using user metadata to route messages while disguising message content	objective	Primary	Days	
44	Ready	Poster	objective	Primary	Days	
24	Ready	Design smartphone UI	task	Secondary	Days	
23	Blocked	Create a smartphone UI	objective	Secondary	Weeks	
28 Y	Done	design trust mechanism	task	Secondary	Days	
26	Ready	Implement a more advanced routing algorithm	objective	Secondary	Weeks	
27	Blocked	Implement a mechanism to decide whether a node is trustworthy or not	objective	Secondary	Weeks	
31	Ready	Find users to participate	task	Secondary	Days	
32 Y	Done	send logs to a central server	task	Secondary	Days	
33	Blocked	obtain performance data	task	Secondary	Weeks	
30	Blocked	Evaluate the performance of the implemented routing algorithms	objective	Secondary	Weeks	
36	Ready	set up simulation	task	Tertiary	Weeks	
37	Blocked	run simulation	task	Tertiary	Days	
35	Blocked	Compare the real world vs simulated performance of the routing algorithms	objective	Tertiary	Weeks	
42	Blocked	publish on app store	task	Tertiary	Days	
43 Y	Done	publish on github	task	Tertiary	Days	
Primary	95%					
Secondary	22%					
Tertiary	20%					
Total	64%					

Figure 17: Snapshot of task list during the development process.

9.2 STORAGE

The project is stored in the school's Mercurial repository¹⁷, with a backup on my personal GitHub account¹⁸ (the git-remote-hg plugin¹⁹ allows pushing to both Git and Mercurial repositories).

Having a backup turned out to be very useful because, both repositories experienced outages during development – GitHub was affected by a massive DDoS attack in March, and I occasionally misconfigured the school's Mercurial repository.

9.3 IMPLEMENTATION

The software was built incrementally, starting with a simple system to detect nearby Bluetooth users and adding features from the task list to move towards the objectives. Once a task is completed and the code is in a working state, it is committed to the repository.

¹⁷ <http://ndw.hg.cs.st-andrews.ac.uk/sh-proj>

¹⁸ <http://github.com/neilw4/OMiN>

¹⁹ <https://github.com/felipec/git-remote-hg>

9.4 REGRESSION TESTING

While it is desirable to have a bug-free program, there is a trade-off between the importance of having few bugs and the time taken to reduce the number of bugs. I considered a number of techniques for reducing the number of bugs.

I believe that the best way to reduce bugs is to catch them as quickly as possible – preferably before they are created. This can be done by

- Writing clear, uncomplicated code: code that is unclear, has too many levels of abstraction or dependencies often causes bugs because it has many edge cases and is hard to modify correctly.
- Writing defensive code: I try to write code capable of recognising, reporting and correctly dealing with unanticipated behaviour, often caused by bugs. This allows the software to continue working when it encounters a problem. It also allowed me to use the central logging server to record details of bugs.
- Rigorously testing every new feature: after modifying code, I manually test it comprehensively with different conditions and inputs to check that it works as intended.

I also occasionally did some more thorough testing of the whole system:

- Static code analysis: I used the PMD²⁰ tool to identify possible bugs in the system.
- User Testing: by asking potential users to try the system I found a number of bugs which I would not have noticed. This also allowed me to collect data to analyse the performance of the network.

There are a number of techniques I considered but did not use:

- Test driven development: I did not believe that it was productive to take the extra time to write unit tests and structure the program to be testable when I can catch most of the important bugs using other techniques.
- Formal verification: while this is the only way to prove that software is bug-free, it takes a large amount of development time.
- Code reviews: these are very good at catching implementation errors and ensuring that the program is well-structured, but require multiple people working on the project.

9.5 PROBLEMS ENCOUNTERED WHILE IMPLEMENTING

9.5.1 *Scala Language*

I initially started using the Scala language to implement OMNiN. However, I experienced a number of problems with the build tools – builds often failed for obscure reasons. It was also very hard to find libraries which were compatible with Scala and Android – for example

²⁰ <http://pmd.sourceforge.net>

I had trouble finding a Scala ORM which could easily plug in to Android database management API. While I would have loved to continue using Scala, I concluded that I would be able to work much faster in Java where, although code is a lot more verbose, I would waste less time debugging obscure build problems and trying different libraries to find one that would work.

9.5.2 *Bluetooth API*

I also had a lot of trouble working with Android's Bluetooth API. There are a number of undocumented bugs in the API – it cannot query a device to find out if it uses OMiN (by requesting UUIDs) until a Bluetooth scan is complete and messages from the OS (intents) may contain an array of arguments or just one, and there is no obvious way to find out. Android is also not kind to the background service running the Bluetooth server – these services are not intended to run permanently and the OS often kills the service because it has a low priority.

10 ETHICS

In order to test the real world performance of the network, a number of people installed OMiN on their smartphones and I collected anonymised data on their interactions. This data includes:

- Device ID but NOT username.
- Anonymised friends list of users.
- Record of message IDs passed during encounters but NOT message contents.
- Date and time of encounters between users.

A sample of this data can be found in Appendix 4. All data which is uploaded to the logging server is displayed as a notification for the user to see.

11 EVALUATION AND CRITICAL APPRAISAL

11.1 IMPACT OF CRYPTOGRAPHY

During the user testing phase, I collected data on how long it took to perform all of the cryptography operations. The complete dataset can be found in Appendix 5.

