Polar: Proxies Collaborating to Achieve Anonymous Web Browsing

by

Heiko Mark Tillwick

Submitted in partial fulfilment of the requirements for the degree

Magister Scientia (Computer Science)

in the

Faculty of Engineering, Built Environment and Information Technology

at the

University of Pretoria

October 2006

Polar: Proxies Collaborating to Achieve Anonymous Web Browsing

by

Heiko Mark Tillwick

Abstract

User tracking and profiling is a growing threat to online privacy. Whilst Internet users can choose to withhold their personal information, their Internet usage can still be traced back to a unique IP address.

This study considers anonymity as a strong and useful form of privacy protection. More specifically, we examine how current anonymity solutions suffer from a number of deficiencies: they are not commonly used, are vulnerable to a host of attacks or are impractical or too cumbersome for daily use. Most anonymity solutions are centralised or partially centralised and require trust in the operators. It is additionally noted how current solutions fail to promote anonymity for common Web activities such as performing online search queries and general day-to-day Web browsing.

A primary objective of this research is to develop an anonymising Web browsing protocol which aims to be (1) fully distributed, (2) offer adequate levels of anonymity and (3) enable users to browse the Internet anonymously without overly complex mixing techniques.

Our research has led to an anonymising protocol called Polar. Polar is a peer-to-peer network which relays Web requests amongst peers before forwarding it to a Web server, thus protecting the requester's identity.

This dissertation presents the Polar model. Design choices and enhancements to the model are discussed. The author's implementation of Polar is also presented demonstrating that an implementation of Polar is feasible.

Keywords: Anonymous Web browsing, proxies, privacy-enhancing technology
Supervisor: Professor Martin S. Olivier
Department: Department of Computer Science
Degree: Magister Scientia

Acknowledgements

I am deeply indebted to my supervisor Professor Martin Olivier for allowing me to join his research group. His insight has helped me at key points during my research. His style of supervision has positively influenced me for life.

This research would not have been possible without the continued moral encouragement and financial support of my parents. I thank them for the foundation they have provided and in particular for the value they place on education.

I am grateful for the discussions provided by members and supervisors of the ICSA Research Group. Special thanks go to Marco Slaviero for the many stimulating conversations as well for the helpful technical insight into LaTeX. I am thankful to my friend and colleague Thorsten Neumann for co-authoring the initial Polar paper where some ideas, developed in this work, were first conceived.

I am also appreciative of the financial assistance provided by the National Research Foundation as well as by Telkom, IST and Unisys through THRIP. Their funding made it possible for me to attend and present at conferences in South Africa and abroad.

On a more personal note, I continue to be grateful for the friendship and encouragement of my two brothers and my friends. In particular, I thank Eirini Kalimeri for her trust and patience during the long periods we spent apart.

Table of Contents

Chapters

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 The case for privacy

Privacy has been defined as *"the ability of individuals to determine for themselves when, how and to what extent information about them is communicated to others"* [60]. Similarly, Lategan et. al. [65] define privacy as *"a state that exists when access to private information about a particular individual can be effectively controlled and managed by that individual even after a third party has collected such private information"*.

The need for privacy is commonly recognised and accepted in social sciences [69, 57]. On a sociopolitical level, for example, privacy allows for political expression and criticism; on the psychological level it affects self-definition, allows for self-assessment and protects personal autonomy [70]. Margulis, a social and environmental psychologist, sums up the need for privacy in the following statement: *"privacy is important because it is posited to provide experiences that support normal psychological functioning, stable interpersonal relationships, and personal development"* [70, p.246].

It should therefore not be surprising to note that, in an increasingly technology-dependent society where electronic information is readily available and easily distributable, the protection of *electronic* privacy has received increased attention [75].

The increased difficulty of protecting electronic privacy is noted in a num-

ber of publications. Berman and Mulligan [12] identify three characteristics of the electronic medium that pose increased challenges to privacy protection: increased data creation and collection, the globalisation of information and communication and the lack of centralised control mechanisms. Whilst centralisation is not necessarily a better solution for privacy protection, lack thereof often translates to enforcement complications (coordinating and securing distributed decision-making is generally more complex). Another reason given by Chung and Paynter [23] is the efficient and inexpensive collection of electronic data as well as the potential financial benefits.

Before analysing online privacy issues, the following should be noted: the concept of privacy is often more complex than realised as it involves a balance between society's needs and those of an individual [77]. For example, when ordering products online, the individual is required to submit a certain amount of personal information such as name, address and credit card details. Whilst most individuals would consider such an exchange of personal information for services rendered as acceptable, it should be noted that redistribution, selling or even collection of personal information should only occur with the individual's knowledge and consent. Unfortunately this is not always the case.

One should therefore consider the following with particular reference to the Internet: to what extent is privacy already threatened on the Internet, what are possible future threats and how can individuals protect themselves.

1.1.1 Invasion of privacy

A well-known case of unethical consumer data-use involved Lotus Development Corporation and Equifax (one of the "big three" credit reporting agencies in the United States). The planned sale of consumer purchasing records was thwarted after a public outcry and 30 000 letters of complaint [77].

In 2002 the Internet ad-serving company DoubleClick came under the spotlight when it acquired the little-known company Abacus Direct [23, 98]. DoubleClick already owned records of consumers' browsing habits. This data

was collected from cookies distributed through their elaborate banner advertising network. Abacus had built up a database of consumer purchasing habits collected from catalogue retailers and marketers. When DoubleClick announced plans to link its data with the 2.9 billion personally identifiable transaction records owned by Abacus, DoubleClick immediately faced a number of class action lawsuits as the data merger was in violation of their existing privacy policy.

Great economic value is placed on collecting browsing habits and user preferences [40]. It is therefore not surprising to note the number of companies and individuals who actively collect, store and analyse personal information on the Internet.

Common methods of tracking users includes the use of cookies [40, 71, 64], Web bugs [71], the *identd* identification protocol [40] as well as the user's IP address. These methods allow for the collection of clickstream data [71] and subsequent log analysis [40, 71].

For example, search engines such as Google use Web bugs to track which links in their search results are clicked by the user [17, 2]. A Web bug is *"any HTML element that is intended to go unnoticed by users, and is present partially for surveillance purposes"* [71, p.260].

Google's Web bugs are not only used to calculate page rankings but are also used in their context-sensitive banner advertisements called AdSense. User clicks are reported back to a Web server which calculates usage statistics as well as a relevant cost-per-click. Web sites hosting the banner earn revenue when users click on the ad-serving company's banner.

User data includes particulars such as the IP address, which Web browser was used, the previous (or referring) Web page, the cookie, the URL and any user-entered form data submitted via HTTP GET or POST requests [40]. This data can be stored in Web server logs [40] and is often used to obtain user-specific clickstream data.

A clickstream is a record of a user's activity on the Internet or on a particular site. Information extracted includes Web sites visited, Web pages visited, how long the user was on a page or site, in what order the pages were visited as well as the user's IP address [40]. This information is often used

to monitor visitor patterns, site problems and even break-in attempts [71].

Collecting user preferences has become popular with Web site owners as it enables Web site personalisation and allows for targeted marketing techniques [108]. Google News and Google Groups are excellent examples of a Web site's use of clickstream data [108]. This data is used in suggesting more relevant news headlines or news groups (in other words, Web content is personalised), influencing the weighting of search results as well as displaying targeted advertisements.

Although Google's privacy policy states that it does not share personally identifiable information, Google is silent about what clickstream data it collects and what it does with the information. Such surveillance techniques make user profiling possible [40, 2], thus revealing a substantial amount of an individual's sensitive and potentially incriminating browsing history and habits.

The collection of clickstream data is not only possible across a single Web site, but banner advertisement services such as those offered by DoubleClick and Google's AdSense have the potential to collect clickstream data across several sites. These banners have become popular as the cost-per-click feature offers Web sites a secondary source of revenue [66]. These banners threaten online privacy because clickstream data can now be collected across all Web sites hosting an ad-serving company's banner.

Another particularly worrying fact about clickstream data is the ability of proxies to collect browsing information. Web proxies generally present a single gateway to the Internet and are commonly used by organisations. This means that an individual risks exposing their private browsing history not only to Web sites but also to organisational proxies as well as ISPs. A 2005 survey by the American Management association reports that 76% of companies monitor employee Web usage [41].

The threat of online profiling thus exists at different levels: the danger of being profiled by *Web sites*, *ad-serving companies*, *organisational proxies* or by *ISPs*. Whereas Web sites can only trace a user's actions on the site itself, organisational proxies, ISPs and to some degree also ad-serving companies can do so over several sites or even all of a user's Internet history.

And finally, whilst current threats are already enough cause for concern, one should consider possible future threats. Dataveillance, the aggregation of personal data from multiple sources, is noted by Oliver [77] as another threat to electronic privacy. The example of DoubleClick and Abacus showed how the merger of different databases can have serious implications on personal privacy.

The threat posed by dataveillance will continue to grow as systems become progressively integrated. Consider the case whereby medical insurance companies start offering free Internet access, or similarly, if multi-discipline companies such as Virgin acquire an ISP as well as an insurance division. Merging an individual's medical history with one's browsing history will certainly infringe on the individual's privacy. This scenario is not unlikely as already in South Africa medical companies are offering credit card facilities giving them the ability to record user purchases and simultaneously track their medical status.

Electronic privacy is clearly an issue that should be addressed.

1.1.2 Privacy protection

Privacy protection can be enforced by legislation, can be self-regulated by obtaining approval certificates from third parties or can be facilitated through the use of technology [23].

Although legislation (or self-regulation if legislation is absent or insufficient) is necessary, it only provides limited assurance that private data is not misused. A formal agreement between the individual and the entity, using his or her private data, is usually needed. According to a set of guidelines published by the OECD (Organisation for Economic Co-operation and Development), this agreement should address the following: *collection limitation*, *data quality*, *purpose specification*, *use limitation*, *security safeguards*, *openness*, *individual participation* and *accountability* [76]. Such agreements exist on the Internet in the form of Web site privacy policies.

However, privacy policies suffer from a number of problems. Olivier [77] notes the following issues: (1) policies are stated in the policy holder's own

terms, (2) the user has no option to "opt-in" or "opt-out" of certain clauses, (3) the user often has no alternative and is forced to agree to the terms in order to use the service, (4) policies are often vague and give the policy holder too much leeway and (5) users usually do not read them at all. Further inadequacies of policies and cases where policies have failed are discussed elsewhere [8].

It should additionally be noted that policies often do not cover implicitly collected information such as clickstream data and policies do nothing to protect the user from an unauthorised third party.

Noteworthy is the proposed Platform for Privacy Preferences initiative (P3P) [89], which is an attempt at giving users more fine-grained control over their personal privacy. However, it still suffers from many of the above-mentioned problems.

It is noted that when online interactions require personal information (as in the case of online purchases), policies are useful and often the only viable solution. Alternative privacy-enhancing technologies do exist; these include encryption and architectures involving trusted third parties [86, 67]. However, these still necessitate the additional use of privacy policies.

When the exchange of personal information is not required, anonymity should be considered as it is a strong and useful form of privacy protection.

This dissertation focuses on anonymity.

1.2 Anonymity

When considering the OECD guidelines, anonymity addresses the *collection limitation principle*. Withholding personally identifiable information prevents others from collecting any private information at all.

Search engines are an excellent example of where anonymity could be particularly useful. Search queries can reveal a great deal about an individual's work, hobbies, personal preferences and private life [2]. Consider having access to someone's search queries submitted over a period of a couple of months or years. Such information could be used for targeted marketing purposes or worse, could reveal incriminating or sensitive information.

In fact, at the time of writing America Online, an American service provider, had released a collection of 20 million Web queries collected from roughly 650 000 users over three months from March 2006 to May 2006 [3]. This data is now public information on the Internet.

A paper by Aljifri et. al. [2] presents interesting research on how online search engines, in particular, infringe on an individual's privacy. Although many of their arguments apply equally well to other Web sites, a number of factors make search engines particularly threatening. Search engines employ elaborate cataloguing and indexing schemes. Extensive click and user tracking is employed in order to rank search results and to display personalised advertisements, and lastly, individuals frequently require search functionality and therefore frequently use it [2].

By browsing anonymously one could prevent search engines or other Web sites from performing user tracking or user profiling. Ideally one would also want to be anonymous from the Web proxy's or the ISP's point of view.

Two common methods of identifying individual Web surfers were already discussed. These were cookies and the source IP address.

While most browsers allow the individual to disable cookies in an attempt to safeguard their browsing privacy, a user is still required to have a unique IP address. This effectively means that disabling cookies only makes user tracking more difficult but not impossible.

One might argue that where IP addresses are assigned dynamically within an organisation or ISP network, only the organisation or the ISP can link the individual to the respective IP address. Whilst this is correct, other methods of linking IP addresses to individuals do exist. A user who logs into his Google mail account reveals his IP address to Google. Google will be able to link all subsequent Google search queries from that IP address to that individual. Brandi and Oliver [17] identify inference attacks as a method whereby the user's identity is inferred based on historic user data.

However, in both cases the user might additionally require protection from his organisation or ISP. Using a dynamically assigned IP address does not protect the user because organisations or ISPs control the IP assignment.

This raises the following question: how does one effectively hide the re-

quester's IP address from Web servers, an organisation, an ISP or other observers?

1.2.1 Connection anonymity

Gabber et. al. [48] introduce the terms *connection* and *data* anonymity. Data anonymity is achieved by (1) removing identifying information from communicated content, (2) the use of pseudonyms or (3) through cryptography [48]. Connection anonymity protects the actual communication channel between sender and receiver but is less concerned with the data that is transmitted. A clear distinction is thus made between transferred data and channel meta-data.

Data anonymity is reasonably well-established – clearly seen by the extensive use of cryptography mechanism on the Internet such as secure sockets and certificates. Connection anonymity has proven to be less successful [51]. To prevent user tracking and profiling, users can disable cookies and are able to control what identifying information is submitted; however, the means of securing against IP address tracking are less mature and infrequently used.

A fair amount of research has been dedicated to building and analysing various forms of connection anonymity technologies. However, these technologies are not widely used. Most either require an individual's confidence in a trusted third party or impose considerable overheads thus making their use too cumbersome for general or daily use. Anonymity mixes (introduced in section 2.3), for example, employ expensive mixing techniques and are thus not very suitable to interactive or real-time media.

1.2.2 Crowds

One anonymity technology that is of particular interest to this dissertation is Crowds [90, 91]. Crowds offers anonymous Web browsing by collecting users into a group called a crowd. Members of this group collaborate by forwarding Web requests among themselves before passing them to a specified Web server.

Anonymity is achieved through a concept called *plausible deniability*. Each node could potentially be the original requester. However, every node can also deny this and claim that it simply forwarded the request on behalf of some other node.

Plausible deniability creates uncertainty as to who the original requester is. Note that the concept of plausible deniability does not apply to encryption. Encryption attempts absolute secrecy. In anonymity research, plausible deniability is often linked to delegation of responsibility. In Crowds, no single node assumes responsibility for a request. Instead, responsibility is deferred to the whole crowd. Plausible deniability is an important concept that features throughout this dissertation.

This dissertation introduces an anonymity technology that also offers shares some similarities with Crowds but specifically aims to improve thereon. Crowds is discussed in more detail in section 2.6.

1.3 Scope and purpose

It has been shown that the lack of online privacy should be a cause for concern. Individual users risk being tracked and profiled by Web sites, ad-serving companies, organisational proxies, ISPs and other unauthorised observers.

Anonymity is useful in many cases where personalised services are not required. This includes performing Web searches and general Web browsing. Solutions already exist but are not commonly used and suffer from a number of disadvantages.

The scope of this dissertation covers anonymity on the Internet and covers current anonymity technologies. Particular interest is taken in connection anonymity solutions that offer anonymous Web browsing. Traditional anonymous activities such as voting, counselling, whistle-blowing, refereeing and voicing political and other dissent should be included as well as general or daily Internet activities that could potentially lead to long-term user profiling.

1.3.1 Problem statement

The literature review identified a number of problems with existing solutions. Most anonymity solutions are centralised or partially centralised thus requiring trust in the operators. Available systems are either vulnerable to a host of attacks or are too inefficient and cumbersome for daily use. A sufficiently adequate solution has still not been found contributing to the fact that anonymous browsing is not commonly performed.

The purpose of this dissertation is to propose an anonymous Web browsing protocol called Polar, which is (1) fully distributed, (2) offers adequate levels of anonymity and (3) enables users to browse the Internet anonymously without overly complex mixing techniques.

More specifically we wish to show how Polar achieves plausible deniability and how it offers various improvements on the original Crowds [90] design. The original Crowds protocol suffers from inadequate routing, features a partially centralised architecture and offers poor levels of anonymity. Subsequent solutions have offered improvements, however, these are still far from perfect thus prompting further research.

1.3.2 Methodology

An extensive literature survey will explore the current state of anonymising technologies. A solid understanding of current research will provide a greater insight into the strengths and weaknesses of existing solutions.

A framework will be developed to provide a structured approach to anonymity design factors, techniques and objectives. This should assist in identifying problems of current solutions. Given these problems, a set of objectives will be proposed that a connection anonymity technology should fulfil.

Our proposed solution to these problems is a model of an anonymous Web browsing protocol. Strengths and weaknesses of the model will be analysed by considering different attacks vectors. Enhancements to the model will be made where appropriate. The model should address all objectives as best it can.

Finally, a proof-of-concept prototype is implemented and detailed, proving the viability of the model.

1.3.3 Research process

The first year of this research was spent reading background material and getting a solid foundation in privacy solutions and in particular anonymity techniques. An extensive review of peer-to-peer technologies was conducted.

The idea of using a peer-to-peer overlay in forwarding anonymous content was conceived in conjunction with Neumann and Olivier [111] and details were first published at an international conference [111]. A follow-up paper that took a more theoretical perspective on anonymity techniques was subsequently written and published [112]. A proof-of-concept prototype was developed in early 2006 giving greater insight into the details of the Polar model. The write-up of this dissertation started with the conference papers and continued until completion in October 2006.

1.4 Overview

Chapter 2 explores the current state of connection anonymity. This is achieved by performing a survey of the literature.

A meta-level look at connection anonymity is presented in chapter 3. An analysis is made on how connection anonymity has evolved and how and why certain design choices are made. A conceptual framework describing what we consider to be important connection anonymity factors is proposed. Design factors, fundamental connection anonymity strategies and objectives are considered.

Chapter 4 applies the framework to Polar and refines our problem statement and solution. Shortcomings of current anonymity technologies are identified. We consider how Polar could solve these.

This dissertation proposes the use of a structured peer-to-peer routing overlay for use in Polar. Peer-to-peer overlays are discussed and categorised in chapter 5. Several overlay protocols are considered; the protocol chosen

for use in Polar is Pastry [97].

Chapter 6 presents the Polar model. Our main ideas and contributions are discussed and analysed. Polar is analysed from an architectural as well as a functional level. Implementation details are presented in chapter 7.

Finally, this dissertation is concluded in chapter 8.

Chapter 2

Anonymising Protocols

2.1 Introduction

This chapter offers an overview of sender and receiver anonymity as well as a brief introduction to anonymous publishing. These technologies provide the foundation upon which Polar is built. A good knowledge thereof is beneficial for a thorough understanding of the research presented in this dissertation.

Specific interest is placed on different methods of achieving anonymity as well as their advantages and drawbacks. First a general introduction to proxies and their adoption as anonymity facilitators is given. This leads on to a discussion about the limitations of simple proxies and improvement made by subsequent technologies.

The order in which the anonymity solutions are given does not follow a strict chronological order but instead groups the technologies according to their approach, their complexity as well according to their intended application domain. We conclude with a discussion on Crowds [90], Flocks [78] and Tarzan [45] which are most relevant to Polar.

2.2 Trusted and semi-trusted proxies

Proxies were originally used for caching to conserve the use of external bandwidth. However, proxies also provide a limited means of identity hiding and

thus feature significantly in connection anonymity.

To our knowledge, dc-nets [21] and variations thereof are the only anonymising solutions that attempt anonymous communications without the use of proxies. Instead broadcasts are used to facilitate a multi-party computation. Although secure, a dc-net does not scale well and is easy to disrupt; a single corrupt member can disrupt the service. A dc-net is only remotely related to the work on Polar and will therefore not be discussed further. Instead we focus on technologies that utilise proxies.

2.2.1 The Penet remailer

The Penet remailer (also referred to as `anon.penet.fi`) [58] was one of the first widely used anonymous remailers. It was started by Johan Helsingius in 1993 and consisted of a mail server that offers anonymous and pseudonymous email.

The server strips the sender's real address from the email header before forwarding the message. It thus functions as a simple mail proxy. Replies to anonymously sent emails are made possible by providing a pseudonymous return address. The respective real email address is kept in a correspondence table by the remailer.

The Penet remailer, although simple and efficient, has numerous drawbacks. Because the service is centrally managed, it is susceptible to a large number of attacks, most notably *denial of service* and *compulsion attacks* [88].

Compulsion attacks include external means of acquiring sensitive user information through extortion, bribing or legal subpoenas. Legal pressure forced the remailer to reveal identities of certain users [59]. The service has since shut down fearing further legal action.

The threat of a compromised system could result in all users' identities being revealed including a history of their Web or mail usage. There is also the threat that system administrators willingly release or sell information. Thus, a considerable amount of trust in the integrity of the system *and* the administrators is required.

2.2.2 Anonymizer

Anonymizer [7] operates similarly to the Penet remailer but offers anonymous Web browsing. It thus suffers from similar trust issues as the remailer. However, a Web request is only active for a relatively short duration allowing for the network connection to stay open thereby eliminating the need for long-term correspondence tables.

Whilst many other anonymising technologies have failed, Anonymizer has been operational since its inception in 1995. Goldberg [51] attributes the success to the following: *"compared to other infrastructure-heavy attempts, Anonymizer.com has a relatively simple architecture, at the expense of protecting against a weaker threat model. But it seems that a weaker threat model is sufficient for most consumers, and we are starting to see other companies similarly relaxing their threat models"*.

Anonymizer's simple architecture offers the following advantages:

- It is simple to use and requires no additional software. HTTP requests are simply relayed via a proxy.
- It is a simple design that imposes little overhead thus allowing for fast response times.

As Goldberg [51] rightly states, it caters for users who require protection from a very weak threat model. These users could be classified as *low-sensitivity* users; they are often not concerned about (or possibly even aware of) sniffing and traffic analysis. Simple identity protection from proxy and/or Web server logs is all that is required.

Anonymizer and other similar solutions have introduced a number of additional, value-adding features [51]. These are mostly data anonymity techniques such as encryption of the communication channel, removal of identifying information embedded in the body of the request and numerous spyware and adware filters.

Another feature or solution is proposed by Gabber et. al. [48]. Many sites ask for some identification (often username or email address) to verify a user's registration and to provide a personalised service. Their solution is called

the Lucent Personalized Web Assistant (LPWA) and manages pseudonyms for use with specific Web sites.

These additional features are not central to our work on connection anonymity and hence will only be mentioned in passing.

2.2.3 Type I cypherpunk remailers

Type I remailers [81] improve on the design of the previous generation of remailers by adding distributed trust and public-key cryptography – the Pretty Good Privacy (PGP) public key infrastructure (PKI) is used. The remailers get their name from the cypherpunks, a group of cryptography enthusiasts and professionals who first published details about them [81].

Cypherpunk remailers allow for multiple remailers to be chained together. The remailers can be located in different jurisdictions making legal attacks less probable and more difficult.

Cypherpunk remailers function as follows: upon receipt of an email, it is decrypted and identifying information is removed from the email. The email is either forwarded in encrypted form to another remailer or is delivered to the final destination.

Replies are facilitated through the use of reply blocks. The remailer's public key is used to encrypt the email address of the sender. This encrypted reply block can be published or can be inserted into a special header of an email. Replies to such emails are handled by the appropriate remailer, which decrypts the contents and forwards the email to the original sender. Remailers can be chained together by supplying a corresponding reply block for each link in the chain. Each remailer can only decrypt its own reply block before passing the email to the next remailer or to the final recipient. This is an improvement over correspondence tables since return addresses are no longer held with the remailer but accompany the email in an encrypted form.

2.2.4 Nym servers

Nym servers [72, 16] (short for pseudonym servers) act as gateways between conventional email and anonymously sent email. Instead of using a previously

acquired reply block and sending the email to the appropriate remailer, nym servers allow users to send email to a pseudonymous email address. Nym servers map pseudonymous email addresses to locally stored anonymous reply blocks. When a nym server accepts an email addressed to one of its pseudonymous email accounts, it looks up the respective reply block and forwards the email to one of the cypherpunk remailers.

Reply blocks are suited to store and forward mediums such as email. Polar, similarly to Anonymizer, facilitates short-term queries and hence an alternate solution is required.

2.3 Chaum's mix

The previous solutions assume a weak threat model and do not protect against a multitude of traffic analysis attacks.

The dangers of traffic analysis were already perceived as early as 1981 when Chaum published his paper titled "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms" [20]. Chaum presents a mix-net protocol (at the application layer) that offers anonymous emailing. This protocol forms the basis for the type II remailers. A considerable amount of research has gone into mix-nets resulting in many variants of the original protocol. The original protocol is described next whilst in chapter 3 some of the more recent mix-nets are discussed.

Chaum proposes the concept of a mix that hides the correspondence between incoming and outgoing messages. This is achieved by

- imposing strict size-invariance through slicing and padding,
- performing cryptographic transformations on the message,
- batching and reordering of messages and
- providing cover traffic which (to an outsider) is indistinguishable to real traffic.

The cryptographic measures proposed by Chaum merit further explanations. Users of the mix network encrypt the contents of their message M

using the public key K_a of the recipient a. Chaum proposes that a random string R be encrypted with the message so as to avoid any brute force attacks (i.e. guessing $X = Y$ by comparing $K(X) = K(Y)$). The recipient's email address A_a accompanies the encrypted block. Thus, the message sent to the recipient looks as follows:

$$K_a(R_0, M), A_a \tag{2.1}$$

A series of mixes can be chained together to form a *cascade*. This, combined with the layered cryptographic measures, protect the system from corrupt and colluding mixes without the need for a universally trusted authority. Additionally, the distributed architecture allows for mixes to be operated from within multiple administrative domains.

Link-to-link encryption is performed at each mix in the cascade otherwise simple *message coding attacks* are possible (i.e. a direct comparison of the byte code of two or more messages). The encrypted message sent by the user thus looks as follows:

$$K_n(R_n, K_{n-1}(R_{n-1}, \ldots (K_a(R_0, M), A_a), \ldots)A_{n-1}), A_n \tag{2.2}$$

Here n is the nth mix from the recipient. Decryption is performed in a reversed order of the encryption process.

Chaum also proposes a solution for untraceable return addresses. Return addresses are essentially reply blocks with nested layers of encryption corresponding to the reverse order of mixes back to the original sender. The sender A_b would embed the following reply block in a message sent to a recipient:

$$K_1(R_1, K_2(R_2, \ldots (K_n(R_n, A_b), A_n), \ldots)A_2), A_1, K_b \tag{2.3}$$

Here each $R_i \in \{R_1, R_2, \ldots, R_n\}$ is a random string which also acts as a key. This key is a public or symmetric key generated by A_b. Upon receipt of a reply block, recipient A_a encrypts its response M_a using the key K_b. Each mix K_i uses key R_i to add a layer of encryption to $K_b(R_{a_0}, M_a)$. The final mix reveals address A_b and delivers the encrypted response. The original

sender can remove all layers of encryption because it created all keys.

This method of return addresses also provides a solution to certified email since the successful receipt of a return message implies the successful delivery of the original message.

Elsewhere [56, 73] other remailers have been successfully implemented in close accordance to Chaum's original design. Variations on the routing and the mixing (or obfuscation) techniques have also been proposed. These are covered in chapter 3.

On a final note, because of the performance and resource implications, mixes are mostly used for low-latency, store-and-forward mediums such as email. Some variations have been adapted to cater for Web and other applications as well. These, however, generally use Onion Routing for the actual data transfer.

2.4 Onion Routing

Onion Routing is related to Chaum's original mix-net protocol but is more suited to Web browsing and other interactive or even real-time media. Although many Onion Routers make use of various mixing strategies, Onion Routing, in the strictest sense, only refers to the actual routing of messages and not the batching and reordering.

The concept of an Onion Router was first conceived by Goldschlag et. al. [54] and has been detailed in a number of publications [54, 110, 53]. It aims to protect the privacy of the sender and receiver of a message whilst also preventing a compromise of the system through any number of (except all) compromised routers. One honest node is sufficient to retain anonymity.

As with mix-nets, Onion Routing also uses layered encryption. The difference lies in the content that is encrypted. Whereas mix-nets wrap the actual message, Onion Routers use Onions to establish an anonymous, bi-directional virtual circuit between two communicating parties (over a number of participating nodes). The virtual circuit then relies on computationally less expensive symmetric keys to encrypt a stream of data.

A communication consists of three steps: connection setup, data transfer

and connection tear-down. An Onion is used during the connection setup phase and is generated by the first (or the initiator's) Onion Router. The Onion is constructed similarly to the layered message in a mix-net: a message is successively encoded with layers of encryption corresponding to each node in a chain of routers.

Unlike mix-nets, an Onion contains information used to establish a virtual circuit – each layer not only contains the next Onion but also a forward and backward symmetric key. Forward and backward symmetric keys are shared with the succeeding and preceding Onion Routers respectively. This effectively allows for a bi-directional, encrypted and anonymous communication channel.

For example, if Alice wishes to communicate anonymously with Bob, she will instruct any number of Onion Routers – say Onion Routers A, B and C – to establish a communication channel to Bob. Alice creates an Onion and passes it to the first Onion Router, A. Only A can decrypt the outer layer of the Onion, thus retrieving further instruction. In our example, A is instructed to establish a tunnel with B and pass the "peeled" Onion to B. Pre-shared keys allows A to safely exchange a symmetric key with B thus setting up a secure communication tunnel between A and B. The same procedure is followed by B, this time to Onion Router C; a different symmetric key is used for each link. Finally, C establishes a connection with Bob.

All subsequent communication from Alice to Bob is passed through the encrypted tunnel between A, B and C. Bob can identify the last Onion Router but has no way of telling who initiated the communication. Similarly, each Onion Router only knows the previous and the next Onion Router.

Onion Routers typically forward requests on behalf of a large number of users. The number, combination and sequence of routers can differ for each established communication channel. This makes it difficult for adversaries to trace each hop in a particular chain until the original sender is reached.

Onion Routing also allows for connection-based and connection-less traffic. Reply Onions allow for the responder to reply after the original virtual circuit has been broken. The reply Onion closely resembles Chaum's method for return addresses.

Whilst Onion Routing focuses on a strong insider threat model (it protects against colluding proxies) it fails to protect against a number of traffic analysis attacks. Onion Routing makes no attempt to hide or obscure related information such as message size or send or receive times. An attacker can gain such knowledge by performing traffic analysis. This does not always lead to identity exposure but can weaken the level of anonymity offered by the system (i.e. there could be less uncertainty as to who the original sender could be).

Many solutions therefore incorporate some mixing functionality which on the downside directly affects performance.

A more recent Onion Router called TOR [37] (The second-generation Onion Router) offers some improvements over the original design. These improvements include perfect forward secrecy, congestion control, directory servers and integrity checking. More information about Tor can be found in their paper [37] or on their Website http://tor.eff.org.

2.5 Anonymous publishing

In order to provide a well-rounded overview of anonymising technologies, a brief overview of anonymous publishing is presented. Although Polar does not cater for document publishing, one should take note of the differences between Polar and anonymous publishing and consider some anonymous publishing solutions that introduce interesting and related methodologies.

There are some notable differences between anonymous publishing and the previous solutions offering sender or receiver anonymity. Previously mentioned solutions attempt anonymous, point-to-point communication channels. Anonymous publishing goes beyond just the channel and includes document storage and retrieval. The object of anonymity is not only the anonymity of a relatively short-lived request or response but rather a long-term document that has an associated author and potentially many readers. This introduces a new set of challenges including document publishing, disk space allocation, document distribution and document revocation.

Note that anonymous publishing differs from server anonymity (such as

that offered by Janus [95]). This is because the document and author is anonymous as opposed to the provider or server of the document. Server anonymity might be considered to be an early attempt at something anonymous publishing does much more effectively.

Dingledine et. al. [34] identify different agents in anonymous publishing and enumerate different notions of anonymity for each. The different notions include: *author*, *publisher*, *reader*, *server*, *document* and *query* anonymity. These notions are layered over the anonymous communications channel. At first glance it might seem as if *reader* and *query* anonymity are relevant to Polar. However, Dingledine et. al. [34] make no distinction between servers and proxies and hence a direct relation can not be made.

A brief discussion of some of the more interesting anonymous publishing systems follows.

2.5.1 TAZ servers and the Rewebber network

The Rewebber network [52] can be likened to the inverse of an anonymising proxy. The Rewebber network does not protect the requester but rather provides a means of hosting and locating anonymous content. Anonymity is achieved by using chains of nested and encrypted URLs – similar in nature to Chaum's wrapped messages. The content is hosted by the Rewebber network but is only retrieved after a number of servers collaborate in sequentially decrypting the URL until the final server locates the appropriate content.

2.5.2 The Eternity Service

Anderson proposes the Eternity Service [4], an anonymous, write-once data store. It does not support document removal but instead aims to make documents always available through the use of scattering and redundancy techniques. The main goal is availability and the prevention of denial-of-service attacks. The use of an anonymous payment scheme is also suggested although none is detailed.

The Eternity Service has a number of unresolved issues such as file indexing and anonymous, digital cash [86, 67]. Mojo Nation [74] was a subsequent

attempt at an anonymous publishing system [74] that employs digital cash, however, it employs trusted third-parties and is hence not considered further.

2.5.3 Free Haven

Free Haven [34] uses a reputation system to avoid the problems of digital cash. Servers employ a content-neutral policy that allows them to trade disk-space with other servers based on a $size \times duration$ cost basis. The documents are split across numerous servers for a publisher specified lifetime.

Free Haven uses inefficient broadcasts for communication. It is interesting to note that Free Haven's inventors [34] acknowledge the inefficiency of the system and point out that system efficiency and the system's perceived benefit could be more important to an end user than its anonymity properties.

2.5.4 Freenet

Whilst both the Eternity Service and Free Haven display some properties of a truly distributed system, Freenet [25, 24] is the first anonymity system discussed thus far to consider an adaptive peer-to-peer network as its underlying communications facility. It would therefore be interesting to note the capabilities and advantages Freenet has over other similar client-server architectures.

Freenet is a co-operative distributed file system that employs transparent lazy replication [25]. It aims to achieve anonymous file storage and retrieval for both producers and consumers of information. Participants store and propagate documents without knowing who the author or the readers are.

Noteworthy is Freenet's decentralised architecture. Freenet claims its search is not a broadcast search. In reality however, its routing differs only slightly from a proper broadcast. Some intelligence is built into the routing; it is, however, still classified as an unstructured routing overlay.

A peer-to-peer overlay is here used as an all-encompassing term for any network consisting mostly (but not exclusively) of identical peers. In chapter

5 different types of peer-to-peer overlays are introduced. Of these, unstructured routing overlays are the most basic and most inefficient.

In Freenet, nodes attempt to route requests to the most likely location. Nodes in Freenet have local and not global knowledge. Routing knowledge thus improves over time as nodes discover other Freenet nodes and "learn" where particular documents might reside. Requests are propagated through the network until a hops-to-live count reaches zero.

Routing is inefficient and slow and object location is not guaranteed. It might prove interesting to consider a structured routing overlay for use in Freenet. This is left as possible future work and instead the benefits of such an overlay are considered for use in anonymous browsing only.

Other anonymous publishing systems worth noting are Publius [68] and Intermemory [49]. However, the remainder of this chapter continues the notion of peer-to-peer networking, introduced in our discussion on Freenet. We next consider those solutions whose architectures have strong peer-to-peer influences.

2.6 Crowds

Reiter and Rubin present Crowds [90, 91] which aims to make browsing anonymous by collecting users into a group called a crowd. Members of this group collaborate by forwarding Web requests amongst themselves before passing them to a specified Web server. The choice of forwarding to another proxy instead of to the end server is a random decision based on some system-wide parameter $\alpha > \frac{1}{2}$. Each member acts as a proxy, but is also able to issue its own request on behalf of the user.

A Web transaction consists of an original requester, the actual request, a number of proxies that forward the request, the end server as well as the end server's response. A request maps out a route through the crowd before finally reaching the Web server. The response traverses the same route but in reverse order.

A Web server is thus only able to trace the request back to the final proxy and is not able to determine where the request originated from. In fact, every

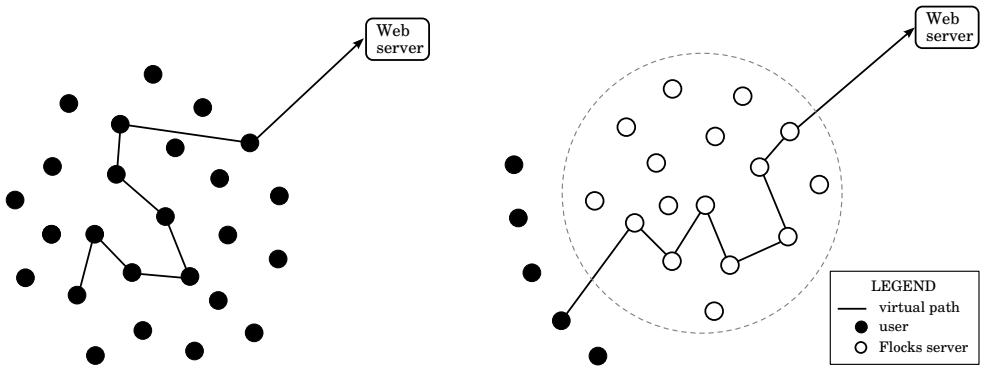
proxy along the route only knows the previous and the next proxy (or the end server). Only the original requester knows who issued the request. Sender anonymity is achieved by allowing members to get lost in a crowd.

Crowds suffers from a number of disadvantages.

- A central server for node discovery and key distribution is required. Crowds is a network of peers but still suffers from many of the trust issues typically related to client/server architectures.
- Each node establishes a static virtual path which is used for multiple requests. Once the path has been compromised all prior and subsequent requests are exposed. Also, static paths need to be reconstructed each time a new node joins the network.
- A simple routing algorithm with highly variable path lengths is employed. The initiating node has no means of controlling the length of the path.
- Crowds requires all nodes on a virtual path to be honest nodes. Simple link-to-link encryption is employed between nodes. This requires intermediate nodes to decrypt and re-encrypt forwarded messages.

Elsewhere [22] reference is made to an enhanced version of Crowds. Central servers are no longer required for key distribution; key distribution is facilitated by the Diffie-Hellman key exchange protocol [33]. In addition, it is suggested that end-to-end encryption be used instead of link-to-link encryption. An implementation of the enhanced version of Crowds is called mCrowds [22, 5] and is aimed at the mobile Internet.

Some issues, however, have not been addressed. Polar specifically aims to address node discovery and reconsiders the suitability of static virtual paths. Path construction is revised to suit a highly dynamic network of peers as can be expected on the Internet. Additionally, Polar aims to achieve better levels of anonymity compared to Crowds.



(a) Crowds (b) Flocks

Figure 2.1: Different autonomies of Crowds and Flocks.

2.6.1 Flocks

Flocks [78] functions similarly to Crowds, but is primarily intended for use in an organisation. It essentially forms a clustered network of controlled or supervised proxies that forward requests originating from outside the Flocks network. The architectural difference between Crowds and Flocks is illustrated in figure 2.1. The figure depicts a typical path that a request and response map through a Crowds and Flocks network respectively.

Having authority over the anonymising network allows for a forensic analysis: all proxies can be forced to co-operate leading to identity exposure. In essence, node autonomy is sacrificed for forensics. Flocks is the first technology discussed thus far that prioritises forensics (or conditional identity exposure) over anonymity. On the down side, Flocks annuls some of the benefits offered by distributed systems, since trust (in the administrators) is still an issue.

Flocks additionally considers caching as a means to improve performance and to some degree also anonymity. Cached content shortens the path and bypasses the Web server.

In a follow-up paper [79] Olivier uses a simulation of Flocks to quantify performance-related issues.

2.7 Tarzan

Tarzan [45, 46] is an anonymising protocol that is closely related to Polar. Both technologies attempt plausible deniability similarly to Crowds, but do so by organising peers using a structured peer-to-peer protocol. This allows Polar and Tarzan to be fully distributed; as opposed to Crowds' partially centralised architecture.

However, various notable differences exist between Tarzan and Polar. Tarzan operates at the network layer whilst Polar does so at the application layer. This gives Tarzan the obvious advantage of being able to provide an anonymising IP layer routing protocol for a number of transport and application layer protocols. Polar focuses on the HTTP protocol and can therefore offer improvements suited to Web browsing.

Tarzan obfuscates network traffic by restricting communication to paired nodes – paired nodes use cover traffic to hide the existence of real traffic. Similar mixing techniques are not employed by Polar; instead an alternate solution is presented.

Tarzan requests are tunnelled through a randomly selected group of participants; hence the relation to Crowds (and Polar). Onion Routing is used for the tunnel setup and allows for a bi-directional tunnel. Although Polar intends to use Onions as well, it differs in how the request is transferred and how the tunnel is established. In addition, Tarzan's tunnels are long-lived whilst those of Polar are short-lived – Polar tunnels are only active for the duration of a single Web request.

Node discovery is performed similarly in both Tarzan and Polar. A structured peer-to-peer routing algorithm is employed. Polar and Tarzan use Pastry and Chord respectively. A discussion on peer-to-peer protocols including Pastry and Chord is presented in chapter 5.

Further differences should become evident as the Polar model is detailed.

2.8 Conclusion

This chapter offered an overview of the evolution of anonymising technologies. The first generation of anonymising proxies such as Anonymizer and the Penet remailer as well as the second generation of Cypherpunk remailers were considered. The fundamental mixing techniques presented by Chaum, upon which numerous other anonymising technologies are based, were also discussed. It was noted how Onion Routing is better suited to low-latency communications than Chaum's protocol. A brief introduction to anonymous publishing was also given and the differences to anonymous Web browsing were noted.

Polar adopts the idea of employing anonymity seekers as active participants. This idea was first presented by Crowds and subsequently revised by an enhanced version of Crowds. Flocks and Tarzan extended the notion. Tarzan is of particular interest to Polar because a structured peer-to-peer overlay is used.

An evaluation of peer-to-peer protocols is presented in chapter 5, but first, an anonymity framework is presented in the following chapter.

Chapter 3

A Framework for Connection Anonymity

3.1 Introduction

In this chapter a more fundamental perspective on connection anonymity is presented. This perspective is defined and explained by a framework.

The framework was first published in one of our papers titled "Towards a framework for connection anonymity"[112]. The research was completed during the earlier phase of the masters. It assists in identifying strengths and weaknesses of current solutions and allows us to position Polar in relation to these technologies.

The framework draws heavily from existing connection anonymity technologies. Our contribution lies in the classification and categorisation of the techniques.

Chapter 4 subsequently discusses how the framework can be applied to Polar.

3.2 Purpose

Within the context of this dissertation the framework aims to achieve the following:

- provide a more structured and formal view of connection anonymity,
- identify strengths and weaknesses of current anonymity techniques,
- position Polar in relation to existing protocols.

A meta-level approach is taken in order to identify, classify and categorise existing connection anonymity techniques.

Connection and data anonymity are closely related; connection anonymity often uses data anonymity techniques such as encryption. This dissertation and the framework consider data anonymity only where it directly aids connection anonymity.

3.3 The Framework

The framework is depicted in figure 3.1. It consists of three sections namely the *design factors*, the *connection anonymity strategy* and the *objectives*. These are covered in sections 3.4, 3.5 and 3.6 respectively. The three sections effectively address the problem statement, the methodology and the desired result commonly observed in anonymising technologies.

- The *design factors* define boundaries within which an anonymity service must operate including any threats it can expect. Anonymity versus unlinkability, the application domain and the threat model are discussed.
- The *connection anonymity strategy* covers the measures taken by a system in order to achieve anonymity. A distinction is made between a strategy and many techniques performed by that strategy. Techniques present researched means of protecting user identities; message batching, reordering and slicing are all examples of connection anonymity techniques. A combination of techniques constitutes a strategy and identifies a certain anonymity technology.

- Lastly, two *objectives* are discussed. The first relates to the degree of anonymity offered by a system and the second considers efficiency or usability factors.

A more detailed discussion on each follows.

3.4 Design factors

An analysis of the literature survey, identified the following noteworthy design factors: anonymity versus unlinkability, the intended application domain and the expected threat model. Each is clearly indicated on the framework in figure 3.1 and is discussed in more details next.

3.4.1 Anonymity and unlinkability

The type of anonymity offered by anonymising technologies is an important distinction that affects later design choices [112].

A definition of anonymity is given by Pfitzmann et. al. [83]: *"anonymity is the state of being not identifiable within a set of subjects, the anonymity set"*. One should also note the difference between anonymity and unlinkability: *"two or more items (e.g. subjects, messages, events, actions) that within a system are no more no less related than they are related concerning the a-priory knowledge"* [83].

The anonymising technologies as discussed in chapter 2 do in fact not achieve anonymity (as per the definition) but rather attempt unlinkability. On the Internet, participants have an IP address making them clearly distinguishable from from each other. Instead, an attempt is made at protecting the communication channel by hiding the relation between a message (an email, Web request or other) and the sender of the message.

Polar, for example, forwards Web requests to Web servers such that the request can not easily be linked to the original requester. All participants of Polar are clearly distinguished by unique identifiers – this includes their IP address. However, Polar makes an attempt of unlinking the identifier and address from a message.

Figure 3.1: A conceptual framework for connection anonymity

A paper from 1987 by Pfitzmann et. al. [82] classifies three types of anonymity: sender anonymity, recipient anonymity and unlinkability of sender and receiver. The revised classification of 2001 [83] more clearly differentiates between *unlinkability* of sender or receiver and the message. However, a clear separation of recipient and sender anonymity, as given in 1987, still exists [114, 24, 95]. The framework therefore also differentiates between sender and receiver unlinkability as indicated by points $I.a$) and $I.b$) in figure 3.1.

The framework considers unlinkability between sender and/or receiver and short-lived messages. Long-term documents as used by anonymous publishing is not covered. Publishing systems go beyond just protecting the communication channel and additionally address storage and anonymity issues of long-term documents (as opposed to short-term messages) [34]. The framework could be extended to include anonymous publishing. This is, however, not directly relevant to Polar and is left as future work.