The first time the PKG runs, it must generate a set of master keys. This takes a median of 2120ms (s.d. 727.7ms), but should only happen once, when the network is set up. After this is done, it takes a median of 2ms (s.d. 0.6ms) to generate a private key for a user. While the actual key generation does not take long, the Android client sees a median delay of 728ms (s.d. 128.2ms) between choosing a username and receiving a private key. This is because of the overhead of networking, CGI, starting the JVM for the PKG and PKG security checks.

Section 11.6.3 explores options to reduce this delay.

When a user sends a message, it takes a median of 785ms (s.d. 409.6ms) to sign it to verify its origin. This should have very little effect on the network – it is a relatively small delay compared to the time it will take to deliver a message.

More worryingly, it takes a median of 2336ms (s.d. 372ms) to verify the authenticity of a message – an operation that happens every time a message is received. While this is unlikely to affect message latency, it is a significant amount of processing which is rarely necessary. Section 11.6.2 explores ways to reduce this overhead.

Overall, the additional time taken to perform cryptography operations is unlikely to have any negative effect on message latency because of the large delay between receiving and forwarding a message. However message verification does introduce a significant amount of processing which will have a negative effect on the battery life of mobile devices. Further work should be able to reduce the number of messages that need to be verified in order to preserve battery life as much as possible.

11.2 OBJECTIVES

Appendix 6 details which objectives the project met and failed to meet. 8 out of 11 objectives were met and 3 lower priority objectives were unmet. In particular, the PRoPHET-based routing algorithm I designed showed promise, but lack of time prevented me from implementing it and assessing its real-world performance.

11.3 REQUIREMENTS

Appendix 7 contains a detailed breakdown of exactly which requirements were met. 24 out of 27 requirements were met and 3 were unmet. All high priority requirements were met,

while 1 medium (implementation of the PRoPHET-based routing algorithm) and 2 low priority requirements were unmet.

11.4 USE CASES

OMiN is a general system which should work well in a number of use cases. In addition to the original use cases, I have also analysed its suitability for a number of other common use cases for opportunistic networks, showing its flexibility.

11.4.1 *Disaster Area*

OMiN supports all of the core needs of the disaster use case described in section 3.1 (page 19): provided that OMiN has been pre-installed on Alex's Android smartphone, she can send a message which will propagate throughout the network, reaching Doctor Brian by the shortest route. Doctor Brian's reply will take a similar path.

11.4.2 *Metro*

OMiN needs a number of improvements before it can work successfully in the subway scenario (section 3.2 – page 19). In particular, it needs a more advanced routing protocol which will prioritise popular messages, such as the PRoPHET-based algorithm I designed. OMiN also currently lacks the ability to send and receive messages over the Internet (between news organisations and base stations in subway stations), which would have increased the utility of the network for this use case. Apart from this, OMiN will allow underground commuters to communicate in a limited way, and high capacity nodes could be installed in subway carriages (similar to Daknet's mobile access points) to improve the network.

11.4.3 *Extra Use Case – Tracking Animal Movements*

Animals could be tracked using electronic tags which form an opportunistic network, for example in SWIM [26]. This has the advantage that trackers do not have to find each animal to obtain data from its tag. In this case, an epidemic routing algorithm is probably the best choice because data should saturate the network in order to reach the trackers. Provided that it is possible for the electronic animal tags to use OMiN, it will be able to support the core needs of this scenario.

11.4.4 *Extra Use Case – Connectivity in Developing Countries*

Many developing areas are unable to access the internet, a great tool for learning. Networks such as DAKnet [25] in India provide a means for remote villages to communicate with each other and to request data from the Internet. OMiN is flexible enough to allow for the exchange of messages and for some messages to be requests for data, although data requests are currently not automated. Provided that villages and

mobile access points have Android smartphones, OMiN should be a suitable replacement for networks like DAKnet.

11.4.5 Extra Use Case – Secure Communications

Firechat is widely used to facilitate secure communications between nearby people, often when their Internet access is restricted or monitored. Its security is questionable [11], so OMiN could be used as a much more secure alternative. OMiN naturally mimics Firechat's locality (communication between nearby users) because messages are more likely to reach nearby users. OMiN is ideal for the scenario where secure, unmonitored communication is desirable – it is provably secure and makes every effort to avoid accessing the Internet.

11.5 TESTING

I tested the software manually myself after adding a feature. This often caught bugs which highlighted errors in my implementation of the feature, which I could then fix. Near the end of the project, I also asked a number of colleagues to comprehensively test the app. This uncovered a number of small defects – typos, crashes caused by edge cases – which I was able to fix. The only bug I found which is unfixed is that the software occasionally fails to run the Bluetooth server for an unknown reason (probably related to the Android runtime). The software was tested extensively and this, combined with the lack of bugs and defensive coding style allow me to conclude that OMiN is a reliable and robust piece of software.

11.6 FURTHER WORK

While OMiN works very well, there are a number of enhancements that could be made to make it more suitable for the use cases.

11.6.1 PRoPHET-Based Routing Algorithm

The PRoPHET-based routing algorithm was designed, although not implemented as the primary focus was on the higher priority requirements such as security. OMiN currently uses an epidemic routing algorithm which is acceptable in most cases, but a more advanced algorithm will greatly improve the overall efficiency and decrease resource usage.

11.6.2 Reducing Verification Overhead

As seen in section 11.1, the policy of verifying every message when it is received will potentially use up the battery due to the significant amount of processing involved. There are a number of policies which could reduce the effects of this.

OMiN already avoids making connections when the battery is low, so it will not receive any messages to verify when the battery is low.

Users will not see every message stored on their device – they will only see messages from users they are following. These messages must be verified, but other messages that are only being stored by the device are less important. If these less important messages are not verified, the worst case is that a false message will be passed on to the destination, which will identify that it is false and discard it. This can be counterbalanced if every node verifies a small subset of unimportant messages. Perhaps this should only happen when the device is being charged when power consumption is not an issue.

11.6.3 Faster Server

The Android client sees a delay of about 0.68s between choosing a username and receiving a private key. Since the actual key generation takes very little time (2ms), the rest is overhead which could be reduced.

This is mostly due to the choice of CGI and Java, which are both inefficient at dealing with requests – CGI because it uses a separate process for every request and Java because the JVM has to boot up for every request. CGI has been made redundant by the more efficient FastCGI protocol, but it currently has no integration with Java. If I had a chance to rewrite the server, I would plug the PKG server into a lightweight Java HTTP server such as Jetty.²¹ This would allow requests to run in separate threads. These could use Java's in-memory locking structures instead of Unix file locking.

11.6.4 Offline Installation

Currently, installing the app requires downloading it from the app store, which requires an Internet connection. This defeats the objective of the network – to communicate without using the Internet. There is an alternative method of installation though – the Android installation file (APK) can be sent to another device via Bluetooth, where it can be installed without access to the outside world. This is possible in Android because apps can locate their own APK files using the ApplicationInfoAPI.²²

11.6.5 Encrypted Private Messages

Many social networks include facilities for sending hidden private messages which can only be seen by the sender and intended recipient. This could be implemented by OMiN using a HIBC capable of encrypting and decrypting messages. Many cryptography algorithms support 'signcryption' – both signing and encryption [27]. If this cannot be done, it is possible to create a hybrid scheme out of a HIBE signing and an encrypting algorithm which uses the same public key (a user's ID) and stores two private keys – one for the signing algorithm and one for the encryption algorithm.

²¹ <http://eclipse.org/jetty>

²² <https://developer.android.com/reference/android/content/pm/ApplicationInfo.html#publicSourceDir>

11.6.6 Multimedia Messages

There is no reason why OMNiN should be restricted to short text messages. Its capabilities could be expanded to allow pictures and formatted text – this would be very useful for distributing news in the Metro use case. The routing algorithm and buffer eviction strategy may have to be modified to consider the size of a message. From a security perspective, we would have to make sure that multimedia messages cannot be used by an attacker to gain control over a device.

11.7 OVERALL EVALUATION

This software project has successfully met all of its high priority requirements – the only medium priority requirement which has not been implemented is the PRoPHET-based routing algorithm although I am confident that, given time, this could be done.

The use of cryptography to satisfy the security requirement is a good solution which does not affect the performance of the network, it has one caveat: cryptographic approaches do introduce an extra amount of computation. For this reason it would be advisable to seek alternatives to cryptographic techniques in situations where power consumption is an issue.

The project has proven that it is possible to implement a secure, reliable and robust microblogging platform using an opportunistic network which, with a few adjustments, can not only be used in the given use cases but is also useable in many other use cases.

12 CONCLUSION

This report describes a complete method of communicating securely in an Internet-free environment. With the addition of the PRoPHET-based routing algorithm, OMiN, my implementation of an opportunistic network, is fully capable of allowing people to communicate securely in the given use cases and many more, while protecting privacy as much as possible. Complete security within an opportunistic network is an unsolved problem, but this report describes a cryptographic scheme which provides near-complete security by minimising the number of trusted parties.

13 ACKNOWLEDGEMENTS

I would like to thank Dr Tristan Henderson for his time, support and guidance over the year. I would also like to thank Clare and Dave Wells for proof-reading this report, and James Spyte and Tom Dalton for user-testing the program.