In this dissertation the terms anonymity and unlinkability are used interchangeably. In all cases, except where explicitly mentioned, anonymity refers to unlinkability.

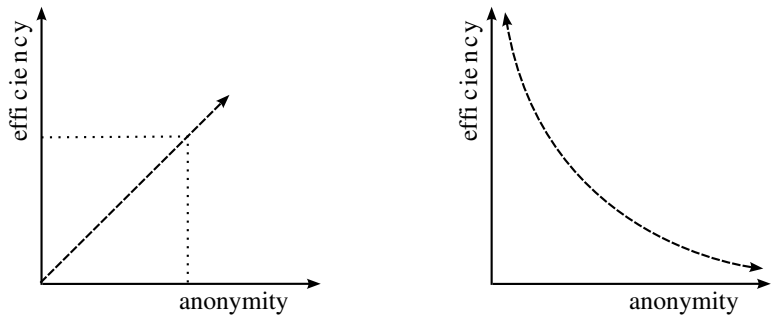
The framework thus acknowledges the discrepancy between anonymity and unlinkability and differentiates between sender and receiver anonymity.

3.4.2 Application domain

It makes sense not only to consider the type of anonymity but also the application domain intended for an anonymity technology. Typical application domains include emailing, Web browsing, Web services and Internet telephony. The respective protocols include SMTP, POP3, IMAP, HTTP, SOAP and SIP. Each requires different volume, speed and latency characteristics and also gives an indication of the type of data transmitted.

Elsewhere [111] related research shows how privacy concerns commonly associated with email and Web browsing can be extended to include Internet telephony.

Of particular interest is the nature of the *communication channel* and the *sensitivity* of the transferred data. Thus, the application domain imposes



(a) (b)

Figure 3.2: The relation of anonymity and communication efficiency.

requirements relating to the desired communication efficiency as well as the level of anonymity. Optimising both properties has proven to be difficult.

Dingledine [37] notes how anonymity research on a high latency push medium, such as email, has led to well established and robust anonymous remailers; however, similar achievements are more difficult for an interactive low latency communication medium such as Web browsing. System parameters such as message delays or the number of routing hops, present a trade-off between increased levels of anonymity and routing efficiency.

A desired situation is depicted in figure 3.2(a) whereby anonymity and efficiency is increased to a point where both values reach acceptable levels (i.e. for some threshold α and β, anonymity $> \alpha$ and efficiency $> \beta$). In reality, however, it can be observed how the one is roughly proportional to the inverse of the other; this is illustrated by figure 3.2(b).

We argue that in many application domains it makes more sense to measure the combined level of anonymity and efficiency. The application domain is thus defined in terms of the expected sensitivity of the data as well as the desired efficiency. Most anonymising technologies focus on anonymity alone and only a few authors [51, 52, 34] note the importance of efficiency.

Goldberg notes: *"But it seems that that weaker threat model is sufficient for most consumers, and we are starting to see other companies similarly relaxing their threat models"* [51]. Similarly, Dingledine states *"in many*

cases, the efficiency and perceived benefit of the system is more important to an end user than its anonymity properties" [34, p.20]. However, anonymity can not be forfeited at the cost of efficiency: *"is there a way to create an anonymous system with a tolerable loss of perceived efficiency compared to its non-anonymous counterpart? And what does "tolerable" mean, exactly?"* [34, p.20].

In order to identify important factors affecting the communication channel, reference is made to chapter 2, which covered many anonymous remailers and Web proxies. By analysing their communication properties, the following communication channel properties were identified: *store-and-forward*, *interactive* and *real-time* communication media. Each associates respective *latency* and *volume* requirements. The nature of the communication can be categorised as either a *push* or a *pull* technology. For example, email is a push technology whereas HTML is a pull technology.

These communication properties affect the choice and suitability of various routing and obfuscation strategies. Hence, they feature in the framework as additional options under the heading of "Application domain".

3.4.3 Threat model

Threats very often present the focus of anonymity research and should therefore be included in our framework.

Attributes that affect the severity and likelihood of attacks should include the following: (1) location diversity of nodes, (2) active or passive capabilities of an attacker, (3) the extent to which an attacker has compromised the internal workings of the system and (4) if forward secrecy can be maintained. These are discussed under the headings of "Location diversity", "Active/passive adversary", "Internal/external adversary" and "Forward secrecy".

Location diversity

The threat of a global adversary considerably complicates attempts at providing anonymity to users. This is illustrated by the illustrious techniques employed by mixes. Mixes aim to protect against a global adversary [20] and

hence require complex mechanisms to confuse and thwart attackers.

On the Internet, the difference between a global and a local adversary depends on the number of nodes located in the fraction of the network that an adversary can observe. This fraction is a collection of one or more administrative domains, also referred to as autonomous systems [42]. An administrative domain is usually managed by one entity such as an organisation or an ISP.

To prevent any one entity from observing all nodes, it would be beneficial to distribute nodes across multiple domains; this increases location diversity.

Feamster et. al. [42] present a detailed analyses of location diversity. They argue that the number of autonomous systems traversed by an anonymity technology should be taken into consideration in order to improve anonymity. Spreading the communication over numerous autonomous systems prevents any one authority from observing all communications.

This property is particularly important for Polar; the existence of a global observer could invalidate the anonymity efforts of Polar.

Active/passive adversary

An adversary capable of observing traffic poses a considerable risk; an adversary who additionally possesses the capability to actively attack a system presents a much bigger threat. An active adversary is able to alter or delete data passing through a system [114]. This is possible when the attacker controls the communication channel and/or one or more participating nodes in the system.

A number of active attacks exist. In mixes, for example, active adversaries could perform trickle or flooding attacks [100]. If an attacker has the ability to delay or delete messages, he or she could wait until the mix has flushed its messages. Thereafter, delayed messages are released one by one, never allowing the mix to batch and reorder multiple messages. Flooding attacks work on a similar principle. Each round the attacker floods a mix buffer of size n with $n-1$ messages, thereby reducing the anonymity set from n to 1.

Although more complex mixes combat these attacks through the use of dummy traffic and advanced pool-flushing algorithms, these attacks never-

theless present a considerable threat to many existing technologies.

Passive adversaries should also not be underestimated. The threats from passive adversaries are two-fold: passive adversaries can observe network traffic [102] or can analyse traffic logs [117]. In both cases, incoming and outgoing messages can be correlated, possibly leading to identity exposure.

Internal/external adversary

Adequate counter-measures depend largely on a thorough knowledge of anticipated threats. The level of sophistication of an attacker should additionally be considered. This includes considering whether attackers are passive or active or if their knowledge of the system is restricted to external information or whether internal intricacies are known.

A further distinction is made between what data is compromised. An internal adversary has knowledge of, or access to, the internal workings of a system, whereas the external adversary can only compromise the communication channel [114].

This distinction is particularly important for uncontrolled systems such as peer-to-peer networks – peers are typically not controlled or protected by a central authority. Instead, such peer-to-peer systems often allow anybody to participate. An adversary would thus be able to actively conduct attacks using knowledge of the internal workings of his locally run node.

Protecting against such an attacker can be difficult. One should therefore consider how a system recovers from an incident; this is addressed by achieving forward secrecy.

Forward secrecy

In cryptography, the terms *forward secrecy* or *forward security* refer to a condition whereby a system continues to be secure even though a previous encryption key has been compromised [27]. This can similarly be applied to anonymity. A definition given by Dingledine et. al. [34] states: *"a system is perfect forward anonymous if no information remains after a transaction is complete which could later identify the participants if one side or the other*

is compromised.". Thus, the compromise of an anonymous communication should ideally not invalidate any subsequent requests for anonymity.

For example, Wright et. al. [118] manage to degrade the anonymity of certain protocols by observing communications over a number of rounds. These systems fail to protect against replay attacks whereby two identical requests are handled in the exact same way. This opens up the possibility of brute force attacks [88] – trying all possible combinations until the winning combination is found.

Hence, an anonymous system should be adaptive and should be able to recover from an incident. Achieving anonymity is one goal, regaining anonymity after it has been lost, is another. If cryptography is used by an anonymity system, the anonymity offered by the system could benefit from periodic key renewals [27].

3.4.4 Concluding the design factors

The threat model presents point *III*. in figure 3.1 and concludes the discussion on connection anonymity design factors.

It should be clear that any anonymity technology could benefit from a thorough analysis of the threat model, the application domain as well as the type of anonymity. We hope that our framework assists in gaining a solid foundation in connection anonymity design factors.

Now that the design factors have been discussed one can proceed with the connection anonymity strategy. Suitability of a strategy depends largely on the design factors discussed thus far.

3.5 Connection anonymity strategy

Whilst many comprehensive anonymity taxonomies exist [80, 9, 116], few take a meta-level approach as is presented here. Instead of listing and comparing individual systems, a connection anonymity strategy is presented and discussed as a set of techniques.

We draw from experiences made by previous and current technologies in

order to identify and explicitly define various techniques. The categorisation, as given here, is our own. Limited means of categorising different mix types has been done before [27, 88]; however, a strict distinction between strategy and techniques as well as a complete classification thereof is not made.

Furthermore, only those techniques which directly affect anonymity are considered. Of particular interest to anonymising technologies are *obfuscation techniques*; these measures aim to thwart possible attackers thereby increasing the level of anonymity.

In section 2.2 it was noted that most connection anonymity technologies are routing-based and use one or more proxies. Routing is an important aspect of anonymity solutions. Many obfuscation techniques therefore address various routing functions or parameters.

Obfuscation techniques are discussed next, covering items *IV*.1 to *IV*.7. of the framework.

3.5.1 Route selection

Various route selection protocols, that facilitate an anonymous communication channel, are considered. More specifically, one can observe how and why certain routing strategies are chosen above others.

Single proxies

Single proxies as used by Anonymizer and the Penet remailer (section 2.2) allow for a simple and efficient implementation that is vulnerable to many attacks [51]. A centralised architecture is employed that has a single point of failure, suffers from trust issues and scales poorly. This architecture offers weak protection against many known attacks [51].

Cascades

Cascades were first introduced by the cypherpunk remailers (section 2.2.3). Cascades form a chain of a fixed number of proxies. Each request traverses all proxies in the cascade. Hence, one entry and one exit point exist.

Cascades introduce a distributed architecture thereby eliminating disadvantages associated with centralised solutions. Several mix-nets adopt this approach including Chaum's original mix [20]. Cascades are ideal for low latency communications and are therefore used in certain real-time systems [84] and some anonymous Web browsing solutions [14].

Berthold et. al. [15] argue that cascade mixes provide better protection against global adversaries than free-route mixes. On the downside, cascades restrict user choice on how to route their messages and also renders the whole system unavailable if a single link fails.

Free-route selection

Free-route selection implies that there are a number of possible routes from a source to a target. A distinction is commonly made between source and loose routing [106, p.104].

The original Onion Routing protocol [54] as well as the more recent Tor [37] and Mixmaster [73] employ source routing. The sender specifies the exact path the message must follow. Message sending fails when all specified hops cannot be traversed in the same sequential order.

Crowds [90] uses loose routing – each hop determines the next hop.

An advantage of free-route selection is that a path can be chosen based on reputation or reliability statistics. On the down-side, Berthold et. al. [15] argue that free-routing is more vulnerable to traffic analysis than cascades. However, unless traffic volume is very low or extensive traffic analysis is performed, the difference in the level of anonymity offered by the two route selection strategies is minimal.

Restricted route selection

Restricted routes were first proposed by Danezis [26]. Restricted routing uses a combination of cascades and free-route mixes to achieve a compromise. Danezis claims it offers advantages offered by both approaches [26].

Randomised routing

Crowds [90] is the best known technology that employs randomised routing – choosing the next hop is a completely random decision. Note that in Crowds each hop makes a randomised decision and hence loose routing also applies. In Crowds, the decision to forward the request or pass it to the Web server is also a random decision; however, the probability is calculated using a system-wide parameter. While routing is unpredictable, the system-wide parameter does give an adversary the means to calculate anonymity probabilities. The level of anonymity offered by such system is therefore a function of the system-wide parameter.

Not only is anonymity affected but also the overall system efficiency. Anonymity and efficiency depend on a trade-off between the average path length and the resultant degree of anonymity [78]. Olivier [78] explores the relation between the average path length and the system-wide parameter. Sui et. al. [109] analyse the expected participant payload in Crowds.

Peer-to-peer routing

Peer-to-peer networking was first introduced in the discussion on Freenet (presented in section 2.5.4).

Crowds also claims to be a peer-to-peer network [90]; however, Crowds is not completely decentralised. Crowds uses centralised servers for node discovery and key distribution. This means that while Crowds participants form part of a peer-to-peer network, a client/server model is still required to support the peer network.

A more comprehensive discussion and classification of peer-to-peer overlays is given in chapter 5. For now it is noted that structured peer-to-peer overlays present a certain class of routing protocol amongst peers. An overlay exposes its routing capabilities to other "over-lying" applications.

Structured peer-to-peer overlays are used in more recent anonymity systems such as Tarzan [45, 46] and Polar [111]. Both systems, similarly to Crowds, attempt anonymity through the use of volunteers instead of dedicated servers. Volunteers are here used to refer to users who choose to share

storage space, processing power and/or network bandwidth with the peer network. This differs to servers which are dedicated to provisioning such services.

Peer-to-peer networks are more resilient against compulsion attacks since no single entity controls the service and hence nobody can be held accountable [112]. Peer networks additionally possess the capability to become completely decentralised thereby eliminating the need for centralised servers. This also eliminates the need for server setup and maintenance.

On the down-side, peer-to-peer systems often suffer from reliability and trust issues resulting from a lack of (centralised) control [38]. This dissertation explores the benefits and drawbacks of using a peer-to-peer routing overlay for the purpose of achieving anonymity.

3.5.2 Path length

In anonymising systems the path length is important because it too offers a trade-off between the degree of anonymity and system efficiency [55].

Fixed path length

Cascades have fixed path lengths. The known path length is used in various attacks [55] to reduce the anonymity set. For example, intercepting a message that has another three hops to go, out of a total of four, identifies the source of the message as the original sender (not just as another forwarding proxy). This reduces the level of anonymity offered by the system.

Variable path length

Variable path lengths are considered to be superior to fixed path lengths [55]. Randomised routing, free-routing and peer-to-peer solutions generally employ this strategy. However, some solutions impose a bounded number of routing hops, thereby invalidating some of the advantages of unlimited path lengths.

3.5.3 Message delay

Batching and ordering of messages was first introduced by Chaum in his design of an anonymising mix network [20]. Reordering of messages prevents timing attacks. Timing attacks [88] assume a first in, first out (FIFO) ordering and correlate incoming and outgoing messages using the arrival and departure time. By delaying the forwarding of messages, multiple messages can be batched, reordered and then released in a non-sequential order thus preventing any observer from tracing a request from sender to the final destination.

Threshold delays

A mix with a threshold delay (a threshold mix) collects a fixed number of messages before releasing them. This method is used in Chaum's original mix [20] and is suitable if message volume is high and messages can be released consistently without too much delay. Since the Internet provides no guarantees of consistent traffic volumes, threshold mixes have become better suited to high latency communications such as email. Emails can tolerate long delays, thus making it a popular choice in mix technologies.

Timed delays

Timed mixes, such as that used in Lance Cottrel's Mixmaster protocol [73], restrict the amount of time a batch of messages can be delayed. This is achieved by periodically flushing the message pool. Timed mixes thus impose an upper bound on the maximum time taken to route a message (through a mix network with a bounded number of routing hops).

At first glance, threshold mixes seem more suited to low latency communications and simultaneously less secure for low volume media – compared to threshold mixes. Serjantov and Newman [101] offer a more detailed comparison of both mix types. Their paper [101] notes how the properties of message delaying techniques have not been well studied and that *"rigorous descriptions of their properties are lacking"* [101]. Thus, a simple statement about the suitability of different delay strategies is not possible without a thorough

comparison of different parameters and parameter values (e.g. pool size, traffic volumes and accepted application latency [101]).

Continuous mixes

Continuous mixes (also called stop-and-go mixes) were proposed by Kesdogan et. al. [63]. Continuous mixes are similar to timed mixes but instead of having the mix decide on an appropriate time interval, the delay is specified by the sender of the message. Every message is assigned a random time delay usually chosen from an exponential distribution function. In effect, messages are treated individually instead of in batches and the trade-off between efficiency and degree of anonymity depends on the user specified delay time.

Continuous mixes prevent flooding attacks (flooding attacks were described in section 3.4.3). Continuous mixes discard messages if they are received after the specified delay has elapsed (i.e. messages expire). This prevents an adversary from successfully delaying messages, as a significant delay results in the message expiring. Hence flooding attacks become infeasible.

3.5.4 Message release

A distinction between a message delay and a message release technique is not commonly made. With the exception of continuous mixes, delay techniques can be implemented with any one message release techniques. The use of a delay technique necessitates the use of a release technique.

Batch mixes

Batch flushing was originally proposed by Chaum [20] and presents the most simple release strategy. Here all messages collected during a single round are released at once.

Pool mixes

By design, pool mixes achieve stronger anonymity than batch mixes. In each round, only a random number of messages are released. The remaining messages are kept for the next round. This is repeated continually allowing for messages to remain in the mix for multiple iterations. Both Mixmaster [73] and Mixminion [29] employ pool mixes and both use timed delays.

A disadvantage is that messages are not guaranteed to exit a mix after a single round. Messages could potentially remain in a mix for a number of rounds, resulting in significant delays.

Continuous mixes

Continuous mixes release messages according to an exponential distribution function. The message release strategy is thus the same as the message delay strategy.

3.5.5 Message slicing and padding

Message slicing and padding was first used by Chaum [20] for the purpose of improving anonymity. Mixes that delay and reorder messages are still vulnerable to *message length* attacks whereby the size of the message is used to correlate incoming and outgoing messages. Large messages are therefore sliced into fixed-size packets whilst smaller messages are padded with random data. A packet-reconstruction protocol is needed to reassemble individual packets once they are delivered.

Serjantov and Sewell [102] make a clear distinction between message-based systems (such as mixes) and connection-based anonymity systems (such as Onion Routing). They argue that the former is used for asynchronous messaging (such as email) whilst the latter is used for low-latency bidirectional communications.

Mixes such as JAP [14], Mixmaster [73] and Mixminion [29] employ traffic slicing and padding. They, however, protect against a strong threat model for anonymous email which is a high latency communication medium. On

the other hand, performance sensitive systems such as Freedom [16] and Tor [37] opt not to use fixed-size messages.

In fact version 1.0 of the Freedom architecture initially featured fixed-size packets but was later dropped in version 2.0. The following reasons for adopting variable-sized TCP packets are given [10]: many smaller-sized TCP packets, including TCP acknowledgements are all padded to a fixed-size length (typically just below 1500 bytes). This leads to reduced useful bandwidth available to the user. Also, the motivation for fixed-size packets is annulled by the fact that other traffic analysis attacks such as timing attacks are possible and allow attackers with similar capabilities to correlate messages with or without fixed-size messages. Neither Freedom 2.0 nor Tor have message delay or sizing capabilities.

Message slicing and padding is thus only useful in combination with message delay and release techniques. And inversely: message delay and release are only useful if fixed fixed-size messages are used. And hence the benefits offer limited additional traffic analysis resistance but cost a lot in bandwidth.

A similar argument exists surrounding the bandwidth-intensive cover traffic technique. Some literature refers to cover traffic as link padding. This should not be confused with message padding.

3.5.6 Cover traffic

Cover traffic involves creating and transmitting dummy messages. The aim is to prevent attackers from distinguishing between fake and real traffic thereby increasing the level of anonymity offered by a system.

The arguments surrounding cover traffic are similar to those for and against message slicing and padding. The additional bandwidth requirements make it impractical for low latency communication mediums. The majority of solutions employing cover traffic techniques are high-latency mix networks [73, 29] (typically used for email). Their Web browsing counterparts [10, 37] on the other hand suggest that this level of resource is not practical or economical and hence argue against it in favour of increased efficiency and usability.

It should be noted that mixes originally aim to protect against a powerful global adversary. In the ideal case, cover traffic achieves unobservability as defined by Pftizmann et. al. [83]. Unobservability guarantees anonymity by preventing outsiders from not even detecting if a communication took place or not. If cover and/or real traffic is communicated continuously, an attacker would not know when a real and when a dummy message was communicated.

It has also been suggested that systems must attract many low-sensitivity users (i.e. users requiring weak anonymity protection) whose network traffic could act as "cover" traffic. The "cover" traffic provided by a large number of low-sensitivity users assists in attracting high-sensitivity users.

Additional effects of dummy traffic algorithms have been theorised elsewhere [30]. Dummy traffic allows mixes to continue to operate effectively even when there is minimal network traffic. When insufficient real messages are available, dummy traffic could be used to fill the messages pools. During high loads, certain solutions [28] throttle dummy traffic to reduce network load. However, traffic policies that are independent from traffic load are considered to be more secure [31], but at the cost of efficiency.

Systems such as JAP [14] originally planned the use of cover traffic. However, the current implementation does not send dummy traffic due to the exorbitant additional bandwidth requirements and because of minimal added benefits.

3.5.7 Cryptographic transformations

It has been noted that traffic analysis poses a greater threat than cryptanalysis [1]. This is understandable as the maturity and strength of current encryption cyphers make cryptanalysis considerably harder than traffic analysis. In addition, encrypted traffic still details who is communicating with whom. Also, a vast amount of Web and email traffic is sent in clear-text.

Section 3.4.1 mentions that anonymity is a subset of security and that anonymity can benefit substantially from existing security techniques. Cryptography addresses confidentiality. Anonymity requires the confidentiality of an identity and hence, a clear relation between cryptography and anonymity

exists. In fact, we know of no implemented anonymity solution that has not made use of cryptography in one way or another.

Many systems [14, 10, 29, 73, 45, 90, 37] implement symmetric and/or asymmetric key cyphers. A more detailed description of cryptography, as used by mixes and Onion Routers, was discussed in sections 2.3 and 2.4 respectively. Also, the importance of achieving forward secrecy through periodic key renewals was noted in section 3.4.3.

A more detailed analysis of cryptography in anonymity systems is not presented – the methods of encrypting data or the communication channel differ little to other computer science fields. It is, however, an important component of anonymity technologies and is therefore listed as an item in the list of connection anonymity techniques.

This concludes the discussion on a connection anonymity strategy. We believe our list of anonymity techniques presents an adequate overview of current solution, but also note that additional techniques might be added in the future. It might even prove worthwhile to explore the use of these techniques in relation to a time-line or with respect to different application domains. This could possibly lead to a more visual representation of the use and appropriateness of these techniques under certain scenarios. This is, however, not central to our research on Polar and instead we conclude the framework with a discussion of two objectives. These feature significantly in existing literature and hence merit being included in the framework.

3.6 Objectives

Whilst anonymity is an obvious objective of anonymity technologies, the measurement of the *degree* or *level* of anonymity is not. In order to justify the usefulness of an anonymous protocol, or perhaps just to compare it to other protocols, a method of quantifying anonymity is needed.