14 BIBLIOGRAPHY

1. Kaplan AM, Haenlein M. The early bird catches the news: Nine things you should know about micro-blogging. *Business Horizons*. 2011;54(2):105-113. Available from: <http://dx.doi.org/10.1016/j.bushor.2010.09.004>.
2. Boyd D, Ellison NB. Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*. 2007;13(1):210-230. Available from: <http://dx.doi.org/10.1111/j.1083-6101.2007.00393.x>.
3. Kaur N. AN INTRODUCTION TO WIRELESS MOBILE SOCIAL NETWORKING IN OPPORTUNISTIC COMMUNICATION. *International Journal of Distributed & Parallel Systems*. 2013;4(3). Available from: <http://airccse.org/journal/ijdps/papers/4313ijdps06.pdf>.
4. Chaintreau A, Hui P, Crowcroft J, Diot C, Gass R, Scott J. Impact of human mobility on opportunistic forwarding algorithms. *Mobile Computing, IEEE Transactions on*. 2007;6(6):606-620. Available from: <http://dx.doi.org/10.1109/TMC.2007.1060>.
5. Scott J, Crowcroft J, Hui P, Diot C, Others. Haggle: A networking architecture designed around mobile users. In: *WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services*; 2006. p. 78-86. Available from: <http://hal.inria.fr/inria-00001012>.
6. Musolesi M, Hailes S, Mascolo C. ; Adaptive routing for intermittently connected mobile ad hoc networks. In: *World of Wireless Mobile and Multimedia Networks*, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a; 2005. p. 183-189. Available from: <http://dx.doi.org/10.1109/WOWMOM.2005.17>.
7. Leguay J, Friedman T, Conan V. ; Evaluating Mobility Pattern Space Routing for DTNs. In: *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*; 2006. p. 1-10. Available from: <http://dx.doi.org/10.1109/INFCOM.2006.299>.
8. Bigwood G, Henderson T. ; IRONMAN: Using Social Networks to Add Incentives and Reputation to Opportunistic Networks [cited In: *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*; 2011. p. 65-72. Available from: <http://dx.doi.org/10.1109/PASSAT/SocialCom.2011.60>.
9. Vahdat A, Becker D. ; Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Duke University; 2000. Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.6151>.
10. Lindgren A, Doria A, Schelén O. ; Probabilistic Routing in Intermittently Connected Networks. *SIGMOBILE Mob Comput Commun Rev*. 2003 Jul;7(3):19-20. Available from: <http://doi.acm.org/10.1145/961268.961272>.
11. Firechat and Nearby Communication. BreizhEntropy. 2014. Available from: http://breizh-entropy.org/~nameless/random/posts/firechat_and_nearby_communication.
12. Hui P, Crowcroft J, Yoneki E. ; 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 '08*. New York, NY, USA: ACM; 2008. p. 241-250. Available from: <http://doi.acm.org/10.1145/1374618.1374652>.
13. Trifunovic S, Legendre F, Anastasiades C. ; Social Trust in Opportunistic Networks. In: *INFOCOM IEEE Conference on Computer Communications Workshops*, 2010; 2010. p. 1-6. Available from:

- <http://dx.doi.org/10.1109/INFCOMW.2010.5466696>.
14. Shikfa A, Onen M, Molva R. ; Bootstrapping security associations in opportunistic networks. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on; 2010. p. 147-152. Available from: <http://dx.doi.org/10.1109/PERCOMW.2010.5470676>.
 15. Capkun S, Butyan L, Hubaux JP. ; Self-organized public-key management for mobile ad hoc networks. *Mobile Computing, IEEE Transactions on*. 2003 Jan;2(1):52-64. Available from: <http://dx.doi.org/10.1109/TMC.2003.1195151>.
 16. Shamir, A. Identity-Based Cryptosystems and Signature Schemes. In: *Lecture Notes in Computer Science*. Springer; 1985. p. 47-53. Available from: http://dx.doi.org/10.1007/3-540-39568-7_5.
 17. Kamat P, Baliga A, Trappe W. ; An Identity-based Security Framework For VANETs. In: *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks. VANET '06*. New York, NY, USA: ACM; 2006. p. 94-95. Available from: <http://doi.acm.org/10.1145/1161064.1161083>.
 18. Kong J, Petros Z, Luo H, Lu S, Zhang L. ; Providing robust and ubiquitous security support for mobile ad-hoc networks. In: *Network Protocols, 2001. Ninth International Conference on*; 2001. p. 251-260. Available from: <http://dx.doi.org/10.1109/ICNP.2001.992905>.
 19. Horwitz J, Lynn, B. Toward Hierarchical Identity-Based Encryption. In: *Advances in Cryptology — EUROCRYPT 2002*. Springer; 2002. p. 466-481. Available from: http://dx.doi.org/10.1007/3-540-46035-7_31.
 20. Seth A, Keshav S. ; Practical security for disconnected nodes. In: *Secure Network Protocols, 2005. (NPSSec). 1st IEEE ICNP Workshop on*; 2005. p. 31-36. Available from: <http://dx.doi.org/10.1109/NPSEC.2005.1532050>.
 21. Parris I, Henderson T. ; Privacy-enhanced social-network routing. *Computer Communications*. 2012;35(1):62-74. Available from: <http://www.sciencedirect.com/science/article/pii/S0140366410004767>.
 22. BH, Bloom. ; Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun ACM*. 1970 Jul;13(7):422-426. Available from: <http://doi.acm.org/10.1145/362686.362692>.
 23. Hern A. Firechat updates as 40,000 Iraqis download 'mesh' chat app in censored Baghdad. *The Guardian*. 2014. Available from: <http://www.theguardian.com/technology/2014/jun/24/firechat-updates-as-40000-iraqis-download-m>.
 24. Bland A. FireChat – the messaging app that's powering the Hong Kong protests. *The Guardian*. 2014. Available from: <http://www.theguardian.com/world/2014/sep/29/firechat-messaging-app-powering-hong-kong-protests>.
 25. Pentland A, Fletcher R, Hasson A. DakNet: rethinking connectivity in developing nations. *Computer*. 2004 Jan;37(1):78-83. Available from: <http://dx.doi.org/10.1109/MC.2004.1260729>.
 26. Small T, Haas ZJ. ; The Shared Wireless Infostation Model: A New Ad Hoc Networking Paradigm (or Where There is a Whale, There is a Way). In: *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking*. [cited Computing. MobiHoc '03. New York, NY, USA: ACM; 2003. p. 233-244. Available from: <http://doi.acm.org/10.1145/778415.778443>
 27. Yuen TH, Wei VK. Constant-Size Hierarchical Identity-Based Signature/Signcryption without Random Oracles. *IACR Cryptology ePrint Archive*. 2005;2005:412. Available from: <http://eprint.iacr.org/2005/412.pdf>.
 28. Li N, Das SK. ; RADON: Reputation-assisted Data Forwarding in Opportunistic Networks. In: *Proceedings of the Second International Workshop on Mobile Opportunistic Networking*.