Unfortunately, a trade-off is often observed between the degree of anonymity and efficiency of a system. Whereas absolute anonymity has featured significantly in the past, a further objective has emerged. This objective

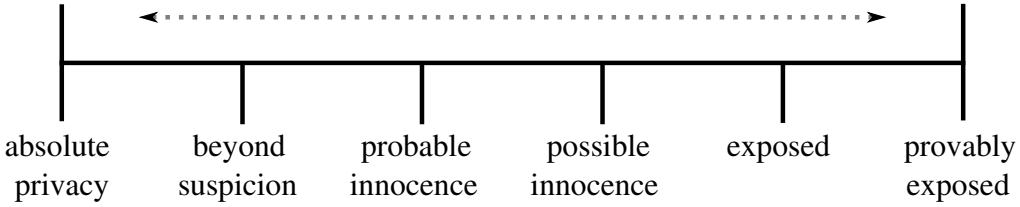


Figure 3.3: Reiter and Rubin's degrees of anonymity

concerns the usability of anonymity technologies. In performance sensitive applications, system efficiency often influences the perceived usability of the system.

An understanding of both the strength of anonymity as well as usability is critical in understanding and balancing both requirements.

3.6.1 Degree of anonymity

Reiter and Rubin [90] present a qualitative scale of anonymity. This scale ranges from *absolute anonymity* on the one side, where one cannot perceive the presence of a communication, to *exposed* on the other end. *Beyond suspicion* is applicable when there is evidence of a communication, but all potential senders are equally likely to have sent the message. *Probable innocence* implies that the probability of having sent a message is less than the probability of not having sent the message; and inversely so for *possible innocence*. Reiter and Rubin's six levels of anonymity are depicted in figure 3.3.

Chaum [21] realises the benefit of anonymity set sizes. The degree of anonymity is directly related to the cardinality of the anonymity set – a system can offer better degrees of anonymity the more users it has. This allows for a quantitative approach and is thus commonly referenced [14, 90, 54]. This approach does, however, not allow for a comparison of different technologies with the same user base. It also assumes a proportional relation between users and the degree of anonymity. More recent research [99, 32] has shown that this assumption is not accurate as not all participants are equally likely to have sent or received a particular message.

Instead, Serjantov and Danezis [99] and Diaz et. al. [32] propose an

information-theoretic metric based on Shannon's [103] concept of entropy. Shannon defines entropy as the level of uncertainty in a system. In anonymity this would pertain to the level of uncertainty surrounding the true sender or receiver of a particular message.

Each participant i from the set of all participants S is assigned a probability $0 \leq p_i \leq 1$ of having sent or received a particular message. This anonymity probability distribution is used in Shannon's discrete probability distribution for entropy. Thus, Serjantov and Danezis [99] give the system entropy $H(S)$ (and thereby the level of anonymity) as:

$$H(S) = -\sum_{i \in S} p_i \log_2 p_i \tag{3.1}$$

If a particular $p_i = 1$, hence all other $p_j = 0$, then H(S) = 0 and so the attacker needs no further information as to identify p_i as the original sender or receiver. If all p_i are equally probable the entropy resolves to a simple anonymity set metric $H(S) = \log_2 |S|$. The aim is thus to increase the entropy of a system – the larger $H(S)$ the more information is required by an attacker to expose anonymous users.

One of the limitations of this metric is that anonymity can only effectively be calculated against a theoretical system model [31]. The probability p_i assigned to each participants often depends on variable factors such as network traffic load, location diversity of the participants and the expected attack model (such as the attacker capabilities).

Nevertheless, the metric is a valid contribution towards quantifying anonymity properties of a system.

Anonymity technologies, and in particular mix solutions, traditionally placed a strong emphasis on achieving absolute anonymity. Absolute anonymity is sought at whatever cost. This was possible for anonymous email solutions where high delays and resource costs are less of an issue. This requirement has changed more recently as anonymity technologies have to provide for anonymous interactive and real-time communications. Due to resource restrictions, the degree of anonymity offered by high-latency mixes is not feasible in low latency, high volume anonymous communications. In-

stead, efficiency or rather usability (being a much broader term) has emerged as an important objective of anonymity technologies.

3.6.2 Usability

An interesting analogy on the progression and trends of anonymising technologies is made by Goldberg and is published in two papers in 1997 [50] and in 2002 [51]. The latter paper discusses the first in retrospect; it explores how anonymity was addressed in 1997 and how this has changed over a period of five years.

Goldberg notes how most commercial anonymising technologies have failed, with Anonymizer being a notable exception. Systems such as SafeWeb and the Freedom Network [10] could not attract enough paying customers to support the overhead costs of running a high-quality network. In the end, it was the large infrastructure requirement that was to be their downfall.

A solution to this came in the form of a distributed architecture that allowed independent entities to join a network and operate nodes as service providers [37]. Crowds [91] went even further and required *all* anonymity seekers to additionally be anonymity providers.

Whilst this solved the costing issue, the network resource problem remained. Anonymous email solutions such as Mixmaster [73] and Mixminion [29] have expensive network bandwidth requirements but are not as severely affected by efficiency constraints such as their low-latency Web browsing counterparts. Thus, a new set of challenges is introduced for low-latency anonymity systems.

Unlike cryptography, anonymity cannot be obtained by one or two users alone. Anonymity systems are only effective once a large number of users have bought into the system. High-sensitivity users require a large user base in order to obtain a high degree of anonymity. High-sensitivity users therefore need many participants to increase the user base and thereby the anonymity set. Suppose many users find a strong, high-latency system inconvenient. Suppose it is so few that the anonymity set is too small for the high-sensitivity users; this would result in the system being is too inconvenient for low-

sensitivity users and too insecure for high-sensitivity users. As Dingledine puts it: *"systems need users for privacy, but need privacy for users"* [35].

Therefore, low-latency anonymity technologies have to relax their threat models so as to increase efficiency, thereby improving usability and attracting a larger user base. After all, anonymity is achieved by hiding a user amongst *many* other users.

This approach is followed by some (more recent) anonymity technologies including Tor [37] and Polar [111]. Tor's strategy is to remain useful enough to attract many users although it has a weaker threat model than some other solutions. In addition, user adoption and usability are given as the most crucial challenges facing Tor [35] and not the degree of anonymity. Back [11] supports this viewpoint by noting that in many cases the efficiency and perceived benefit of a system is more important to an end user than its anonymity properties.

As a final note, that our usage of the term "usability" is a rather broad one. What we consider to be "usable" (or rather "user-friendly") includes efficiency, configurability and deployment issues. Usability describes a user's perceived benefits of the actual use or operation of the anonymity technology.

Our paper [112], that initially detailed the connection anonymity framework, introduced the term *continual anonymity*. Continual anonymity places increased emphasis on usability with an acceptable (as opposed to absolute) level of anonymity. What constitutes acceptable depends on the user, the application domain and the intended uses.

An important notion of continual anonymity is that systems should make it difficult (or infeasible) for an attacker to repeatedly expose anonymous users. For example, Polar aims to offer an acceptable level of anonymity for casual or day-to-day Web browsing. A user might not have highly sensitive personal information that he or she wishes to protect but might want to prevent anybody from profiling them. Thus, the exposure of a single Web request is not as detrimental as the exposure of a collection of requests.

Any solution that is inconvenient, inefficient and time consuming will not gain widespread acceptance for use in daily browsing activities. This means that solutions that have considerable overheads or impose detracting delays

will only cater for a select few users (i.e. those users who require strong anonymity for use in very specific tasks). These solutions will probably not be used by individuals who require weak protection. This will inevitably result in users refraining from using the anonymity service. A need exists to offer anonymity on an ongoing (or continual) basis.

Usability is therefore noted as an important objective in addition to the degree of anonymity and is listed as item VI in figure 3.1.

3.7 Related work

A number of taxonomies exist that list and compare individual anonymising protocols [80, 50, 51, 31, 116]. The fact that few taxonomies actually group anonymising protocols according to their functionality or capability leads us to believe that connection anonymity, as a research field, has not matured fully.

In fact, we know of no other attempts at defining an anonymity framework. A meta-level approach to connection anonymity is not often attempted.

Some related work is presented by Wright et. al. [114, 115, 116] in a number of published and unpublished papers. A characterisation of anonymous transactions that bears some relation to the framework is presented in one of their papers [115].

Both our attempt and that of Wright et. al. [115] share a common goal of providing a more formal view of anonymity but differ in the approach taken. Their work could possibly complement sections 3.4.1 and 3.4.3. Some of their research bears similarities to our analysis of connection anonymity functions and can be found in an overview paper [116]. Their thesis [114] examines anonymity with particular reference to the formal specification of anonymity.

3.8 Conclusion

In this chapter a meta-level approach to connection anonymity was presented. Issues relating to the problem statement, methodology and objectives of connection anonymity were discussed. These issues were split into design factors, connection anonymity strategies and objectives.

The framework aims to provide a more structured and formal view of connection anonymity. A classification of important anonymity concepts was offered. Two important objectives of anonymity concluded the discussion.

The greatest benefit to Polar of using such a framework should become evident in the following chapter where an in-depth analysis of Polar in relation to the framework is made.

Chapter 4

Introducing Polar

4.1 Introduction

Preceding chapters provide the requisite background of anonymising technologies. The literature review, as well as research done on the anonymity framework, assisted in identifying a number of problems with current anonymising technologies.

This dissertation aims to address anonymity issues at three levels. First and foremost the given problem statement, which was was introduced in chapter 1, is addressed. Secondly, it wishes to identify a number of higher-level objectives and address these as best it can. As a result, Polar offer numerous technical improvements.

This chapter considers the identified problem statement and derives five high-level objectives. The rest of this dissertation presents our solution to these objectives and furthermore offers a number of technical benefits.

In addition, this chapter applies the framework to Polar thereby offering an initial feasibility study and giving a more detailed insight into how Polar aims to achieve its objectives.

At the end of this chapter the reader should have a good understanding of Polar's objectives and how this dissertation will proceed in addressing these. Subsequent chapters detail and analyse the model.

4.2 Problems identified

Our research identified the following problems with the current state of connection anonymity:

1. **Low-latency** communications have received some attention; however, these solutions are not as mature as their high-latency counterparts.

2. **Decentralised** architectures are well-suited to anonymity technologies. Centralised solutions present a single point of failure and require full trust in the operator of the system.

3. **High costs** of providing an adequate anonymity service has been the downfall of many commercial anonymity ventures.

4. **Large user bases** are desirable. Unlike, cryptography, anonymity cannot be attained by the sender and receiver alone. An individual hides amongst many other individuals in order to become anonymous.

5. **Usability** of an anonymity technology has been noted as an important factor contributing to the success or failure of an anonymity technology. Usability describes a user's perceived benefits of a system compared to the hassles of operating the system. A user's perceived benefit depends largely on the anonymity requirements of a user; high-sensitivity users require stronger protection compared to low-sensitivity users. Operating hassles include deployment and configuration complications but most significantly comprise performance deficiencies (particularly for for low-latency communications). Poor system usability would deter low-sensitivity users first. Thus, usability affects the user base and indirectly also the degree of anonymity as the user base increases. Whereas in the past the emphasis has been on absolute anonymity for high-sensitivity users, little has been done to promote continual use of anonymous technologies amongst low-sensitivity users.

Of the technologies covered in the literature review, only a few are still operational. A brief overview is presented before proceeding to Polar's requirements and objectives.

4.3 Currently active projects

Anonymizer (`http://www.anonymizer.com`), and Anonymizer clones, are still operational although they protect against a weak threat model. The individual has to trust the operator's integrity and technical prowess in safeguarding his or her browsing history.

JAP [14] (`http://anon.inf.tu-dresden.de/`), a mix cascade, is an active project. It requires each mix operator to officially declare that logs are not kept and are not exchanged with other operators. It only uses limited mixing techniques.

Gathered from the amount of media attention, Tor [37] (`http://tor.eff.org/`) seems to be one of the more popular anonymous Web browsing solutions. It is a more recent implementation of the original Onion Routing [54]. An attractive feature of Tor is the fact that it aims to promote anonymity by improving usability [35]. Tor has a client/server architecture, but allows anybody to join the network as an Onion Router. Thus, to some extent, it allows anonymity seekers to also become anonymity providers.

By comparing the list mentioned here with the list of technologies discussed in the literature review, it becomes apparent that only a few are still actively used. We believe there is merit in further researching anonymity technologies, thus hopefully, further promoting their use. Polar should contribute thereto.

4.4 Polar's requirements

In the introductory chapter in section 1.3 the purpose of this dissertation was specified: to propose an anonymous Web browsing protocol which is (1) fully distributed, (2) offers adequate levels of anonymity and (3) enables users to browse the Internet anonymously without overly complex mixing techniques.

The requirements that the protocol be fully distributed and also minimise mixing techniques, are challenges.

Early research on peer-to-peer technologies identified structured peer-to-peer overlays as a possible solution to the requirement that the technology

be fully distributed. This dissertation hence also explores the viability of a structured peer-to-peer overlay for purposes of routing in an anonymity protocol.

Different types of peer-to-peer overlays are covered in the following chapter. For now it is assumed the reader has a basic understanding of peer-to-peer networks and can differentiate between these and client/server systems.

4.5 Polar's objectives

The advantages and disadvantages of using a peer-to-peer network over a client/server network are many. Pre-existing trust relationships and security requirements differ significantly. This becomes particularly interesting when anonymity is given as an additional objective.

Crowds was identified as an existing anonymity technology that already uses a peer-to-peer network. However, Crowds is not fully distributed and employs an inefficient routing algorithm. Our model titled Polar will attempt to improve on the original Crowds design whilst attempting a fully distributed architecture.

With specific reference to the list in section 4.2, we discuss how Polar's fully distributed peer-to-peer network will attempt to address each problem.

1. **Low-latency:** Polar will attempt anonymous Web browsing. Web browsing is typical of an interactive low-latency communication medium.

2. **Decentralised:** Polar will attempt to be fully distributed; this is a requirement of Polar. We believe that the use of a structured peer-to-peer overlay will enable Polar to achieve this. Polar should prevent some of the problems inherent to centralised or partially centralised system.

3. **Costs:** The Polar network aims to be a free service maintained and operated by volunteers. The peer-to-peer architecture should require all

volunteers to also be participants. This alleviates any logistical problems associated with obtaining and maintaining dedicated hardware.

4. **Large user base:** the fact that Polar is a free service should promote the rapid growth of the user base. This is, however, difficult to measure before-hand as Polar's success ultimately depends on a number of factors including the degree of anonymity and usability.

5. **Usability:** the Polar model should consider the impact that anonymity-enhancing measures could have on the usability of the system. Polar should try to cater for high-sensitivity users but not at the expense of low-sensitivity users. Polar should specifically aim to promote anonymous Web browsing amongst low-sensitivity users. This will increase the user base thus providing improved protection for high-sensitivity users. It should be noted that efficiency and anonymity present a trade-off. Improving both or finding a suitable balance, is a difficult task.

We believe that a peer-to-peer architecture is a suitable choice, able to assist in achieving these objectives. The rest of this dissertation aims to explore this statement by detailing and evaluating the Polar model.

First the connection anonymity framework is applied to Polar. This provides an initial feasibility study of Polar and simultaneously offers a preview of succeeding chapters.

4.6 Applying the framework to Polar

The framework should assist in identifying relevant concerns of an anonymising technology. This should allow for higher-level planning and analysis before delving into the technical details.

The framework is thus applied to Polar. Items in the framework, depicted in figure 3.1, are discussed next in a top to bottom order.

4.6.1 Anonymity versus unlinkability

Polar prevents linking a Web request to the original sender. It thus offers sender unlinkability. No attempt is made at anonymising the Web server.

Furthermore, although a Web server can trace a request back to a proxy within the Polar network, it cannot make a claim about the source of the request. All nodes participating in Polar are potential proxies in an anonymous transaction. Thus, every Polar node can disassociate itself from a Web request as it could claim that it only forwarded the request and did not issue it. Reiter and Rubin [90] refer to this property as *plausible deniability*.

4.6.2 Application domain

Polar offers anonymous Web browsing. Web browsing is an interactive pull medium: a Web request is followed by a Web response.

The latency requirements of Web browsing are considerably stricter than for email. JAP [14] and Tor [37], both of which offer anonymous Web browsing, even refer to their solutions as providing a real-time communication. Whilst we do not fully agree with this, we do acknowledge that Polar's communication overhead should be minimal so as to provide the user with an "acceptable" browsing experience. What users consider to be acceptable and what not ultimately affect the usability properties of Polar.

Unfortunately, these performance metrics can only be measured effectively in a live system. Polar's peer-to-peer architecture means that the anonymity service is offered by an uncontrolled environment. There are many unknowns that might affect the performance of the overall system. Such unknowns include the number of users, the number of requests, the average volume of requests and the Internet connection speeds of participants.

These issues apply equally well to Crowds. Polar attempts to improve on the Crowds design by improving overall routing efficiency. This is achieved through the use of a structured routing overlay which offers some advantages, namely bounded number of routing hops as well as topology-aware routing.

4.6.3 Threat model

An advantage of Polar is its highly distributive architecture. Large-scale distributed systems can operate in multiple administrative domains making compulsion attacks more difficult [88]. Systems that span multiple networks, organisations, service providers and countries require an attacker to have increasing technical authority over multiple networks. Thus, Polar's "location diversity" makes end-to-end traffic analysis attacks increasingly difficult.

Polar specifically aims to prevent identity exposure through traffic analysis performed over a single node. Polar does take a number of measures to prevent this. More notably, Polar introduces the concept of a tail path as well as a circular communication channel. A clear distinction is also made between forwarded Web content and Polar control messages. These are introduced in the following chapter.

A disadvantage of Polar is the fact that it is relatively easy to become an active, internal attacker. Anybody can join the Polar network including malicious users. This is an inherent disadvantage of a peer-to-peer architectures compared to client/server models. This additionally opens the possibility of colluding nodes. Powerful adversaries with such capabilities do indeed threaten the anonymity efforts of Polar.

Whilst this is not a strength of the architecture, Polar does try to minimise the possibility and severity of such an attack. This is achieved by incorporating some concepts related to Onion Routing.

Lastly, Polar achieves forward secrecy because each Web request is treated individually. A previous Web request has no bearing on current or future Web requests; this excludes any inference attacks as discussed by Brandi and Oliver [17]. Such "behavioural" attacks are much harder to protect against. Again, this is not the focus of this dissertation.

4.6.4 Anonymity strategy

Polar is a peer-to-peer network and hence employs a peer-to-peer routing protocol. More specifically, Polar operates on a structured peer-to-peer overlay called Pastry [97]. Pastry path lengths are variable but also bounded by

a maximum value. This results in a hybrid situation of variable and fixed path lengths.

Similarly to Tor, Polar employs minimal obfuscation techniques. No overly-obtrusive attempt is made at mixing Web request or responses. Message delay and release strategies are not employed for the communication channel.

The arguments for not using these techniques are similar to those offered by the designers of Tor [37, 35]: the costs of employing such obfuscation techniques outweigh their benefits. Either significant mixing techniques would have to be implemented so as to adequately increase the degree of anonymity, or none at all. A half-measure renders the system too vulnerable to traffic analysis attacks [35], and a fully fledged mixing strategy, as employed by Mixmaster [73] and Mixminion [29], imposes lengthy message delays.

Furthermore, Polar achieves anonymity through plausible deniability. It is therefore no secret when a node issues a request, as long as it is not known whether *it* issued the request.

4.7 Conclusion

This chapter introduced Polar and specifically covered Polar's objectives. The connection anonymity framework was applied to Polar, thereby providing the reader with an initial overview and analysis of our solution. Some advantages and disadvantages of Polar where already identified. The rest of this dissertation provides a more detailed discussion.

First, a number of structured routing overlays are considered. The most suitable overlay is subsequently chosen for use in Polar.

Chapter 5

Peer-to-peer Overlays

5.1 Introduction

A peer-to-peer network has numerous advantages and disadvantages compared to more traditional client-server architectures. The trade-off between these advantages and disadvantages is particularly relevant to privacy applications. Specifically, the notion of *"ownership of a system"* and the respective trust implications differ considerably between the two architectures.

A peer-to-peer routing algorithm seems to be an obvious choice for a PET based on Crowds – a crowd consists of a group of peers. However, a very simple routing algorithm with highly variable path lengths is employed in the original design of Crowds. Also, a central server for node discovery and key distribution is required. Crowds is a network of peers but still suffers from many of the trust issues typically related to client/server architectures. Polar is an attempt to improve on this by employing a peer-to-peer architecture that is fully distributed.

This chapter discusses peer-to-peer routing algorithms and their relevance to Polar. More specifically, structured routing overlays are discussed. An overview of some of the more well-known structured routing overlays is given as well as a motivation for why Pastry was chosen for use in Polar.

5.2 Peer-to-peer networks

Shirky [104] defines peer-to-peer as *"a class of applications that take advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet"*. Although the notion of resource sharing is one of the major motivations for peer-to-peer applications, the author believes that the user's level of participation as both user and provider is an important notion that should be included in the definition. We therefore add to this definition by stating that peer-to-peer applications are *distributed systems where the majority of nodes are equal or at least similar in terms of capability and responsibility.*

A distinction is made between systems where the majority of nodes are equal and systems where *all* nodes are equal. Both are commonly referred to as peer-to-peer networks, however, only the latter is a *fully decentralised* network.

Consider Crowds [90] which is referred to as a peer-to-peer network but where centralised servers still coordinate amongst peers. Other examples include some well-known file-sharing applications such as Napster and Kazaa [113]. Although Napster uses peer-to-peer communication for file transfer, locating a file is still centralised. Thus, core functionality of the network is provided by a server and not by peers. In Kazaa's case centralised servers are used for a subset of non-core tasks such as bootstrapping and reputation management.

Theotokis et. al. [6] classify Napster's architecture as *hybrid decentralised* and Kazaa as *partially centralised*. Invariably, Crowds would also not be classified as *fully decentralised* because centralised servers are needed for node discovery and key distribution.

This chapter only discusses *fully decentralised* peer-to-peer routing architectures.

5.3 Unstructured and structured protocols

Whilst the level of decentralisation given by Theotokis et. al. [6] refers to the capabilities of peers, a further distinction is made between different types of peer-to-peer routing protocols. Protocols can be unstructured or structured [113].

Gnutella is one of the more well-known peer-to-peer applications with an unstructured routing overlay [113]. Queries are performed by broadcasting to neighbouring nodes. This lookup mechanism generates large loads and is not very scalable [113, 6].

Wang and Li [113] classify unstructured peer-to-peer routing protocols such as Napster, Kazaa and Gnutella as first generation peer-to-peer architectures. Structured peer-to-peer routing overlays are considered to be second generation architectures.

Theotokis et. al. [6] give a definition of peer-to-peer networks which seems more suitable for structured peer-to-peer networks, *"peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organise into network topologies with the purpose of sharing resources such as content CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralised server or authority"* [6].

The principal keyword in this definition is the word "topology". Unstructured protocols generally display little structure and rely on randomised routing or broadcasts. Structured protocols, on the other hand, function according to a well-defined and structured topology.

The routing employed by Crowds is based on a random decision. Crowds' routing protocol is thus unstructured. Polar explores the viability of achieving anonymity on a fully decentralised network that utilises a structured routing protocol.

The motivation for Polar's use of a structured peer-to-peer routing protocol can be summarised as follows: structured protocols offer the ability to efficiently function, scale and self-organise in the presence of a highly tran-

sient population without the need of a central server.

5.4 Structured overlays

A structured overlay is a routing protocol amongst peers that operates on top of, and independently from, the underlying physical network. An overlay exposes its routing functionality to one or more applications running on top. The peer-to-peer network information of an overlay is soft state thus allowing for efficient overlay construction and repair.

Nodes and queries are associated with an identifier and a key respectively. Both identifier and key are unrelated to the underlying physical network. Instead they map onto a nodeid space which is also referred to as the logical peer-to-peer topology. Common topologies include mesh, ring and d-dimension torus topologies [6], all of which facilitate distributed hash-table techniques.

Queries are generally messages addressed to a key. The content of the messages typically depend on the applications running on top of the routing overlay. Given a key, an overlay is used to route the message to a node responsible for the key. This overlay should be deterministic, scalable and arrive at consistent results from any point in the network.

Five structured peer-to-peer overlays are considered. Chord [107], CAN [87], Pastry [96], Plaxton [85] and Tapestry [119] are discussed next in that order. A general understanding of the differences between each of the overlays is beneficial but not essential in understanding the Polar model. The reader should note some advantages and disadvantages of the discussed overlays. This dissertation only requires a more thorough understanding of Pastry.

5.4.1 Chord

Chord provides a *distributed lookup* algorithm that, given a key, yields the IP address of a node responsible for the key. Chord [107] is a variant of the consistent hashing lookup routine [62]. The main difference between the two is the following: 1) the routing table in Chord is distributed (i.e. each node

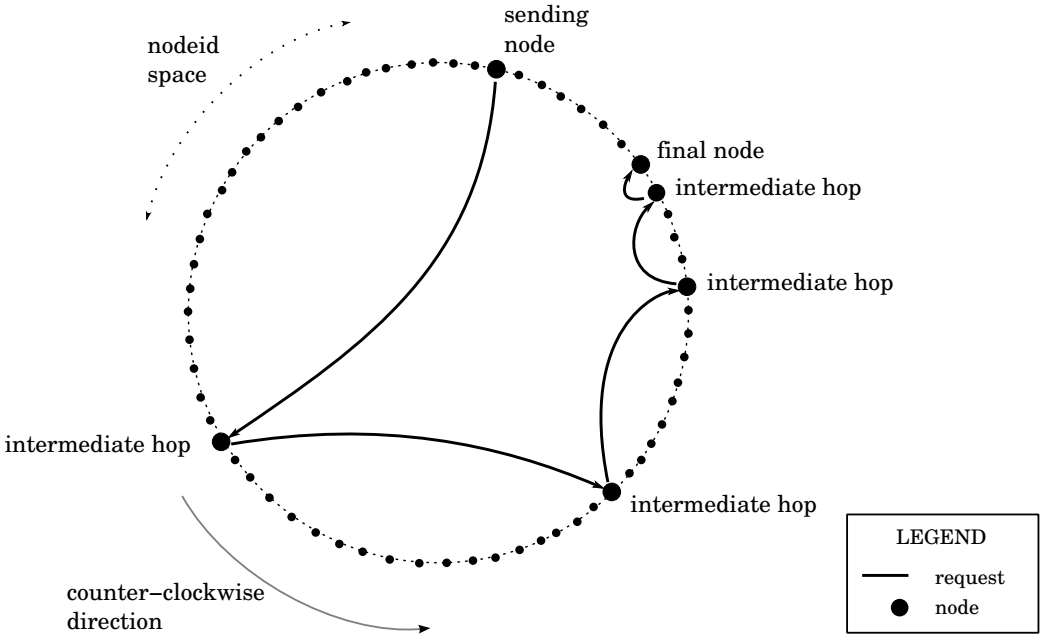


Figure 5.1: Message routing in Chord

only keeps track of a portion of the whole network) and 2) Chord is able to perform in a highly transient node population.

Chord's nodeid space is a circular ring of size 2^m where m is the identifier length (in number of bits). Each node maintains a routing table with at most m entries. Each entry i contains the IP address, port and the Chord identifier of a node that succeeds the current node by at least 2^{i-1}. Thus, successor s_i for node n is thereby given by $s_i = successor(n + 2^{i-1})$ where $1 \leq i \leq m$.

Given a key, each Chord node queries its routing table to find that node which has a nodeid closest to the key. This node is then notified of the request for the corresponding key. This routine is performed iteratively until the absolute closest node is reached. The request only travels in one direction (clockwise) around Chord's circular nodeid space. Chord's circular nodeid space is illustrated in figure 5.1. An example of how a request is routed from a sending node to the respective responsible node, is also illustrated.

Some of Chord's routing statistics are as follows: for network size N,

given by $N \leq 2^m$, one can deduce that each node has a routing table size of $O(\log N)$. Similarly, queries are executed in $O(\log N)$ messages with an average lookup of $\frac{1}{2}\log N$ [107].

When the Nth node joins the Chord ring it assumes responsibility for $\frac{1}{N}$th of all keys. Each node also keeps track of its predecessor. When a new node joins the network, its immediate predecessor is notified. This is repeated for all subsequent predecessors thereby allowing routing state changes to be updated iteratively in a counterclockwise direction.

Chord aims to achieve the following:

- **Load Balancing:** the hash function (SHA-1 [44, 39]) distributes the load with high probability
- **Decentralisation:** through the use of a distributed hash function
- **Scalability:** routing table size and lookup cost grow logarithmically
- **Availability:** supports concurrent node joins and failures

As will be seen next, these objectives differ little to those of other structured peer-to-peer routing algorithms. However, Chord claims simplicity, provable correctness and performance as its advantages [107].

5.4.2 CAN

The Content Addressable Network (CAN) [87] is another protocol that adds fault-tolerance and self-organising capabilities to an existing hash table technique.

CAN is a d-dimensional Cartesian coordinate space on a d-torus (i.e the coordinate space wraps). The space is divided into d-dimensional blocks, also known as zones. Each node is assigned a particular zone and is responsible for all keys that are located in its zone. Nodes keep track of their neighbours in the coordinate space. Requests for a particular key are performed by routing through intermediate CAN nodes whose zones cover the direct path (in the coordinate space) from the requesting node to the CAN node whose zone

contains the key. Path length is given by $\frac{d}{N}^{\frac{1}{d}}$ for perfectly partitioned coordinate spaces where N is the number of nodes in the network [87]. Average routing path length is $\frac{d}{4}N^{\frac{1}{d}}$. In order to get the same logarithmic lookup performance as Chord d has to be chosen such that $d \leq \frac{\log 2N}{2}$.

The advantage with CAN is the fact that node additions are relatively simple; an existing node splits its zone and shares one half with the joining node. Only $O(2d)$ other nodes (all neighbours) have to be notified. Thus, CAN suits highly dynamic environments such as sensor networks.

A disadvantage is the node deletion procedure. Node deletion results in another node assuming responsibility for the orphaned zone. Two or more zones can be held by one node, thus affecting the load distribution of the network. A background zone-re-assignment process is required to periodically reallocate multiple zones.

5.4.3 Pastry

Pastry [96, 97] is another wide-area peer-to-peer routing protocol. It offers application level routing, decentralised control, self-organisation, fault repair and efficient routing for a scalable and dynamic network.

Because of its relevance to Polar, a more detailed description of Pastry is presented.

Introduction

Pastry as well as Plaxton and Tapestry (discussed next) are prefix-based location and routing protocols. They bear some resemblance to Classless Inter-Domain Routing [47] – addresses are evaluated from left to right (or right to left) in order to find the largest common prefix (or suffix).

Whereas IP addresses are organised into a hierarchical structure, addresses in Pastry indicate a position within a circular node identifier space. Nodeids are 128-bit incremental identifiers of base b, where b is some global parameter. Nodeids wrap from the maximum value $2^{128} - 1$ back to 0 thus completing the circle. Each node is assigned a single nodeid.

Each message is routed towards a key. Keys map onto the circular nodeid space. Given a key, Pastry routes the message to the node whose identifier is numerically closest to the key. Depending on the intended application of Pastry, a respective response can then be sent to the original requester (indicated by the dashed line).

Pastry nodes maintain partial routing state tables. This often requires multiple intermediate hops before the final node is reached. Pastry's routing algorithm converges; Pastry will always route a key to a node whose nodeid is closer to key the than the current node's nodeid. This allows Pastry to have a bounded number of routing hops and guarantees message delivery.

The discussion on Pastry routing is split into three parts. First the basic routing is presented, followed by a discussion on route locality and topology-aware routing and finally, an analysis of Pastry's self-organising and fault-repair capabilities is presented.

Routing

Routing tables consists of $\log_{2^b} N$ rows with $2^b - 1$ entries each. A typical value for the global constant b is 4. The total number of nodes in the network is given by N. The nth row contains pointers to nodes that share a common prefix of length n. Each entry in that row has one of the $b^2 - 1$ other values for the $(n + 1)$th digit. A node forwards a request to an entry in its routing table that shares the longest matching prefix with the given key. Similar lookup and forwarding is performed by the next and all following nodes until the overall numerically closest node is reached.

To illustrate, a routing table for a 24-bit nodeid space of base 4 is considered. An example of such a routing table is given in table 5.1 for a node with a nodeid of 311213010232.

Note that in table 5.1 not all routing table entries are populated. Appropriate nodes might not exist or might not yet have been discovered.

Note that the illustrated routing table has $\log_{2^b} N = \log_{2^2} 4^{12} = 12$ rows and $2^b - 1 = 3$ columns. Although four columns are shown, each row has an entry for the local node's identifier thus only three entries per row need to

	0	1	2	3
1	–030102302100	–132121301021	–232021301021	*311213010232*
2	3–02020301010	*311213010232*	3–20231130102	3–31313003102
3	31–0320102302	*311213010232*	31–2101102013	31–3222312010
4	311–023111022	311–120100203	*311213010232*	311–
5	3112–01220110	*311213010232*	3112–20123130	3112–31323130
6	31121–0231010	31121–1133233	31121–2312001	*311213010232*
7	*311213010232*	311213–130111	311213–212311	311213–331021
8	3112130–	*311213010232*	3112130–21020	3112130–3023
9	*311213010232*	31121301–3201	31121301–2012	31121301–3121
10	311213010–032	311213010–	*311213010232*	311213010–330
11	3112130102–01	3112130102–13	3112130102–	*311213010232*
12	31121301023–0	31121301023–1	*311213010232*	31121301023–3

Table 5.1: Example of a Pastry routing table

	Key		Next hop	Prefix length	Table entry
1	112032132121	→	–132121301021	1	$T_{1,1}$
2	310230321120	→	31–3222312010	3	$T_{3,3}$
3	311213203122	→	311213–212311	7	$T_{7,2}$
4	311213010231	→	31121301023–1	12	$T_{12,1}$
5	311213010232	→	*311213010232*	12	–

Table 5.2: Routing table lookup example for table 5.1

be stored.

Pastry will always route a key to an entry in the routing table with the longest matching prefix. Some routing examples are given in table 5.2. By analysing the examples given it should be clear how Pastry's node lookup and routing is performed. Note that examples one to three require at least one intermediate hop.

Figure 5.2 illustrates a typical path that a request takes through the network. The figure shows how three intermediate hops are required before reaching the final node. It also illustrates how the algorithm converges – each successive intermediate hop is numerically closer to the final node.

The figure also indicates a possible return-path for the response. The initiating node could embed its network address in the Pastry message, thus allowing the final node to contact it directly. This return-path is less feasible

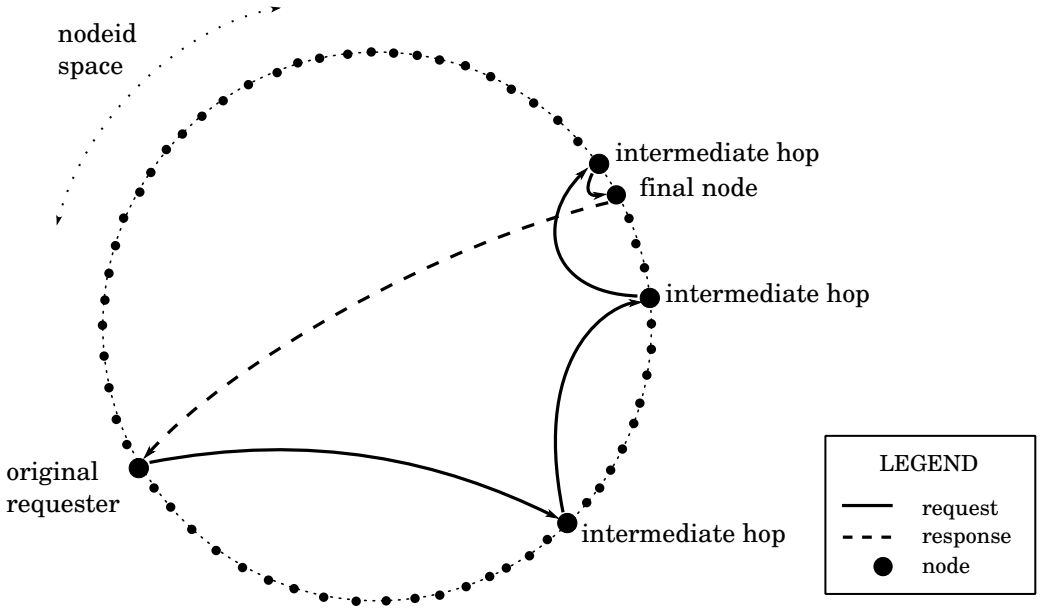


Figure 5.2: Message routing in Pastry

for Polar because this leads to identity exposure of the original requester. Our solution is presented in the next chapter.

To summarise: Pastry's routing algorithm is suitable for large-scale networks, it has a bounded number of routing hops ($\log_{2^b} N$) that increases logarithmically with the size of the network (i.e. its time complexity is given by $O(\log N)$). Also, routing tables are comparatively small with at most $(\log_{2^b} N) \times (2^b - 1)$ entries.

Route locality

Not only does Pastry have a bounded number of routing steps but it also considers network proximity. A scalar proximity metric such as the number of routing hops or the response time is taken into consideration when populating or maintaining its routing tables. A Pastry node uses the metric to compare the newly discovered node with the respective entry in its routing table. The existing entry is replaced if the new node is closer.

To assist in finding nearby nodes, Pastry maintains a neighbourhood set M. This is a list of usually 2×2^b of the closest Pastry nodes according to

the proximity metric. It is not used for routing but only for updating the routing table entries.

Over time Pastry requests maps out routes through the network which minimise the actual distance or time taken for the request. This gives Pastry a significant advantage over Chord or CAN. Route locality is also commonly referred to as topology-aware or proximity-based routing.

Self-organisation, fault tolerance and repair

The procedure of joining the network is similar to other structured overlays. At least one live node has to be known beforehand. This node acts as the bootstrap node. The joining node notifies the bootstrap node that is wishes to join the network. The bootstrap node proceeds to route the message via the network to the joining node. Other nodes along the route can thus learn of the new nodes' presence. All nodes that forward the "join" message share their state tables with the joining node, thus allowing the joining node to initialise its tables with appropriate values. Finally, the leaf set (discussed next) is initialised and all neighbouring nodes (in the nodeid space) are notified of the new node.

Pastry, unlike Freenet (discussed in section 2.5.4), guarantees message delivery even in a highly transient node population. Each node is required to acknowledge receipt of a message. Failure to do so results in the previous node rerouting the message. *Fault repair* is performed lazily (i.e. a failed forwarding attempt results in the respective node pointer being removed from the routing table).

The maintenance protocol minimises failed routing attempts. The *maintenance protocol* runs on each node and ensures that all state tables are up to date. This is achieved though periodic keep-alive packets that verify the liveliness of referenced nodes. Failure to respond to a keep-alive message results in Pastry requesting nearby nodes for a suitable replacement.

Each node maintains a *leaf set* L (with typical size of 16 or 32) which keeps track of the neighbouring nodes in the nodeid space. Responsibility for a key or set of keys can be distributed across the leaf set. This ensures

that object location is guaranteed unless $|L|$ nodes fail simultaneously. The likelihood of all $|L|$ nodes failing simultaneously is highly unlikely. PAST [96], for example, is a peer-to-peer storage facility that uses Pastry and replicates objects across the leaf set.

Conclusion

Pastry is used by other systems, more notably, a group communication and event notification facility called Scribe [18], a decentralised Web cache called Squirrel [61] and as mentioned before PAST [96], an archival and storage facility.

Similarly, Polar intends to use Pastry's node discovery and node maintenance mechanisms. Chapter 6 considers the suitability of a Pastry-like routing and location overlay for an anonymity service.

To summarise, Pastry was chosen because of its capability to self-organise a highly dynamic, fully distributed network. Pastry additionally offers a bounded number of routing hops, utilises proximity-based routing and guarantees object location.

5.4.4 Plaxton and Tapestry

The last structured overlays presented here are Plaxton [85] and Tapestry [119]. Both are prefix-based routing protocols. Tapestry in particular bears significant resemblance to Pastry. Tapestry has its roots in the Plaxton location and routing protocol but adds fault tolerance, fault repair and topology-aware routing. Plaxton, on the other hand, assumes a static data structure and does not support node insertions or deletions.

The main difference between Tapestry and Pastry lies in the way that object location and replication is achieved. When an object is published in Tapestry an object/server mapping is stored at each hop between the publisher and the server. Subsequent requests are forwarded to the server directly if such an object/server mapping is encountered. Replication occurs at these intermediate nodes unlike Pastry where objects are replicated close to the final node.

Polar has no need for replication. Smaller routing tables such as the Pastry routing tables would be more beneficial. There is thus little reason to choose Tapestry above Pastry.

A further consideration swaying our decision to choose Pastry, is due to the fact that an open-source implementation of Pastry exists. The implementation is called FreePastry [94] and is covered briefly in the implementation chapter (chapter 7).

5.5 Conclusion

Polar aims to be fully distributed and hence a structured peer-to-peer overlay was chosen. The overlay should facilitate the organisation of and the routing amongst nodes.

This chapter considered different peer-to-peer routing protocols. The superiority of structured routing overlays over unstructured routing overlays was noted. Four structured peer-to-peer routing protocols were discussed in more depth namely: Chord, CAN, Tapestry and Pastry. Our analysis identified Pastry as a suitable protocol for use in Polar.

Chapter 6

The Polar model

6.1 Introduction

At this point the reader should have a good understanding of the current state of anonymising technologies, the problems Polar wishes to address, and perhaps an idea of how it aims to achieve its objectives. What remains is to give a detailed description and analysis of Polar.

This chapter details the Polar model. Some concepts specific to Polar are introduced. We motivate our design choices and provide an in-depth functional analysis. Measures are presented that aim to improve Polar's level of anonymity. How anonymity is upheld under certain attack scenarios is also discussed.

Chapter 7 delves into more detail and covers our implementation of Polar.

6.2 Objectives

The reader is reminded of Polar's objectives: Polar wishes to be fully distributed, offer adequate levels of anonymity and enable users to browse the Internet anonymously without overly complex mixing techniques.

Polar should function as a proxy between a Web client and a Web server. The Hypertext Transfer Protocol (HTTP), which is most commonly used to request and transfer Web content, should be supported. This includes the

currently active versions of HTTP: version 1.0 [13] and 1.1 [43].

Pastry is the overlay of choice that will enable Polar to operate as a fully distributed system. This should allow Polar to harness the advantages of a peer-to-peer architecture. Polar should, however, also try to minimise disadvantages of such an architecture (compared to client/server models).

6.3 Assumptions

Some assumptions were made during the design of Polar.

For Polar to function efficiently a fully connected network is assumed. This would require all participating nodes and their respective Internet gateways or firewalls to allow both inbound and outbound Polar traffic. Nodes that do not allow inbound traffic can not effectively participate in the Polar network. A request originating from such a node would unconditionally expose its identity: if it can not receive a forwarded request it means any request originating from that node is its own.

Another assumption is that it is difficult, if not impossible, for an adversary to monitor all (or even the majority of nodes) in a Polar network. The likelihood of a global observer is considered negligible. Such an observer would cripple the anonymity efforts of Polar as traffic analysis on all nodes would invariably expose the identity of the original requester. We believe this assumption is not unrealistic on condition that the Polar network spans multiple networks, administrative domains, ISPs and countries. Location diversity is thus required.

This assumption is not unique to Polar. Many anonymising technologies, including Crowds [90] and Tarzan [45], also make this assumption. The reader is referred to section 3.4.3 where this requirement was first discussed.

A foreseeable practical problem of location diversity is the adoption of Polar during its initial phases, when few nodes have joined the network. The size and diversity of the network might not be as high as desired. We hope that the general popularity of peer-to-peer networks amongst Internet users will promote rapid adoption, thus also increasing node diversity.

On an academic level, we first wish to prove the viability of the Polar

model assuming a widely deployed network. The practical concerns of deploying and promoting Polar are only mentioned here briefly and are left as future objectives.

Next, some terminology specific to Polar is introduced. This terminology is used throughout this and the next chapter.

6.4 The Polar architecture

Polar operates a fully distributed peer-to-peer architecture and hence consists of a network of identical nodes. Note that the objective of being fully distributed is one of our design challenges; the use of Pastry for the overlay is a design choice.

The configuration of external and internal participants of Polar relates to that of Crowds. Each node should act as an intermediary between a Web content requester, the Polar network and the Internet (multiple Web servers). A Polar node should never act as a single proxy but should always redirect client requests via other Polar proxies. *Collaboration* amongst nodes is thus required.

Because of their integral role, the following external participants of a Polar communication are considered:

- *clients*
- *Web servers*

Clients request Web content. A client would most commonly be a Web browser, but can be any application that is able to communicate via TCP using the HTTP protocol [13, 43]. No configuration is required by the client or the Web servers, except that the client redirect the Web requests to a Polar node instead of directly to the Web server.

Web servers participate passively; they merely respond to Web requests. Polar nodes are internal participants of the architecture.

6.4.1 Roles

All Polar nodes are identical, and therefore, one can not differentiate between different functional entities. One can, however, differentiate between different roles assumed by nodes during an anonymous communication. Crowds distinguishes between the initiating node, the final node and intermediate proxies [90] (note that Reiter and Rubin [90] use alternate terminology but the meaning does not differ). We adopt these roles and add a further one: the dummy start node. This term is our own and is introduced later in section 6.6.2.

The first three roles are required because Polar, similarly to Crowds, attempts anonymity by plausible deniability. No single node assumes responsibility for a request. Instead, responsibility is deferred to the whole crowd. The initiating nodes accepts requests from the client and initiates the setup of an anonymous communication. The serving node is the last node in the anonymous communication and deals directly with the Web server; it functions as a gateway to the Internet. A chain of intermediate proxies link the initiating node and the serving node and simply relay data back and forth.

The identity of the serving node is no secret as it merely acts on behalf of the Polar network. Web servers should be aware of this. Any attempt at tracking or profiling serving nodes will prove futile. Web behaviour displayed by the serving node is not indicative of that node's true behaviour. This concept of a "delegation of responsibility" is nothing new as it is common to all proxy-related anonymity technologies.

The following roles a Polar node can assume, are suggested:

- *initiating node*
- *serving node*
- *intermediate proxy*
- *dummy start node*

Detailed functions of each role should become apparent as specifics of the Polar model are detailed.

6.4.2 Communication in Polar

Polar's use of a routing overlay already implies a clear separation between routing and other functionality. After all, routing overlays expose routing functionality to other applications. One would thus have to consider how closely integrated Polar's anonymity efforts are with the routing performed by Pastry.

The previous chapter discusses how routing overlays use messages to communicate with other nodes. Similarly for Pastry, control messages co-ordinate routing state amongst nodes while data messages distribute application-data.