- MobiOpp '10. New York, NY, USA: ACM; 2010. p. 8-14. Available from:
<http://doi.acm.org/10.1145/1755743.1755746>.
29. Android Developer Dashboard. 2015. Available from:
<https://developer.android.com/about/dashboards/index.html>.
30. Paterson KG, Schuldt JCN. ; Efficient identity-based signatures secure in the standard model. In: Information Security and Privacy. Springer; 2006. p. 207-222. Available from:
http://dx.doi.org/10.1007/11780656_18.

15 APPENDICES

APPENDIX 1 DICTIONARY OF TERMS

- CAR – Context Aware Routing – a routing algorithm where a single copy of the message is passed between nodes and never duplicated.
- CGI – Common Gateway Interface – a standardised interface between web servers and programs to generate dynamic content.
- HIBC – Hierarchical Identity-Based Cryptography – a form of IBC where public keys consist of a list of short identities and any party with a private key can act as a PKG and ‘delegate’ a private key to another party.
- HIBE – Hierarchical Identity-Based Encryption – a form of HIBC where the public key can be used to encrypt a message so that it can only be decrypted using the corresponding private key.
- HIBS – Hierarchical Identity-Based Signing – a form of HIBC where the private key can be used to sign a message and the public key (the identity) is used to verify the signature.
- IBC – Identity-Based Cryptography – a form of asymmetric key cryptography where the public key is a short identity and the private key is generated by a central PKG.
- IBE – Identity-Based Encryption – a form of IBC where the public key can be used to encrypt a message so that it can only be decrypted using the corresponding private key.
- IBS – Identity-Based Signing – a form of IBC where the private key can be used to sign a message and the public key (the identity) is used to verify the signature.
- JSON – JavaScript Object Notation – a format for encoding data in human-readable strings.
- MAP – Mobile Access Point – an opportunistic network node which moves such as a bus.
- OMiN – Opportunistic Microblogging Network – my implementation of a microblogging platform and opportunistic network for Android smartphones and tablets.
- ORM – Object Relational Model – a mapping from an object-oriented data structure to a relational database.
- OSNR – Obfuscated Social Network Routing – a method of obscuring metadata for use in routing algorithms by encoding it in a Bloom filter.
- PKG – Private Key Generator – a central server used by IBC schemes to generate and distribute private keys.
- PRoPHET – Probabilistic Routing in Intermittently Connected Networks – a routing algorithm where many copies of a message are distributed throughout the network according to a heuristic.
- PSN – Pocket Switched Network – a form of opportunistic network which is optimised to run on devices carried by people.

- SSL – Secure Sockets Layer – a standard for securely communicating with a server over the Internet.
- SSNR – Statisticulated Social Network Routing – a method of obscuring metadata for use in routing algorithms by adding false data.

APPENDIX 2 USER MANUAL

Installation

The app can be installed from the Google Play store.²³

Managing Usernames

If you have no username you can still send messages anonymously. To set or change your username, press the  icon in the action bar at the top of the screen. You can now fill in a username. When you are done, press the  icon to set your username. If the username has already been taken, you will be notified and the username will be removed.

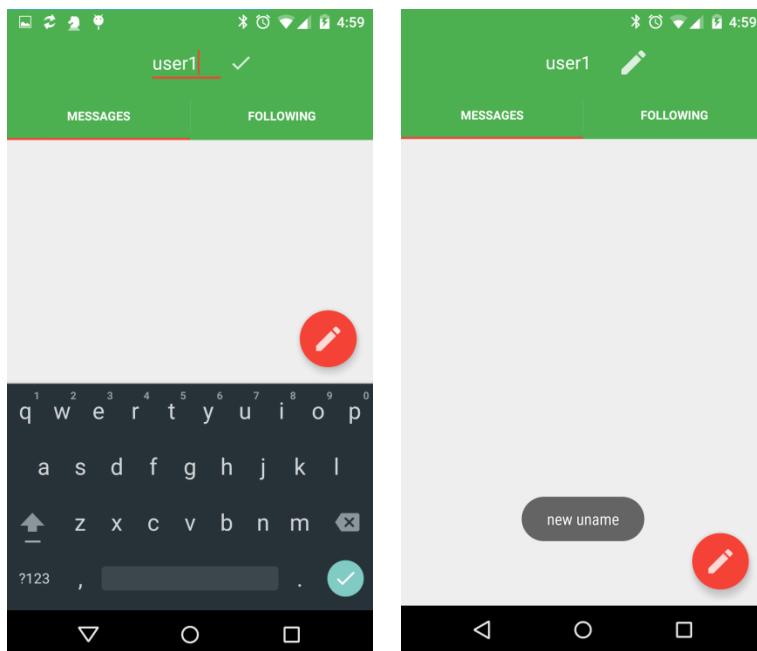


Figure 18: Editing a username.

Sending Messages

To send a message, click the red  button at the bottom right of the messages tab. Type in your message and press “SEND” to send it or “CANCEL” to cancel the message. If you have no username, the message will be sent anonymously.

²³ <https://play.google.com/store/apps/details?id=neilw4.0min>

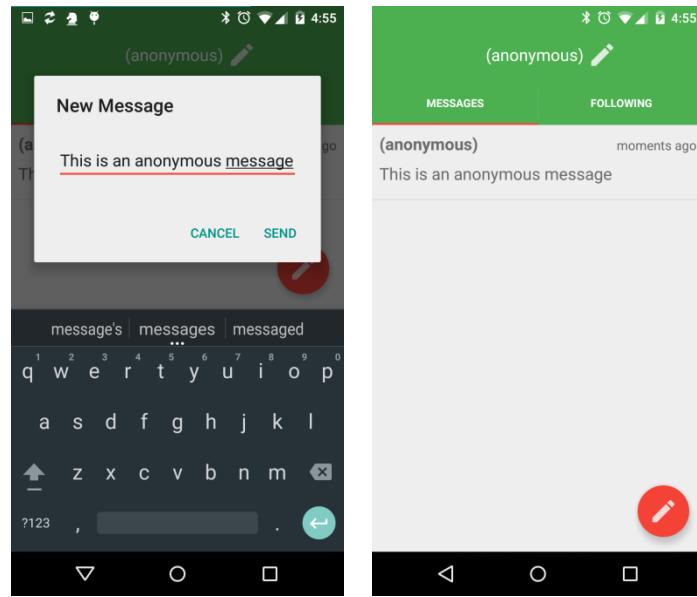


Figure 19: Sending an anonymous message.

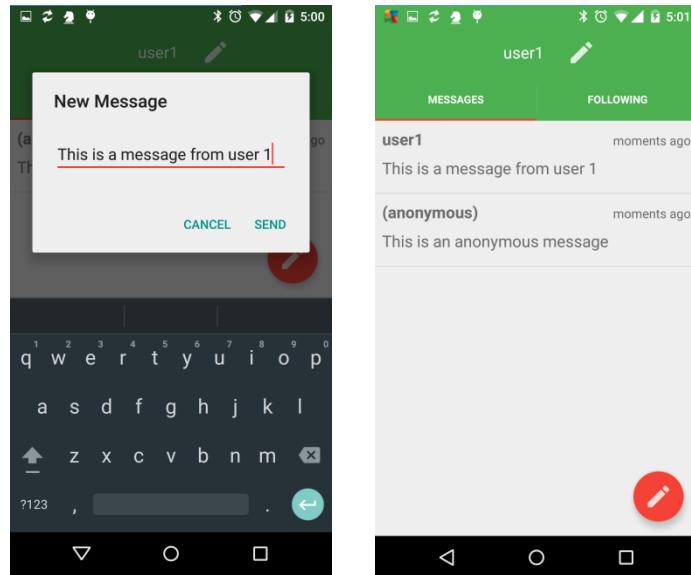


Figure 20: Sending a non-anonymous message.

Following Users

The list of messages will only show messages you sent, anonymous messages and messages from people you follow. To follow someone, click on the “FOLLOWING” tab. Then click the red + button at the bottom right of the screen. Choose the name of the person you want to follow (this is arbitrary) and fill in their username. Click “FOLLOW” to follow that person or “CANCEL” to cancel. To stop following someone, find their name in the list of people you are following and swipe it left or right.