Polar aims to anonymise Web content; however, HTTP applications typically treat the transferred content as a stream of data and not as individual messages [13, 43]. It would therefore make sense to distinguish between a communication channel (where content is treated as a stream of data) and Pastry messages.

Channels

TCP/IP (Transmission Control Protocol over the Internet Protocol) is an obvious choice as a transfer medium between nodes; TCP provides stream-based communication and HTTP is generally transferred via TCP/IP.

It would therefore make sense to link the TCP connections in such a manner that a TCP/IP tunnel is established from the initiating node to the serving node. In fact, the tunnel starts at the Web client, traverses all participating nodes and passes through to the Web server. This tunnel is referred to as a channel. The channel is thus a logical term for a chain of TCP/IP *connections* between Polar nodes.

A channel would typically be used to transfer the Web request in one direction and return the Web response in the other. However, a significant contribution of the work presented here is the clear segregation of a Web request and its response. We believe it makes sense to treat the request as a message and the response as a data stream.

Web requests and responses

It was mentioned how HTTP content is often transferred as a stream of data. Whilst this is true, it should also be noted that HTTP [13, 43] is a request/response protocol: a Web response is only issued on receipt of a Web request. The request traverses the channel in one direction and the response is returned in the opposite.

HTTP/1.1 [13, 43] differentiates between eight different types of requests. Particular attention is drawn to the GET, POST and PUT requests. This differentiation is important because it gives an indication of the size of the request. GET requests are most common and typically small in size. Only the URL is passed in the request (besides some possible other header fields). A POST message is used to submit additional HTML form data. One can generally expect POST requests to be somewhat larger than GET requests. Nevertheless, Web requests are generally relatively small compared to the average Web response. This holds true for all Web requests except for the PUT request; the PUT request uploads a potentially large resource.

Although both HTTP RFCs [13, 43] do not enforce any rigid size categories, the intended use of each type implies a certain size expectation. Web responses are expected to be considerably larger than Web requests (with the exception of some PUT requests). This should be obvious when considering that Web servers host *requested* content.

Two reasons necessitate an interest in HTTP request types. Firstly, we wish to prove that a Web request and not necessarily a Web response could expose the identity of a Polar Web surfer. It will be shown how Polar introduces an anonymity-improving technique that prevents the Web response from exposing the identity of the original requester. Secondly, if the first reason holds, then one should attempt to protect the user's Web request.

We believe that the use of message-based communication is useful if the message content is small. Returning large Web responses as messages, instead of via a stream, is less practical. One should ascertain whether the same holds for smaller Web requests. Perhaps one should reconsider the assumption that HTTP communication in Polar should be fully stream-based? Our proposed

solution is presented later in this chapter.

Messages

The Pastry overlay uses messages to transfer small amounts of data from one node to another.

A distinction is made between two types of Pastry messages. The first type, a control message, is used internally by Pastry to communicate the routing state of the peer-to-peer network amongst nodes. Control messages are instrumental during node discovery and node maintenance. The second type of message transfers application data. In Polar's case, the second type of message could assist in establishing a communication channel. They are thus referred to as channel setup messages.

How channel setup messages are used and how they are related to Web requests is discussed in the functional analysis which is presented after the requisite terminology has been introduced. Terminology, relating to the types of communication in Polar, is summarised by the following list:

- *channels* – transfer *Web requests* and *Web responses*
- *control messages*
- *channel setup messages*

Clearly differentiating between the three types of communication will play an increasingly important role as Polar's anonymity capabilities are explored.

6.4.3 Proxy and routing layer

The last Polar-specific terminology is the proxy and the routing layer. A clear separation between routing and channel setup is advisable; a functional routing protocol is required in order to distribute messages and establish an anonymous communication channel. We therefore propose delegating Polar's routing and channel setup functionality into two layers:

- *proxy layer*

- *routing layer*

The proxy layer should operate above the routing layer because it depends on the routing layer's capability of distributing messages. The routing layer, on the other hand, operates independently from the proxy layer. The proxy layer establishes an anonymous communication channel between a client and a server via a number of Polar proxies.

The main functionality of the routing layer is node discovery, route selection and the distribution of application data via messages. Channel setup messages are used to inform other nodes of requests or changes in the proxy layer. Pastry was chosen as the routing protocol and is responsible for servicing the routing layer.

6.5 Functional overview

Now that the requisite terminology has been defined, functional processes of the Polar model can be considered. This is done by analysing the required steps in setting up an anonymous communication channel.

A user wishing to browse anonymously should configure his client to redirect his Web requests to a local Polar node and not directly to the Web server. As far as the user is concerned, Polar should simply be considered as a distributed proxy. Any routing and channel setup details should be hidden from the user's Web client.

Local nodes

Although the initiating node could potentially run on a remote machine, it makes sense to have a local instance. Client requests to a remote node would expose that particular request as the original request. Consider the scenario whereby requests originate from a machine, but the machine does not accept requests itself – either because no Polar node is running on that machine or because inbound traffic is being blocked. This would imply that all requests originating from that machine are original and not redirected requests.

Incidentally, this feature makes leeching unattractive. Leeching refers to the process of deriving benefit from services offered by a peer-to-peer network without having actively participated in the network. Successful participation in the Polar network requires a fully functional local Polar node. We hope this feature will increase Polar's user base as users are required to join in order to benefit from the service offered.

Channel setup

Channel setup is initiated by the initiating node. The following questions remain: how does the initiating node choose an intermediate node, how do intermediate nodes choose each other and how do nodes know when to assign a serving node?

Our initial paper on Polar [111] argues that channel setup messages be routed using the Pastry overlay. Loose routing is performed as each node determines the next routing hop. Emphasis is placed on improving routing efficiency by employing a protocol with a bounded number of routing hops. However, subsequent research identified numerous implications that such a loose routing approach has on anonymity. Re-evaluating the advantages and disadvantages of loose versus source routing was thus required.

Two options were considered. The first is loose routing similar to that of Crowds: forwarding is a random decision based on some global parameter α. Onion Routing [54, 110] presented the second option: the initiating node uses an Onion to establish a bi-directional channel passing through a preselected sequence of nodes.

Tarzan [45] already explores the combination of Onion Routing over a structured peer-to-peer overlay. However, a number of notable differences exist between Tarzan and our proposed solution. Polar's differential treatment of Web requests and responses presents the first deviation. In addition, our revised solution employs a hybrid technique: an Onion is used to transfer the Web request and simultaneously create a uni-directional channel for the Web response. The Polar Onion is detailed later in this chapter.

It is also noted that unlike Onion Routing, Polar aims to hide the initiat-

ing node amongst peers; the original Onion Routing design clearly differentiates between Onion Routing servers and anonymity seekers. In Polar (and Tarzan), these two roles are combined.

The rest of this chapter is organised as follows. Improvements to Polar's original channel setup are first considered. Moreover, the concepts of a symmetric communication and a communication tail are introduced. We briefly explore the problems of Polar's initial loose routing approach and continue to describe the revised channel setup. Finally, Polar is evaluated in light of a number of attacks.

6.6 Improving anonymity

Adversaries can often infer a significant amount of information leading to possible, or even probable, identity exposure. Polar should thus aim to improve the degree of anonymity.

6.6.1 Symmetric communications

Danezis [27] introduces the concept of *entropy* in his design of a mix network. He equates entropy, the measure of disorder of a system, to the achievable degree of anonymity of a system. This makes sense in a mix technology where the aim is to create disorder for the purpose hiding information.

We believe this is an important concept in anonymity research, however, our notion of symmetry also plays an important role. The term symmetry is used informally elsewhere [45]. We are not aware of any research that considers the relation of entropy and symmetry.

Anonymity is achieved by hiding identifying information. We believe this can be accomplished by increasing entropy *or* symmetry. Entropy seeks disorder to hide identifiable information amongst "noise". Symmetry seeks order and equality to hide identifiable information amongst similar information.

Communication symmetry is here defined as the inability to infer identifying information by making the communication patterns of the original requester appear no different to that of other participating nodes. Traffic

patterns include network connections and data transfer.

Crowds, for example, violates the symmetry property as the initiating node displays different traffic patterns to nodes that simply forward the request. Consider an adversary that is capable of monitoring inbound and outbound traffic of a single node. In the original Crowds design, the initiating node only has one outbound connection. All other nodes have two: one inbound connection from the preceding node and one outbound connection to the succeeding node. As illustrated in figure 6.1(a), node A is the only node without an inbound connection. An adversary that observes an outbound Web request without a prior inbound request (as for node A) could safely deduce that this node is the original requester. This is a serious shortcoming of Crowds and henceforth we intend to improve on this.

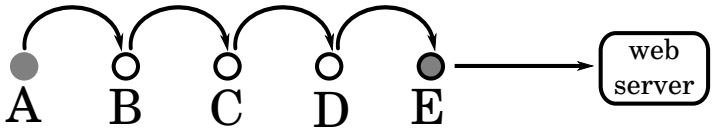
It should be noted that Crowds encrypts the Web request. Therefore, an adversary would still have to determine which request was issued to successfully infringe on an individual's privacy.

However, in Crowds, the Web request is visible to each intermediate hop. In related technologies, such as mCrowds [22], Tarzan [45] and Polar [111], the Web request is only visible to the initiating as well as the final node. Invasion of privacy is thus possible if the final node is corrupt and the initiating node has been identified.

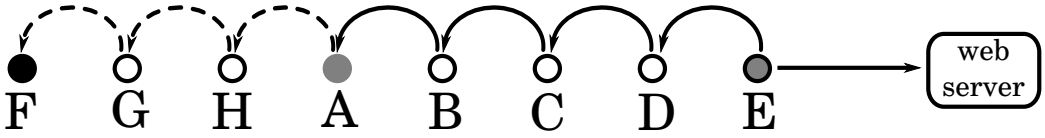
Polar attempts anonymity by plausible deniability. Each node seeks to refute the fact that it issued the request – whether the actual Web request is known or not. Thus, if the initiating node is able to effectively hide amongst its peers, even though the Web request has been compromised, then a better degree of anonymity is offered.

Symmetry is thus an additional mechanism that could assist in offering better levels of anonymity. We believe the notion of symmetry should play an important role in connection anonymity.

Communication symmetry prevents identity exposure through traffic analysis performed on a single machine. We note that symmetry is effective against attackers with a partial view of the system. A global observer could still use timing attacks to expose the original requester amongst a set of seemingly symmetric nodes. We already assume the likelihood of a global



(a) Forwarding as performed by Crowds



(b) Forwarding with a tail

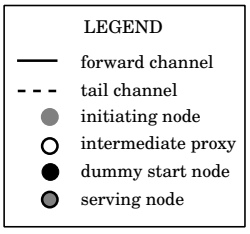


Figure 6.1: Towards symmetric communications

observer to be infeasible. The question thus remains: how can Polar achieve communication symmetry?

6.6.2 Communications tail

Polar attempts communication symmetry by requiring the initiating node to not only construct a channel to the serving node, but also to some previous node. This node is referred to as the *dummy start node*. Polar should not necessarily route directly to the dummy start node but could do so over a number of hops.

The channel to the dummy start node is illustrated in figure 6.1(b) by the dashed line from node A to node F via nodes H and G. This channel is referred to as the *tail* path. The channel from the initiating node to the serving node is referred to as the *forward* path.

The arrows in figure 6.1 indicate the source and destination of a TCP connection (e.g. in figure 6.1(a) node A connects to node B). Note that this differs to figure 6.1(b) where node B establishes the connection back to node A. The choice of connecting to the preceding or succeeding node is trivial as long as communication symmetry is achieved.

Incidentally, arrows in the second figure also indicate the direction of the returned Web response. The serving node submits the request to the Web server and relays the Web response back through the channel to the dummy start node. Any intermediate node could potentially be the initiating node.

One question remains: how does the initiating node communicate the Web request to the serving node? The initiating node can not send the request via the forward *and* the tail path as this violates the symmetry property and reveals the initiating node.

The proposed solution to this problem is two-fold. Firstly, channel setup messages should be readable by the two communicating nodes only. Encryption of messages is thus required. Secondly, the request should be communicated via channel setup messages and not through a channel.

Message-based communication could be used for the request whilst stream-based communication should be used for the response. We wish to show that

if a tail is present then the returned response can not expose the identity of the initiating node. Thus, only the Web request needs to be protected.

The proposed channel setup is detailed first followed by a discussion on how channel setup messages can be obfuscated.

6.6.3 Illustrated by example

To illustrate the proposed channel setup, one could consider the traffic patterns of the nodes from figure 6.1(b). Table 6.1 lists the chronological order of sent channel setup messages (Msg), successful channel connections (Chl), the submitted Web request (Req) and the respective server response (Rsp). The arrows indicate data flow between the source and the destination node.

Note that TCP connections, used during a channel setup, need to be established before data transfer is possible. A TCP connection is established with another machine using a three-way handshake [93]. A machine either initiates or accepts a connection. A connection attempt is observable by any entity capable of monitoring network traffic at or in between either end of the connection.

The setup of the tail and the forward channel can be done in parallel; with the only restriction being that the times $t_1^T, t_2^T, t_3^T \leq t_i^F$ for $1 \leq i \leq 12$ (refer to table 6.1). At time t_1^T a message is sent from the initiating node (node A) to node H, notifying it of a new channel setup. At time t_2^T a TCP connection is made from node A to node H commencing the tail setup. Note that the tail node does not know if this is the initiating node or simply another tail node. The tail setup process is repeated for the tail nodes G and F.

Once the initiating node has established its tail connection (with node H), it pretends it is a node on the forward path and initiates a channel to the serving node. This process is detailed at times t_1^F and t_2^F. Using a channel setup message, node A notifies node B that a forward path should be created. This channel setup message additionally contains the Web request. Node B connects back to node A at time t_2^F. Node B repeats this forward path setup, and subsequently, so do nodes C, D and E.

Each node links the inbound and outbound TCP/IP connection thus

forming a channel from the dummy start node to the serving node. The serving node is the only node that forwards the Web request to the Web server (time t_{11}^F). The response is relayed back to the dummy start node from time t_{12}^F to t_{19}^F.

6.6.4 Analysis

The process of establishing a forward channel is similar to that of the tail channel. The only significant exception is the fact that on the forward channel a connection is established from the notified node back to the requesting node (compare times t_2^T and t_2^F). This allows the initiating node to blend in with other forward nodes bringing us one step closer to communication symmetry.

Note that from an internal observer's perspective: node H can not deduce that node A is the original requester as node A could simply be a another tail node. Similarly, node B does not know if node A is an intermediate proxy or an initiator. Each node only knows the preceding and succeeding node. Nodes H and B know of node A but should not know of each other. In fact, nodes H and B have no means of telling which other nodes, beyond A, are involved. How to best choose the preceding and succeeding nodes is covered in the discussion on location diversity further on in this chapter.

The Web response is passed from the serving node right back to the dummy start node. Any node along this path could potentially be the original requester. Once a channel has been established one can at most determine which nodes are involved, but it is impossible to identify the original requester. A Web response does not expose the original requester.

Identity exposure can thus only occur during a channel setup. This is considered next.

6.6.5 Traffic patterns

Nodes located on the forward path display an outbound connection (to the preceding node) followed by an inbound connection (from the succeeding node). This is reversed for Tail nodes which display an inbound connection

Time	Forward path
t_1^F	**A** $\xrightarrow{Msg}$ **B**
t_2^F	**B** $\xrightarrow{Chl}$ **A**
t_3^F	**B** $\xrightarrow{Msg}$ **C**
t_4^F	**C** $\xrightarrow{Chl}$ **B**
t_5^F	**C** $\xrightarrow{Msg}$ **D**
t_6^F	**D** $\xrightarrow{Chl}$ **C**
t_7^F	**D** $\xrightarrow{Msg}$ **E**
t_8^F	**E** $\xrightarrow{Chl}$ **D**
t_9^F	**E** $\xrightarrow{Chl}$ **D**
t_{10}^F	**E** $\xrightarrow{Chl}$ ws
t_{11}^F	**E** $\xrightarrow{Req}$ ws
t_{12}^F	ws $\xrightarrow{Rsp}$ **E**
t_{13}^F	**E** $\xrightarrow{Rsp}$ **D**
t_{14}^F	**D** $\xrightarrow{Rsp}$ **C**
t_{15}^F	**C** $\xrightarrow{Rsp}$ **B**
t_{16}^F	**B** $\xrightarrow{Rsp}$ **A**
t_{17}^F	**A** $\xrightarrow{Rsp}$ **H**
t_{18}^F	**H** $\xrightarrow{Rsp}$ **G**
t_{19}^F	**G** $\xrightarrow{Rsp}$ **F**

Time	Tail path
t_1^T	**A** $\xrightarrow{Msg}$ **H**
t_2^T	**A** $\xrightarrow{Chl}$ **H**
t_3^T	**H** $\xrightarrow{Msg}$ **G**
t_4^T	**H** $\xrightarrow{Chl}$ **G**
t_5^T	**G** $\xrightarrow{Msg}$ **F**
t_6^T	**G** $\xrightarrow{Chl}$ **F**

Table 6.1: Channel setup for figure 6.1(b)

Abbreviation	Description
Msg	Channel setup message
Chl	Channel setup by TCP/IP
Req	Web request
Rsp	Web response
ws	Web server

Table 6.2: Description for table 6.1

followed by an outbound connection. It seems communication symmetry has not been fully achieved. However, it can be proven that an external observer might at the very most deduce whether a node is located on the forward or tail path but can not deduce whether a node is the initiating node or not; this holds true for any traffic monitoring performed on a single Polar node.

Tables 6.3, 6.4 and 6.5 list the observable traffic of an initiating, forward and tail node. Node labels of the requesting and notified nodes are substituted by r and n respectively to simplify the comparison between tables. Note that for tail nodes the requesting node is closer to the serving node whilst the notified node is closer to the dummy start node. This is the opposite for forward nodes.

By analysing the traffic patterns one can observe how the traffic is not symmetric across all three nodes. Each displays a different sequence of sent messages (Msg) and channel connections (Chl). However, Polar requires the initiating node to blend in with the Crowd. Clearly, this is currently not the case.

6.6.6 Control Messages

Currently, the only difference between the initiating node and a forward node is the fact that the initiating node requires two outbound messages (at time t_1^T and t_1^F). Forward nodes on the other hand have one inbound and one outbound message. This identifies the initiating node and therefore deserves further attention.

Channel setup messages differ little to Pastry's network control messages and hence we propose encrypting all message-based communication in Polar. This would allow channel setup control messages to blend in with the standard join and maintenance messages. An adversary should not be able to identify whether an initiating node sent two channel setup control messages or whether only one or both of these control messages where join or maintenance messages.

The reader is reminded of the discussion on HTTP Web request sizes. Hiding channel setup messages, that contain the Web request, is only feasible

$Time$	Inbound	Outbound
t_1^T		$\mathbf{A} \xrightarrow{Msg} \mathbf{r}$
t_2^T		$\mathbf{A} \xrightarrow{Chl} \mathbf{r}$
t_1^F		$\mathbf{A} \xrightarrow{Msg} \mathbf{n}$
t_2^F	$\mathbf{n} \xrightarrow{Chl} \mathbf{A}$	

Table 6.3: Traffic patterns of initiating node A

$Time$	Inbound	Outbound
t_7^F	$\mathbf{r} \xrightarrow{Msg} \mathbf{D}$	
t_7^F		$\mathbf{D} \xrightarrow{Chl} \mathbf{r}$
t_9^F		$\mathbf{D} \xrightarrow{Msg} \mathbf{n}$
t_{10}^F	$\mathbf{n} \xrightarrow{Chl} \mathbf{D}$	

Table 6.4: Traffic patterns of node G on the forward channel

$Time$	Inbound	Outbound
t_4^T	$\mathbf{r} \xrightarrow{Msg} \mathbf{G}$	
t_5^T	$\mathbf{r} \xrightarrow{Chl} \mathbf{G}$	
t_7^T		$\mathbf{G} \xrightarrow{Msg} \mathbf{n}$
t_8^T		$\mathbf{G} \xrightarrow{Chl} \mathbf{n}$

Table 6.5: Traffic patterns of node D on the tail channel

if the size of the channel setup message differs little to existing Pastry control messages. It is assumed that GET requests, as well as restricted POST requests, are small enough to satisfy this criteria. We believe this assumption is realistic; modifying an implementation of Pastry, to control message sizes in line with expected channel setup message sizes, should not be an obstacle. It is additionally noted that the use of message slicing and padding is also a viable possibility.

One could ask why similar obfuscation techniques could not be applied to the anonymous communication channel. In other words: why not encrypt the inbound and the outbound channel thereby complicating traffic analysis? Would this not enable one to hide the inbound and outbound connection of an initiating node amongst other connections also serviced by the initiating node?

We argue that this solution is less feasible for lengthy data streams than for control messages because of the following reasons:

1. The highly dynamic nature of Pastry's peer-to-peer network means that frequent keep-alive and network maintenance messages are required. It is therefore assumed that control messages are plentiful and are sent out periodically at unpredictable intervals. By making channel setup control messages indistinguishable from other control messages, it can be argued that an adversary's efforts at identifying the initiating node by observing Pastry messages are severely complicated if not infeasible. Essentially, existing control messages achieve the same result as the use of cover traffic (i.e. control messages obfuscate channel setup messages).

2. The number of channels serviced by any Polar node are less predictable. Successfully hiding an initiating node's inbound and outbound connection amongst other connections depends on the number of currently active connections.

3. Control messages lend themselves to fixed message sizing because of their limited size. Data streams on the other hand are less suitable to message slicing and padding. Section 3.5.5 discussed the benefits

Time	Inbound	Outbound
t_2^T		$\mathbf{A} \xrightarrow{Chl} \mathbf{r}$
t_2^F	$\mathbf{n} \xrightarrow{Chl} \mathbf{A}$	

Table 6.6: Amended traffic patterns of initiating node A

Time	Inbound	Outbound
t_7^F		$\mathbf{D} \xrightarrow{Chl} \mathbf{r}$
t_{10}^F	$\mathbf{n} \xrightarrow{Chl} \mathbf{D}$	

Table 6.7: Amended traffic patterns of node G on the forward channel

Time	Inbound	Outbound
t_5^T	$\mathbf{n} \xrightarrow{Chl} \mathbf{G}$	
t_8^T		$\mathbf{G} \xrightarrow{Chl} \mathbf{r}$

Table 6.8: Amended traffic patterns of node D on the tail channel

and drawbacks of message slicing and padding for data streams. It argued that the additional overhead and minimal benefit outweigh the advantages.

Chapter 3 presents a number of mixing strategies. These included cryptography, cover traffic and message slicing and padding. Research on our framework observed how these obfuscation techniques were particularly useful for message-based anonymity technologies, but become less practical as data size and network latency requirements increase. For this reason Polar employs these strategies only for its message-based communications.

Now that it has been argued that channel setup messages could be hidden from an external observer, it is possible to revise the traffic patterns and omit (or rather hide) the channel setup messages of tables 6.3 – 6.4. The revised traffic is listed in tables 6.6, 6.7 and 6.8. Note how the traffic pattern of the initiating node is identical to that of the forward node.

Therefore, from an external observer's perspective, the initiator node blends into the crowd as it appears indifferent (and hence symmetric) to

any other participating forward node. An observer can thus only differentiate between a forward and a tail node but can not identify the initiating node. The tail path allows for the Web response back to (and past) the initiating node without compromising its identity.