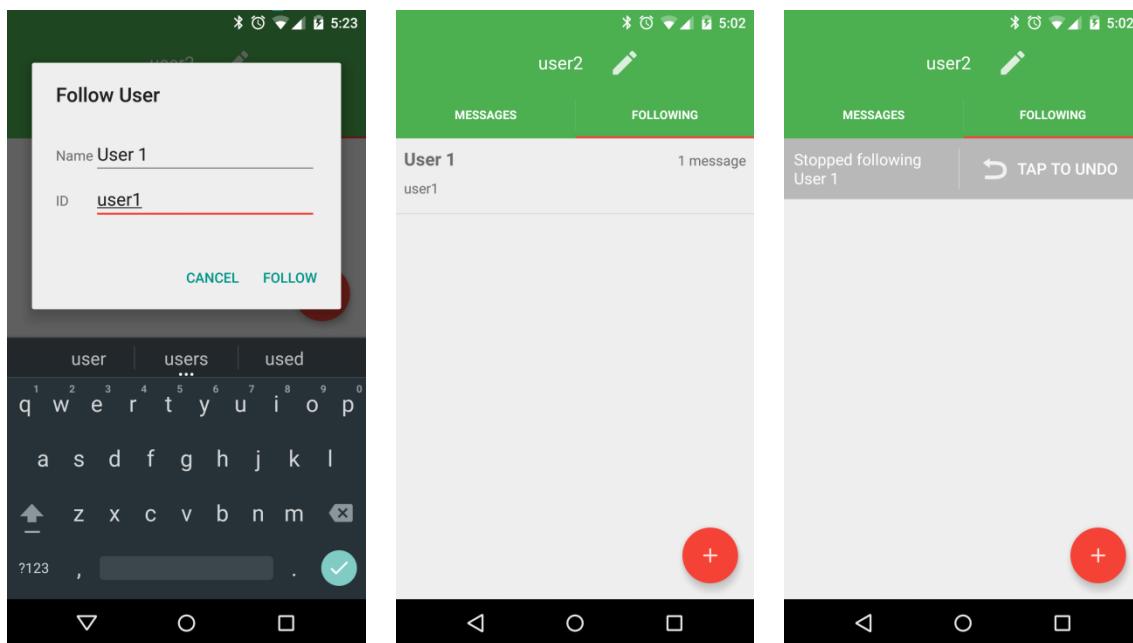


Figure 21: Following and unfollowing a user.

APPENDIX 3 MAINTENANCE DOCUMENT

Source Code

Source code is stored using the school's Mercurial service at <http://ndw.hg.cs.st-andrews.ac.uk/sh-proj> and on GitHub at <https://github.com/neilw4/OMiN>. To download the source code, use the command

```
hg clone http://ndw.hg.cs.st-andrews.ac.uk/sh-proj  
or
```

```
git clone http://github.com/neilw4/OMiN.git
```

Project Layout

The project is split into three different modules, each built using Gradle:

- The *app* module contains the Android app to be installed on every node.
- The *pkg* module is the central authentication server, which runs on the school's host server via CGI.
- The *crypto* module is a library of cryptography functions used by both the app and authentication server.

Building

Executing the following command from the project directory will build everything, downloading libraries, build scripts and the Android SDK if necessary:

```
./gradlew build  
The binaries will now be in the following locations:
```

- The main app will be located at app/build/outputs/apk/app-debug.apk
- The cryptography library will be at crypto/build/libs/crypto.jar
- The authentication server will be at pkg/build/libs/pkg.jar and can be executed using the CGI script at pkg/omin.cgi

Installation - Authentication Server

To run the authentication server, configure a web server to run the *omin.cgi* script in the *pkg* directory. The Android app will have to be modified to use the new server location and master public key. The server stores private information such as the master keys in the working directory, so it is essential that the web server cannot serve these files (e.g. by creating a separate CGI script in the public directory of the web server to call the authentication script in a non-public directory).

Installation - Android App

The app can be installed from the Google Play store²⁴ or by executing the following command from the project directory to build and install the app:

```
./gradlew installDebug
```

APPENDIX 4 EXAMPLE LOGGING DATA

```
28 Mar 2015 16:08:29.550 deviceID=02ec7e7ce078202c Sat 28 Mar 16:08:28 +0000 2015, severity=INFO, SendMessageManager: new message: 2015-03-28 16:08:28.773
28 Mar 2015 16:08:30.082 deviceID=02ec7e7ce078202c Sat 28 Mar 16:08:29 +0000 2015, severity=INFO, Signer: signed message 2015-03-28 16:08:28.773 in 514ms - 384 bytes
28 Mar 2015 16:08:32.702 deviceID=02ec7e7ce078202c Sat 28 Mar 16:08:31 +0000 2015, severity=INFO, Signer: verified message 2015-03-28 16:08:28.773 in 2594ms
28 Mar 2015 16:10:03.221 deviceID=02ec7e7ce078202c Sat 28 Mar 16:10:02 +0000 2015, severity=INFO, ConnectionService: started discovery
28 Mar 2015 16:40:17.866 deviceID=02ec7e7ce078202c Sat 28 Mar 16:40:17 +0000 2015, severity=INFO, ConnectionService: started discovery
28 Mar 2015 16:41:41.224 deviceID=02ec7e7ce078202c Sat 28 Mar 16:41:40 +0000 2015, severity=INFO, UnameManager: new uname
28 Mar 2015 16:41:41.230 deviceID=02ec7e7ce078202c Sat 28 Mar 16:41:40 +0000 2015, severity=INFO, AsyncFetchTask: Fetching secret keys from PKG
28 Mar 2015 16:41:42.260 deviceID=02ec7e7ce078202c Sat 28 Mar 16:41:41 +0000 2015, severity=INFO, AsyncFetchTask: successfully got secret key in 1025ms
```

APPENDIX 5 CRYPTOGRAPHY TIMING DATA

Time to Generate Master Keys

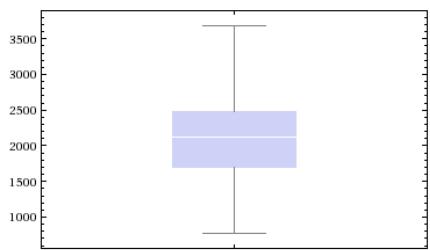
2120ms, 1954ms, 2023ms, 2791ms, 2290ms, 1692ms, 2483ms, 3682ms, 1692ms, 771ms, 2334ms.

Mean: 2166.5ms

Median: 2120ms

Standard Deviation: 727.7ms

²⁴ <https://play.google.com/store/apps/details?id=neilw4.omin>



Time to Generate Private Keys by the PKG

2ms, 2ms, 1ms, 2ms, 2ms, 3ms, 2ms, 2ms, 1ms, 2ms, 1ms, 2ms, 3ms, 2ms.

Mean: 1.9ms

Median: 2ms

Standard Deviation: 0.6ms

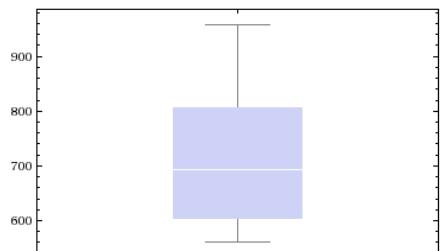
Total Time to Request a Private Key from the PKG

957ms, 884ms, 595ms, 606ms, 694ms, 562ms, 603ms, 785ms, 808ms, 762ms.

Mean: 725ms.

Median: 728ms.

Standard Deviation: 128.2ms.



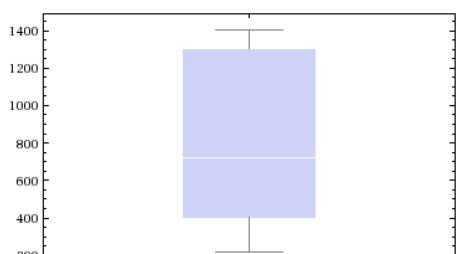
Time to Sign a Message

1303ms, 403ms, 218ms, 719ms, 1403ms, 573ms, 875ms.

Mean: 785ms.

Median: 719ms.

Standard Deviation: 409.7ms.



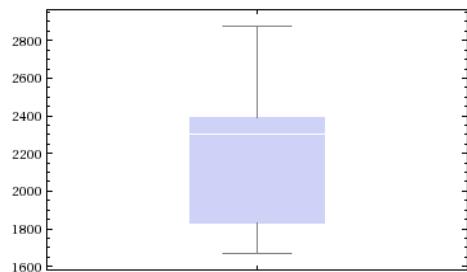
Time to Verify a Message

2876ms, 2570ms, 1829ms, 2301ms, 2392ms, 2004ms, 1668ms, 2373ms.

Mean: 2251.6ms.

Median 2337ms.

Standard Deviation 372ms.



APPENDIX 6 OBJECTIVES MET

Importance	Description	Met
Primary	Design and implement protocol for discovering nodes in close proximity and passing messages and necessary metadata between them.	Met
Primary	Create core library to manage message storage and routing.	Met
Primary	Implement epidemic routing algorithm to send messages to all available nodes.	Met
Primary	Design routing algorithm using user metadata to route messages while disguising message content and metadata.	Met
Primary	Design scheme to allow verification of the origin and integrity of a message.	Met
Secondary	Create user interface.	Met
Secondary	Implement more advanced routing algorithm.	Unmet
Secondary	Implement message verification scheme.	Met
Secondary	Evaluate impact of message verification scheme.	Met
Secondary	Evaluate performance of the implemented routing algorithms.	Unmet
Tertiary	Compare real world vs. simulated performance of the routing algorithms.	Unmet

APPENDIX 7 REQUIREMENTS MET

Met Requirements

Type	Priority	Description
Functional User	High	The user shall be able to create a unique identity.

Functional User	High	The user shall be able to send plain text messages to all others who follow the user.
Functional User	High	The user shall be able to 'follow' any user and receive messages sent by that user.
Functional User	High	The user shall be able to send messages without requiring an Internet connection.
Non-Functional User	High	The network shall be usable on a large number of mobile devices.
Non-Functional User	Medium	The user shall be able to be confident in the origin and integrity of a message.
Non-Functional User	Low	The user shall be able to use the network with minimal training.
Functional System	High	The system shall work on portable electronic devices such as smartphones or tablets.
Functional System	High	The system shall allow creation of user identities with a unique cryptographic identity.
Functional System	High	The system shall automatically connect to nearby nodes and pass on relevant information.
Functional System	High	The system shall pass on messages until they reach their destination.
Functional System	Medium	The system shall ensure that messages cannot be modified in transit or that any such modifications can be detected.
Functional System	Medium	The system shall ensure that nodes cannot send a message that appears to be from another user.
Functional System	Medium	The system shall restrict the size of the message store.
Functional System	Medium	The system shall protect against Sybil attacks.
Functional System	Medium	The system shall prevent messages from being modified while in transit.
Functional System	Medium	The system shall protect user metadata from all other nodes.
Functional	Medium	The system shall be scalable to an arbitrary number of nodes.

System		
Functional System	Low	The system shall block attempts to prevent message propagation.
Non-Functional System	High	The system shall work in an unstructured environment with random encounters between nodes.
Non-Functional System	High	The system shall not require a connection to any other network (such as the Internet).
Non-Functional System	Medium	The system shall be robust and able to continue functioning when it encounters an unexpected state such as a malfunctioning or untrustworthy node.
Non-Functional System	Medium	The system shall minimise the number of messages lost before they reach their destination.
Non-Functional System	Medium	The system shall deliver messages as quickly as possible.

Unmet Requirements

Type	Priority	Description	Comments
Functional User	Low	The user shall be able to send messages via the Internet to Internet-connected nodes.	Not implemented – would require a lot of work.
Functional System	Low	The system shall have mechanisms to mitigate Denial of Service attacks.	Not implemented.
Non-Functional System	Medium	The system shall route messages effectively given a semi-predictable set of encounters between nodes.	Non-epidemic routing protocol designed but not implemented.