By encrypting and sizing control messages, we claim that traffic analysis performed over *a single node* is not sufficient to identify a node as the initiating node. From an external observer's perspective, communications seems symmetric across all participating forward or tail nodes. Polar thus offers a significant advantage over Crowds.

6.7 Routing in Pastry

The focus thus far has been mainly on the channel configuration. Whilst channel setup messages were covered, routing thereof was only briefly mentioned. We alluded to the possible use of Onion Routing. How an initiating node chooses participating nodes and how an Onion could establish an anonymous communication has not yet been detailed.

Our inceptive Polar paper [111] proposed a loose routing approach. Next we argue against such an approach and subsequently introduce a Polar Onion which facilitates source routing.

6.7.1 Convergent and bounded routing

Section 5.4.3 introduces Pastry and discusses the join, routing and maintenance protocols. To summarise briefly: Pastry is a prefix-based routing algorithm. Identifiers indicate a position in a circular nodeid space. The algorithm is guaranteed to converge [96]. A node closest to the key can be found, although possibly via a number of routing hops. Pastry's number of routing hops is bounded to $\lfloor \log_{2^b} N \rfloor$ where N is the number of nodes in the Pastry network and b is a global routing parameter.

From this summary two properties in particular should raise concerns: the fact that routing is convergent and bounded. Both properties affect Polar's anonymity efforts and hence need to be addressed.

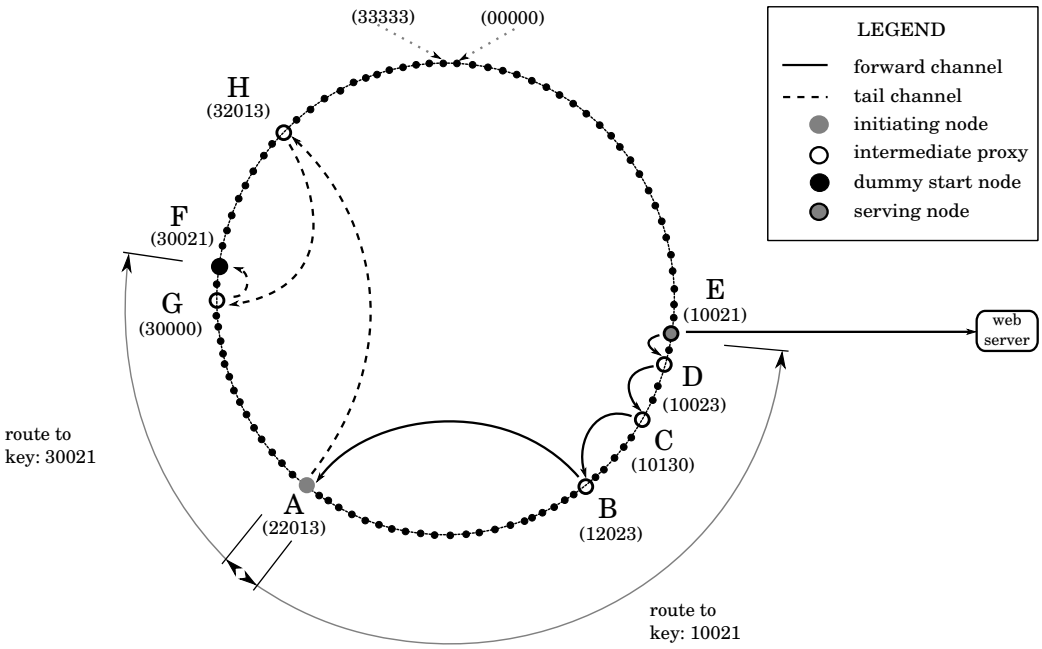


Figure 6.2: Polar's original channel configuration

The problem is briefly discussed. Let us assume that the serving and dummy start node are at opposite ends of the Pastry ring and the initiating node is located somewhere in between. The original Polar routing algorithm would have resulted in a channel configuration similar to the one illustrated in figure 6.2. The figure demonstrates how two channel setup messages are routed to the keys 10021 and 30021. These keys indicate the location of the serving and dummy start node.

Pastry nodes keep partial routing tables and hence routing is generally achieved via intermediate nodes. In the best-case scenario, the specified nodeid is found in the routing table. In this case, the message is delivered to the destined node in only one hop. Consider the scenario where, in figure 6.2, node A can route directly to node E and F. This results in only three nodes maintaining an anonymous communication channel. This property is not ideal as Polar requires a greater number of routing hops in order to increase the anonymity set.

In the worst-case scenario it would take the initiating node A $\lfloor \log_{2^b} N \rfloor$

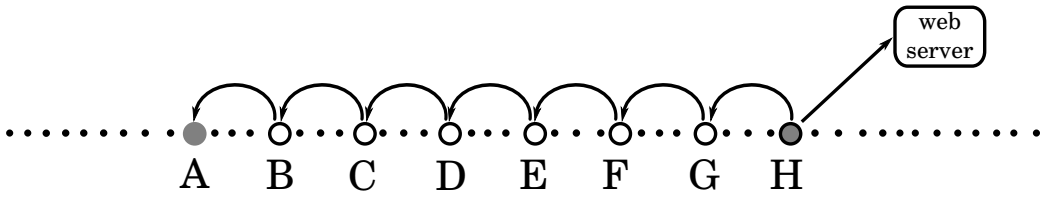


Figure 6.3: Anonymity exposure through convergent routing

hops to route to the specified nodeid.

The fact that Polar's algorithm is bounded also means that Polar's forward and tail path is bounded. Thus, the anonymity set decreases the further an initiating node is from the target node.

Figure 6.3 is used to illustrate the problem. Let us assume that the maximum number of routing hops is seven. It is also assumed that nodes are sorted in alphabetical order meaning node C will always route to A or B and never to nodes $\{D, E, \ldots, Z\}$. A Web request by node A is compromised if:

1. the initiating node requires the maximum number of routing hops to reach the target node (in the given example: seven hops to reach node H)

2. the first hop knows how many nodes precede or succeed it (e.g. if node B knows it is the second node of a sequence of eight)

This statement can be generalised. Let N be the total number of nodes in the network and $S_{(n_T,n_I)} = \{n_1, n_2, \ldots, n_k\}$ be the sorted sequence of nodes located between some initiating node n_I and some target node n_T. If

$$|S_{(n_T,n_I)}| = k \tag{6.1}$$

then the size of the anonymity set $A_{(n_T,n_I)}$ is given by

$$|A_{(n_T,n_I)}| = N - k - 1 \tag{6.2}$$

If node n_1 claims it is not the initiating node, then the anonymity set is the set of all nodes excluding those which lie between node n_1 and node

n_T. Note that this holds if the routing algorithm converges (i.e. the request is always routed towards the target node). Also note that the serving node (node n_T) is excluded from the anonymity set.

The anonymity set decreases as $k \rightarrow N$; the more nodes numerically closer to a target node the smaller the anonymity set.

Let us assume that there are no non-participating nodes in figure 6.3. Therefore $N = 8$ and $k = |B, C, D, E, F, G| = 6$ and $|S_{(H,A)}| = 8 - 6 - 1 = 1$ meaning that anonymity is compromised. Although this is a very crude example for a minimal set of nodes, it does illustrate the problem.

These complications are not unique to Pastry. It can be shown that the case is similar for Chord [107], CAN [87] and Tapestry [119] – these algorithm also converge and are bounded.

6.7.2 Source routing

It was shown how bounded and convergent routing algorithms are unsuitable for attempting anonymity. Still, it would be desirable to harness the efficiency and the self-organising capabilities of a peer-to-peer overlay. It would therefore be advisable to fully separate node maintenance and channel setup.

Loose routing is appropriate for the routing layer; however, source routing should be considered for the channel setup. Source routing gives the initiating node increased control and allows for cryptographic techniques to assist in securing a channel setup.

A well-known source routing protocol for anonymity is Onion Routing [54, 110]. Onion Routing was detailed in the background chapter. It protects against colluding nodes, the channel length is bounded thus reducing latency and it is geared towards stream-based communication.

Onion Routing might at first glance appear to be an appropriate technology for Polar's anonymous Web browsing; however, Polar already differentiates between message-based communication for Web requests and stream-based communication for Web responses. Unlike Onion Routing, Polar has a communication tail and operates amongst peers – it does not differentiate

between Onion Routing servers and anonymity seekers. Polar thus requires a revised Onion Routing algorithm.

We adopt the Onion Routing terminology and propose using a Polar "Onion". The Onion should distribute Web requests securely and should establish a uni-directional stream of data for the Web Response.

6.8 Polar Onions

It is noted that the term "Onion" is generally reserved for Onion Routing [54] and implies a messages wrapped in layers of encryption. Chaum [20] first proposed the concept of a layered message for the purpose of achieving anonymity. Chaum's layered messages and Onions differ in the type of content that is transferred. Onions do not carry the actual secret but rather assist in establishing a channel via a sequence of nodes. The secret is communicated via the channel. The Onion secures the channel by distributing forward and backward symmetric keys amongst successive Onion Routers.

Our proposed solution uses an Onion to transfer the secret (the Web request) and establish a channel for the response. Strictly speaking, it does not comply with the original definition given by Goldschlag et. al. [54]. However, the term "Onion Routing" has been commoditised to the extent that numerous flavours exist [37, 45]; hence the terminology is used here as well. Note that Polar Onions are embedded in channel setup messages, the terms Onions and channel setup messages are thus used interchangeably.

The proposed channel setup, using Onions, is detailed next.

6.8.1 Channel construction

It is assumed each node has a suitable routing table from which it can select participating nodes. This routing table is called the node map and is covered later in this chapter.

For each node n the routing table maps the node's network address A_n and a public key K_n to the respective Pastry nodeid I_n. The routing table entry for node n thus consists of the tuple (A_n, I_n, K_n).

Upon receipt of a client Web request, the initiating node selects a random sequence of nodes $\{n_1, n_2, \ldots, n_\varphi\}$. The sequence contains one dummy start node, one initiating node, one serving node and zero or more intermediate nodes.

The sequence should thus consists of three or more nodes – a greater number increases latency but also lessens the chance of a collusion attack.

The length of the channel is implementation-specific or alternatively can be user-specified according to the desired strength of anonymity versus the expected latency.

The first node of the sequence is the dummy start node whilst the last is the serving node. The initiating node n_I is inserted at any position except first or last. A channel is therefore given by:

$$S_{channel} = \{n_1, n_2, \ldots, n_I, \ldots, n_\varphi\} \tag{6.3}$$

$$= \{n_1, \ldots, n_{I-1}\} \cup \{n_I\} \cup \{n_{I+1}, \ldots, n_\varphi\} \tag{6.4}$$

$$= S_{tail} \cup \{n_I\} \cup S_{forward} \tag{6.5}$$

The initiating node prepares two Onions: one for the tail path and one for the forward path. The routing table is queried for the respective address A_{n_i} and public key K_{n_i} for all $n_i \in S_{tail} \cup S_{forward}$. The Onion is constructed by successively encoding a secret with layers of encryption corresponding to each node in the sequence of nodes. A layer is encoded using K_{n_i} such that only n_i can decrypt it.

The secret, in our case, is the Web request (Req) and a symmetric encryption key K^τ generated by the initiating node. The symmetric key is used by the serving node to encrypt the response. A symmetric cipher should be used because these are generally computationally less expensive than asymmetric ciphers and are well suited for stream-based encryption. Note that Polar treats the response as a data stream. The Onion for the forward path is thus given by:

$$(K_{n_{I+1}}(K_{n_{I+2}}(\ldots(K_{n_\varphi}(Req, K^\tau), A_{n_\varphi}), \ldots), A_{n_{I+2}}), A_{n_{I+1}}) \tag{6.6}$$

The iterative process of constructing an Onion is detailed in steps (6.7), (6.8) and (6.9). Step (6.7) shows an Onion with a single layer. The public key of the last (and serving) node is used to encrypt the Web request (Req) and the symmetric encryption key K^{τ}.

$$K_{n_{\varphi}}(Req, K^{\tau}) \tag{6.7}$$

$$K_{n_{\varphi-1}}(K_{n_{\varphi}}(Req, K^{\tau}), A_{n_{\varphi}}) \tag{6.8}$$

$$\vdots$$

$$(K_{n_{I+1}}(K_{n_{I+2}}(\ldots(K_{n_{\varphi}}(Req, K^{\tau}), A_{n_{\varphi}}), \ldots), A_{n_{I+2}}), A_{n_{I+1}}) \tag{6.9}$$

The second step (6.8) shows an Onion with a second layer. The public key of the second-last node $A_{n_{\varphi-1}}$ is used to encrypt the encoded message as well as the address of the succeeding node (node $A_{n_{\varphi}}$ in this case). When node $A_{n_{\varphi-1}}$ unwraps its layer of the Onion it will retrieve a peeled Onion as well as the address of the next hop.

Note that node $A_{n_{\varphi-1}}$ does not know that it is the second-last node. It should not be able to calculate its position from the size of the embedded Onion. It does not know how many layers still need to be unwrapped to get to the Web request. It does also not know the size of the Web request. Any inference based on the size of the Onion is therefore infeasible except for very small Onions.

Inference is only possible if the Onion only has one layer *and* if the Web request is close to the minimum Web request size. This allows an adversary to guess that it is unlikely for the Onion to have more than one layer. This can easily be rectified by having the initiating node expand the size of the Web request. This can be done by inserting unneeded HTTP header values (by including fake user-agent or referrer [43] details for example). Also note that even if a node can guess how many hops are still to come, it can not infer how many hops have already been completed.

The process of recursively wrapping the Onion and the address of the next node is repeated for all nodes on the forward path. A similar Onion is

constructed for all nodes on the tail path (i.e for all $n \in S_{tail}$). The Onion for the tail path is given by:

$$(K_{n_{I-1}}(K_{n_{I-2}}(\ldots(K_{n_2}(A_{n_1}))\ldots), A_{n_{I-2}}), A_{n_{I-1}}) \tag{6.10}$$

Note that the Onion for the tail path does not have an embedded Web request. This is not necessary. The tail merely allows for a Web response to be returned to (and past) the initiating node without revealing its identity.

Pseudo-code for the Onion construction routine is detailed in algorithm 1 and 2.

Algorithm 1 Onion construction algorithm for the forward path

Require: Req, K^{τ}
1: $n_I \Leftarrow initiator$
2: $forward\_nodes \Leftarrow \{n_{I+1}, \ldots, n_{\varphi}\}$
3: $onion \Leftarrow (Req, K^{\tau})$
4: **for** $n \Leftarrow n_{\varphi}$ to n_{I+1} **do**
5: $onion \Leftarrow K_n(onion), A_n$
6: **end for**
7: **send** onion **to** n_{I+1}

Algorithm 2 Onion construction algorithm for the tail path

1: $n_I \Leftarrow initiator$
2: $tail\_nodes \Leftarrow \{n_1, n_2, \ldots, n_{I-1}\}$
3: $onion \Leftarrow ()$
4: **for** $n \Leftarrow n_1$ to n_{I-1} **do**
5: $onion \Leftarrow K_n(onion), A_n$
6: **end for**
7: **send** onion **to** n_{I-1}

Once both Onions have been constructed, they are passed to the first node on the forward and tail path respectively. Each intermediate node can only remove its respective layer from the Onion. Doing so reveals the network address of, as well as the Onion for, the next hop.

The serving and the dummy start node remove the last layer. The serving node additionally retrieves the embedded Web request. It determines which

Web server this HTTP GET request is destined for, issues the request, encrypts the response using the received symmetric key and sends the response back through the recently established communication channel. The response then traverses the channel all the way back to the dummy start node.

The initiating node can be located at any position along the channel. The initiating node accepts the response, decrypts it and passes it to the requesting Web client.

6.8.2 Channel reconstruction

Polar channels are short-lived: a new channel is required for each Polar Web request.

This differs to both Crowds [90] and Tarzan [45] where static channels are used for multiple Web requests. These static channels pose a threat to forward secrecy: once a channel's initiating node has been revealed, all prior communication could be compromised (assuming that adequate logging was done). In Polar, only a single Web request would be compromised.

Reiter and Rubin [90] claim that collaborators are more likely to link many distinct, dynamic channels to an original requester than one static channel. They argue that distinct channels could be linked using related channel content or timing of communication on channels. Whilst this is a threat, we believe that a much more sophisticated attacker is required to compromise and link (using inference) many distinct channels compared to an attacker who only needs to compromise a single channel.

In addition, node joins and failures require all static channel in Crowds to be rebuilt [90]. Crowds is thus very susceptible to denial-of-service attacks – repeatedly joining and leaving the Crowd forces the whole network to repeatedly construct new channels. The highly dynamic nature of the Polar network makes a similar approach infeasible.

In Tarzan [45], channels are long-lived because of the expensive channel setup routine. Symmetric keys first need to be shared with all participating nodes before the channel can be established. Only then can the Web request be transmitted via the channel.

Polar's Channel setup is less expensive: the serving node receives the Web request as the channel setup is completed. In fact, querying the Web server and completing the channel setup occurs almost simultaneously.

This advantage of a Polar Onion is next listed together with a number of other advantages.

6.8.3 Advantages of Polar Onions

The Polar Onion is used to establish a uni-directional channel from the serving node to the dummy start node. This differs to Onion Routing's bi-directional stream from the initiating node to the serving node.

Advantages offered by Polar Onions are two-fold: Polar Onions offer improvements over Crowds and the original Onion Routing protocol:

- Embedding the Web request inside the Polar Onion means the serving node can immediately issue the Web request as soon as channel setup has been completed. The serving node does not have to wait for the request to traverse the recently established channel. This is the case with the original Onion Routing protocol [54].

- Once a communication channel has been established, identity exposure is no longer possible by tracing the returned Web response. The response is sent past the initiating node to the dummy start node. An adversary can at most determine which nodes participate in a channel but can not infer which node originally issued the request.

- Polar is not susceptible to network edge analysis as the original Onion Routing design. Onion Routing differentiates between Onion Routers and clients. A compromised request immediately exposes the identity of the client. Polar consists solely of peers. A peer can claim that it merely forwarded the request but did not issue it.

- Onions prevent any number of corrupt nodes (except all) from compromising the Web request. All participating nodes have to successfully

unwrap an Onion before the Web request is revealed. This is particularly effective against collusion attacks. Also note that if a Web request is compromised, it does not necessarily reveal the identity of the initiating node.

- Encrypting the Web response provides a weak form of data anonymity. Encryption complicates traffic analysis; if by some means the original requester's identity is indeed exposed, Polar still guarantees that the request and response is concealed from all bar itself and the serving node.

We believe that Onion Routing offers numerous advantages over other loose routing protocols. It was furthermore shown how Polar offers numerous improvements over the original Onion Routing protocol.

6.9 Node selection

Thus far it was assumed that each Pastry node has an appropriate routing table from which channel participants can be chosen. The process of populating and maintaining such a routing table is detailed next.

Care should be taken to prevent adversaries from polluting routing tables as this could increase the chance of a collusion attack. Possible collusion attacks would thus have to be considered first before an appropriate node selection process can be suggested.

6.9.1 Collusion attacks

It was shown how communication is symmetric across all nodes on the forward channel; nodes on the forward channel display similar TCP connection attempts.

However, traffic patterns still differ between nodes on the forward and tail path. Consider an adversary capable of monitoring nodes located on the forward *and* tail path. Identity exposure is at greatest risk when the

first node on the tail and forward path, party to the same communication channel, collude.

Onion Routing makes collusion attacks more difficult but not impossible. Onion Routing allows the initiating node to choose the participants and requires collusion of *all* nodes responsible for an Onion. Only once *all* nodes have colluded is privacy compromised; the request and the original request could be revealed.

We believe identity exposure in Polar is only possible if nodes collude. One would therefore have to consider how these attacks could be performed and how likely or successful these could be.

Timing attacks

Channel setup requests addressed to two nodes in quick succession could indicate both nodes are party to the same communication channel. Note that Polar does make an attempt at hiding channel setup messages amongst Pastry control messages. This, however, only provides weak protection as colluding nodes can share information regarding received and sent channel setup messages. Channel setup messages contain Onions which are layered in encryption and differ in size and content. Thus, the only useful information would be receive and sent times.

Identifying whether two nodes participate in the same channel is therefore possible by comparing channel setup times. The request is not revealed because this requires collusion across all nodes. This attack is therefore less successful at compromising the actual Web request but could be used to guess if a node is an initiating node.

Message coding attacks

A somewhat more likely attack can be performed by analysing the returned Web response. Note that Polar does encrypt the response using end-to-end encryption. Comparing the byte-code of the response is therefore possible.

However, the response is sent through to the dummy start node. Any node between the serving node and the dummy start node could potentially

be the original requester. Traffic monitoring should therefore have occurred during channel setup. Only during channel setup is it possible to distinguish between forward and tail nodes. Identifying the original requester after a channel has been completed is no longer possible.

Global observers

The greatest threat is posed if an adversary is capable of observing the majority of nodes party to a communication. More specifically, identity exposure is possible if the first tail and forward node as well as the serving node collude. Note that this differs with Tarzan [45], mCrowds [22] and Onion Routing [110] where collusion of only the first and last hop is required.

Polar is thus vulnerable to collusion attacks. However, an attempt is made to minimise such attacks by adequate node selection. Node selection is done by retrieving nodes from a routing table.

6.9.2 Routing tables

Pastry's internal routing table is unsuitable for the purpose of selecting which nodes participate in a channel. All nodes in the Pastry routing table share a prefix of length l where $0 \leq l \leq \log_{2^b} N$. Using Pastry's internal routing table for node selection would make inference, based on the prefix of selected nodeids, possible.

The proxy layer should therefore create and maintain an additional node map. This map is geared towards improving anonymity through adequate node selection.

Note that the Pastry routing table is still required. The proxy layer depends on the routing layer for adequate node discovery and message routing. In fact, the routing layer is queried for appropriate entries for the node map. The routing table maps nodeids to their respective network address. The node map, on the other hand, maps IP addresses of participating nodes to their respective nodeid. The node map should additionally store the respective public key of a node.

There is at least one compelling reason to create an additional table that

maps IP addresses instead of nodeids: Polar should aim to maximise location diversity.

6.9.3 Location diversity

In section 3.4.3 it was discussed how location diversity plays an important role in preventing collusion attacks. In other words, when constructing an anonymous channel amongst nodes, the nodes should be chosen in such a manner as to maximise the number of administrative domains traversed by that channel.

The effectiveness of such an approach is based on the assumption that an adversary can operate multiple corrupt Polar nodes, but that the IP addresses thereof are rarely scattered throughout the IP address space. The adversary would need to acquire or compromise a sufficient number of machines located in differing administrative domains. However, an adversary is more likely to operate a contiguous address space. This assumption is also made by Tarzan [45] and MorphMix [92].

The IP address prefix gives a good (but not guaranteed) indication of how diverse two IP addresses are. Two addresses which share a short prefix are more likely to originate from different domains than two addresses which share a long prefix. Feamster et. al. [42] propose a method whereby network topology maps are used to model inter-domain routing. These are complex and difficult to obtain or generate and hence, are less for suitable for Polar. Comparing IP address prefixes is simpler, more practical and sufficiently adequate for Polar.

6.9.4 Node map

Polar's node map is related to Tarzan's routing table [45]. In fact, we borrow from Tarzan and use a hierarchical routing map that groups IP addresses into a three-tier hierarchy: first amongst all known /16 subnets (this includes all class A and B addresses) then amongst all /24 subnet and finally the complete IP address. A sample node map is illustrated in figure 6.4.

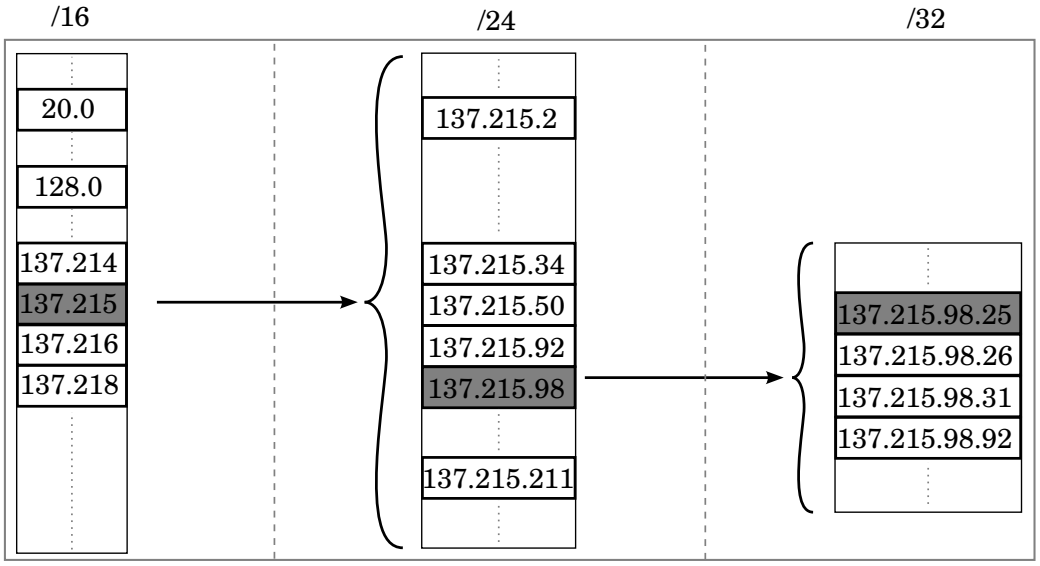


Figure 6.4: Polar's node map

Each IP address in the node map represents a discovered Pastry node. The entry maps the IP address to the node's nodeid and public key. The node map can be potentially large (however, a minimal amount of information is stored). For a Polar network of N nodes, the node map will at most contain N entries.

To construct an Onion with φ layers, φ different /16 addresses are randomly selected from the first tier of the node map. If the node map is populated with less than φ /16 addresses then multiple /24 addresses can be selected.

The set of φ nodes now presents a random sequence of $\{I_{n_1}, I_{n_2}, \ldots, I_{n_\varphi}\}$ nodes. The initiating node can now be inserted at any position in the sequence except first or last.

The biggest threat is posed if the initiating node's preceding and succeeding node and the serving node collude. The sequence should therefore be reordered in such a manner that the Initiating, the preceding, succeeding and the serving node have sufficiently distinct IP address prefixes. This can be done by selecting three nodes from the sequence that share the shortest matching prefix with each other, as well as with the initiating node. These

three nodes are then selected as the preceding, succeeding and serving nodes.

6.9.5 Populating the node map

The Pastry overlay should be used to assists in locating appropriate entries for the node map.

We propose that node discovery message be used to query remote nodes for their network address and their public key. A solution by Andersson et. al. [22, 5], titled mCrowds, proposes an enhancement to the original Crowds protocol [90]. It is proposed that the Diffie-Hellman protocol [33] be used to exchange public keys between nodes. A central authority, as employed by Crowds, is not needed.

A similar method could be adopted by Polar. Node discovery messages should be used to initiate a key exchange using the Diffie-Hellman protocol. Node discovery messages should be sent out periodically at unpredictable intervals to random nodeids. The node closest to the nodeid should respond.

It is additionally proposed that when node discovery messages are sent, nodes also share their Pastry leaf set. The leaf set was introduced in chapter 5 and contains a list of $|L|$ of the closest neighbouring nodes in the Pastry's nodeid space. The public keys of the nodes in the leaf set should also be shared.

By sharing the leaf set, additional entries can be created in the node map. These entries should be classified as "unverified" if no prior key exchange was performed.

This added measure of sharing the leaf set allows a Pastry node to learn of other nodes and to cross-check public keys reported by different nodes. This prevents any one node from assigning a different key to each requesting node.

If public keys are not cross-checked, it would be possible to perform key-mapping attacks. Key-mapping attacks are performed by sharing a different key with each Polar node. A table is kept mapping an encryption key to a Polar node. If encrypted content is received, all stored keys are tried until the successful key is found. The table identifies which node performed the

encryption. thus compromising anonymity.

Cross-checking public keys with other nodes prevents a key-mapping attack. If nodes report different public keys for any particular node, that node should be flagged as "unverified".

6.9.6 Node transience

Polar is designed to operate in a highly dynamic network of peers. Node joins and failures are expected to be frequent. This could render a large number of entries in the node map obsolete. Polar, similarly to Pastry, employs a lazy node repair algorithm. Obsolete routing table entries are removed when routing to that entry fails. A failed Onion Routing attempt should similarly be communicated back to the initiator. To prevent identity exposure, the initiating node should relay the node failure message through to the dummy start node or the serving node – depending on whether the failed node was situated on the forward or tail path. A node failure message simultaneously indicates to participating nodes that a particular nodeid is obsolete and needs to be removed. Participating nodes subsequently close the channel. The initiating node is required to restart the channel setup.

We believe that node transience is less of a problem for Polar than for Tarzan [45] as Polar's channel setup routine is less expensive than Tarzan's – Polar already initiates a new channel for each Web request, unlike Tarzan, where a long-term channel is intended.

Polar is a peer-to-peer network that is open and accessible to anyone. This, however, introduces a number of complications and are considered next.

6.10 Security

We have shown how Polar establishes an anonymous communication channel. This channel services a requesting node whilst hiding its identity from (1) the Web server, (2) participating nodes and (3) unsophisticated adversaries. However, the examples thus far have assumed a cooperative Polar network. How does Polar fare against a more sophisticated adversary with insider

knowledge and/or capable of manipulating the Polar network?

When considering the presence of global observers and powerful insiders, it becomes apparent that Polar *is* vulnerable to a number of attacks. These do not necessarily lead to identity exposure but can affect the level of anonymity and/or the reliability of the system.

Some attacks are common to a variety of anonymity solutions whilst others bear particular reference to Polar. Polar's attempt at anonymity, whilst maintaining its status as a fully distributed peer-to-peer architecture, makes for a particularly interesting discussion on anonymity, security and peer networks.

6.10.1 Security and anonymity in peer-to-peer networks

The peer-to-peer paradigm introduces a number of notable differences to traditional client-server architectures. Peer networks have become popular because of their ability to scale well and to effectively share resources in a highly transient population whilst resisting centralised control [6]. However, peer networks are also notoriously difficult to secure. Literature on security in peer-to-peer networks [19, 36] lists the following reasons why security enforcement in peer-to-peer networks is so difficult:.

- Many traditional means of ensuring accountability between participants becomes unworkable. In particular, there is a *lack of top-down enforcement* of accepted behaviour.
- The open environment allows many diverse autonomous parties, *without preexisting trust relationships*, to pool their resources.
- The large, dynamically changing network makes it difficult to *track* badly behaving participants. A pseudonym (a nodeid) that has acquired a bad reputation can simply be thrown away.
- Routing is complicated because users must choose nodes in the network without knowing the entire state of the network. This is not true for all peer-to-peer networks. It is, however, true in our case because Pastry nodes have incomplete or partial routing tables.

- An adversary might render a network useless by volunteering a flood of unreliable and/or corrupt participants. This attack is called the *Sybil* attack [38].

Nevertheless, the fact that peer-to-peer networks are resistant to centralised control, makes them particularly attractive for privacy-enhancing technologies. This fact was also the primary motivation for initially considering further research in a peer-to-peer anonymity technology.

We should note that the combination of anonymity and peer-to-peer networks introduces further complications: it is hard to detect or verify a participants behaviour and at the same time maintain a participant's anonymity [36]. As Dingledine et. al. [36] put it: *"reputation data is harder to gather in the presence of anonymity"*. However, this fact can also be used to Polar's advantage.

We argue that because Polar attempts plausible deniability (as a means of achieving anonymity), a fully distributed peer-to-peer network does offer a suitable architecture. The fact that reputation data is harder to gather can work in our favour; it increases the uncertainty (entropy) that a particular node was or was not the original requester.

Next we highlight a number of attacks on structured peer-to-peer networks. We conclude with the statement that peer-to-peer networks are suitable in attempting plausible deniability; however, it is also much harder to guarantee availability in such networks.

6.10.2 Attacks on structured peer-to-peer networks

In their paper titled "Secure routing for structured peer-to-peer overlay networks", Castro et. al. [19] offer an informative analysis of security in structured peer-to-peer networks. We use their paper as well as that of Dingledine et. al. [36] to consider different attack vectors on Polar.

According to Castro et. al. [19] secure routing in structured peer-to-peer networks requires the following:

1. secure assignment of nodeids

2. secure routing table maintenance

3. secure message routing

Unfortunately, these three requirements have not yet been addressed by Polar. Secure assignment of nodeids is discussed next followed by a discussion on the security of Polar's routing table maintenance and message routing.

Secure assignment of nodeids

A secure assignment of nodeids prevents corrupt nodes from misusing the identity of other nodes. This includes identity theft for the purposes of sending or receiving messages on behalf of other nodes. It additionally prevents a malicious user from obtaining a large number of nodeids. This attack is known as the Sybil attack [38] and is particularly threatening to Polar as it increases the chances of corrupt nodes colluding against honest nodes.

Castro et. al. [19] offer one solution only: the use of a trusted certification authority to assign and digitally sign nodeid certificates. The certification authority should restrict the number of nodeids any particular user can obtain by requesting some form of payment in return. However, even then, certification authorities are not fully guaranteed to prevent an attacker from acquiring a large collection of nodeids [19].

The use of a trusted certification authority clearly defies Polar's objective of being a fully distributed anonymising technology. It furthermore inhibits the roll-out of the Polar network by requiring users to first register with a trusted certification authority. Where privacy, central control and trust is an issue, this will prove problematic.

Castro et. al. [19] dash any hopes of a fully decentralised nodeid assignment by the statement *"it appears to have fundamental security limitations. None of the methods we are aware of can ultimately prevent a determined attacker from acquiring a large collection of nodeids"* [19, p.6].

Polar does not make any attempt at securing nodeid assignment. However, Polar's Onion Routing approach and its location diversity efforts greatly reduce the chance of a successful collusion attack and so some form of security is already built into Polar.

Secure routing table maintenance and message routing

Determined attackers can pollute routing tables by supplying bad routing table updates to existing nodes. This increases the fraction of rogue nodes in the routing table which furthermore reduces the probability of routing successfully. Polar's node discovery messages, used to populate the node map, make Polar vulnerable to such an attack.

Unfortunately the suggested solution to this problem is also less than ideal. Castro et. al. [19] assume a "relatively" secure assignment of nodeids, make use of a constrained routing table (in addition to the normal routing table) and apply a failure test to determine the successful routing of a message. Neither technique is suitable to Polar.

Tarzan [45] offers a solution by requiring routing table entries to be validated before being used. This is similar to our proposed solution of sharing the leaf set and flagging node map entries as "verified" or "unverified". In Tarzan, a score is attributed to a routing table entry. Each node that verifies the correctness of the entry increases the score. A similar technique can be applied to Polar's node map. For every leaf set received in response to a node discovery message, an entry is created in the node map, or the score of an existing entry is increased. A threshold value is then used to select those nodes with a score greater than the threshold.

A greater threat is posed by a corrupt bootstrapping node. The bootstrapping node issues initialisation information to the joining node. In such a case, an attacker can guarantee that the joining node only learns of corrupt nodes. Polar does not protect against such an event. It is left up to the user to acquire a trustworthy bootstrapping nodes.

Polar can at most offer some levels of protection to users who join a legitimate network. Polar's Onion Routing and location diversity offers some protection provided not all nodes are compromised.

If our assumption holds, that an adversary operates most nodes from within an administrative domain, then it is possible to employ "sanity checks". An alarm should be triggered when most routing table entries contain nodes from only a few (or even one) administrative domains.

6.11 Summary of Polar

A brief summary of the chapter is presented next. It highlights the contributions made by the author. It additionally gives an analytical overview of Polar and briefly compares it to previous solutions.

Plausible deniability

Polar achieves anonymity though plausible deniability. Anonymity is upheld if there is sufficient doubt about whether a node issued the original request or not.

Although Polar employs a derivative of Onion Routing, it is not susceptible to network edge analysis as the original design. Onion Routing differentiates between Onion Routers and clients. A compromised request immediately exposes the identity of the client. Polar consists solely of peers. In Polar, the "client" could claim that it merely forwarded the request but did not issue it.

Crowds [90] and Tarzan [45] also employ plausible deniability. However the initiator can still be exposed even after the request has been issued. This is possible if the request can be traced back to the initiating node. Polar prevents such "trace-back" attacks through the use of a tail. The response can no longer expose the identity of the initiator.

Protection against log analysis

A tail does increase the risk of collusion attacks between nodes on the forward and tail channel. More importantly, however, it prevents identity exposure through traffic analysis performed on a single machine. We perceive this threat to be far greater than the threat of colluding nodes. Polar is therefore particularly resistant against simple log analysis performed by organisations, ISPs or external observers.

Fully decentralised

Polar provides a routing layer which allows it to organise and maintain a network of peers in the absence of a central authority. This makes Polar particularly resistant against compulsion attacks (achieved through extortion, bribing or legal subpoenas).

Resistance against collusion attacks

Although collusion attacks pose a threat to Polar, they are severely complicated by Polar's combined attempts at Onion Routing and node diversity. A highly sophisticated adversary is required to expose Polar's anonymity efforts.

Efficiency

Channel setup is less expensive than Tarzan's. The differential treatment of Web requests and Web responses allows Polar to transmit the Web request inside an Onion. Two round-trips, one for the channel setup and one for the transfer of the Web request, are no longer needed. A response can be issued as soon as the channel has been established.

Forward secrecy

The less expensive channel setup additionally allows a new channel to be efficiently built for each request. A compromised channel only implicates one request. This differs to Crowds [90] and Tarzan [45] where a compromised channel implicates all transmitted Web requests.

Peer validation

Polar adopts Tarzan's peer validation concept. Although a weak form of security, it does add an extra layer of protection and prevents key-mapping attacks.

Peer-to-peer security

Polar's peer-to-peer network has some inherent security implications. Polar tries to minimise the severity of the implications. Polar Onions require collusion amongst all participating nodes. Peer validation complicated network-wide attacks. Polar's node diversity and proposed "sanity-check" offers some means to alert the user of a possibly insecure peer network.

6.12 Conclusion

This chapter discussed the Polar model. The author's main contributions were presented, analysed and compared to existing solutions.

Message-based communication for the Web request and stream-based communication for the Web response was suggested. Such an approach enhances the anonymity efforts of Polar. Communication symmetry was introduced and a communication tail was subsequently proposed.

A Polar Onion enables secure message-based communication and also establishes a communication channel. The combination of using layered encryption for the request and a communication tail for the response gives Polar significant advantage over existing solutions.

Pastry continues to play a vital role in organising the network. It additionally assist in populating the Polar node map.

Polar's attempt at anonymity in a fully distributed peer-to-peer network makes for an interesting discussion on security. Ultimately, Polar's plausible deniability and location diversity properties allow it to protect its users by severely complicating traffic analysis. A sophisticated global adversary is required to adequately and consistently breach Polar's communications. This is highly unlikely.

The following chapter presents implementation details. It aims to detail any specifics omitted in this chapter.

Chapter 7

Implementation

7.1 Introduction

In this chapter the author's implementation of Polar is detailed. The proof-of-concept prototype serves to prove that an implementation of Polar is feasible.

The proxy layer in particular is examined. We start by listing internal components of a Polar node. Their role and functionality is described using flowcharts. This chapter aims to clarify any remaining details which have not yet been explicitly stipulated.

7.2 Prototype

A proof-of-concept prototype was developed in the Java programming language and consists of roughly 1500 lines of code. The prototype complies to the Java language specification version 1.5; it is capable of running on most operating systems provided that the Java run-time version 1.5 or greater has been installed. This makes the application accessible to a large user-base.

An open-source version of Pastry called FreePastry [94] is used for the routing layer. FreePastry is a Java implementation of Pastry and is developed and maintained at the University of Rice.

The prototype serves in identifying any issues not considered during the theoretical design of Polar. This implementation was not widely deployed

and hence no statistical data or benchmarks were collected.

To effectively benchmark Polar one would have to acquire a large user base from many different autonomous systems. A fully developed Polar application would thus have to be made available for general download. A prototype not robust enough to be used for production would most certainly discourage users from using it, thus hindering the collection of adequate statistical data.

The aim of this dissertation is not the deployment and benchmarking of Polar, but rather to introduce the Polar model and reason its suitability as an anonymous Web browsing protocol. The prototype merely serves as a proof-of-concept demonstrating that an implementation is possible. How such an implementation could be designed is also addressed. A fully-fledged implementation and deployment of Polar would most certainly assist in further promoting the Polar model; however, this is not presented in this dissertation and is left as future work.

The presentation of our Polar model is completed by providing a more fine-grained functional insight into the suggested logical components of a Polar node. Components are grouped into modules, constructs and participants.

7.3 Modules, constructs and participants

The author suggests that seven functional modules facilitate the operation of a single Polar node. The modules interact with three logical constructs and create and maintain an anonymous communications channel between one or more local clients and multiple Web servers.

Figure 7.1 illustrates the relationship between modules, constructs and the external participants. Modules perform the required functionality of a Polar node. A construct is used here to describe a logical entity created and maintained by one or more Polar nodes. For example, a channel is logical construct created and operated by multiple Polar nodes. Similarly for a Pastry message; a message is created by a particular node and is used to communicate information to other nodes. The node map is unique for each Polar node. Its main use is to store pointers to participating nodes:

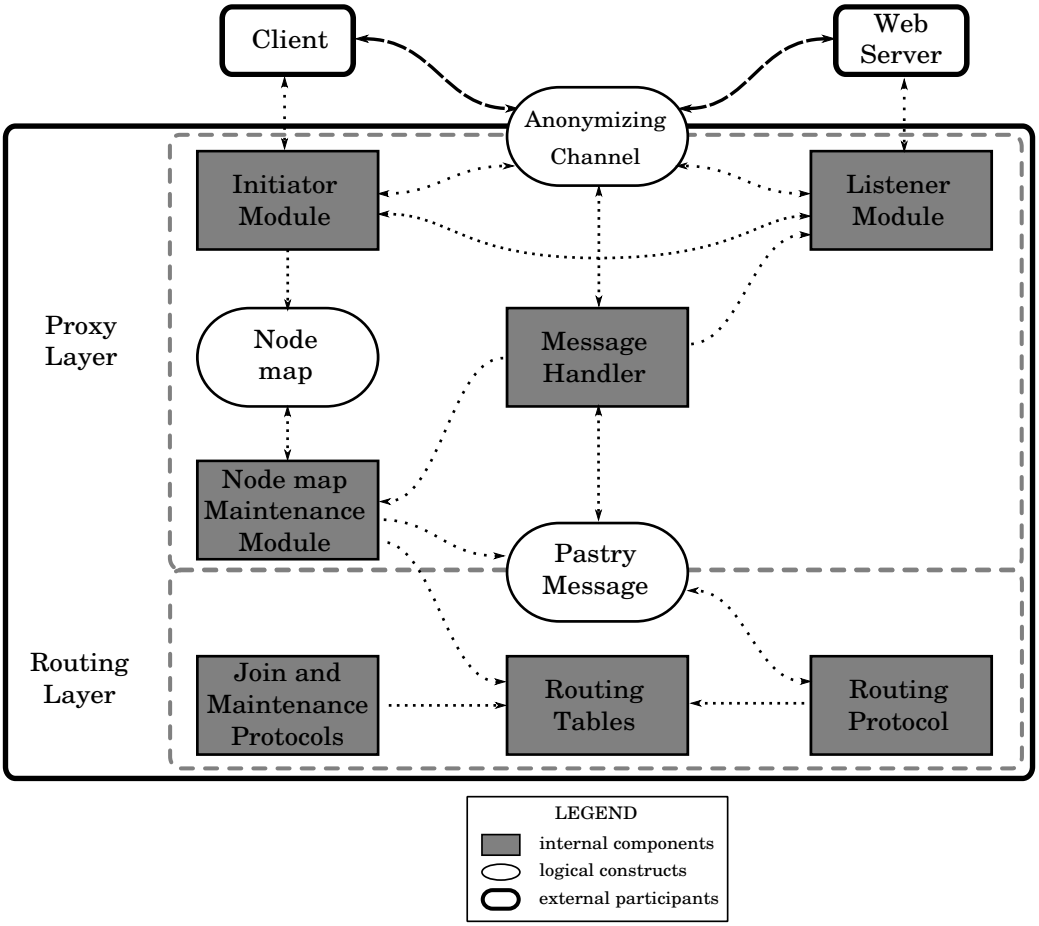


Figure 7.1: Components of a Polar node

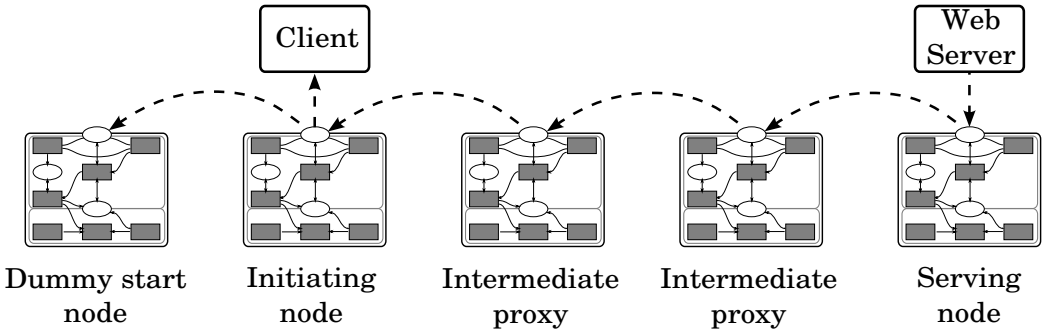


Figure 7.2: Overview of a Polar communication

Each module is responsible for a specific set of tasks.

- The **initiator module** accepts client Web requests, selects a set of nodes from the node map, constructs the Onions and initiates the setup of a channel.
- The **message-handler module** processes message-received events triggered by the routing layer. Messages received include channel setup messages, channel failed messages and node discovery messages. The module performs the necessary steps according to the message type.
- The **listener module** listens for and accepts TCP/IP connections. It links an inbound and an outbound connection belonging to the same channel. Data received from the inbound connection is tunnelled through the outbound connection.
- The **node map maintenance module** ensures the node map is populated with appropriate values.
- The **routing protocol** implements the Pastry routing algorithm. The protocol routes a message to a given nodeid.
- The **routing tables** consists of Pastry's leaf set, neighbourhood set and the primary routing table.
- The **join and maintenance protocols** are also specific to Pastry. The protocols ensure that Pastry's routing tables are up-to-date.

Figure 7.1 groups the components according to the layer they operate in. The initiator, listener, message-handler and node map maintenance module operate at the proxy layer. The routing tables as well as the routing, join and maintenance protocols are situated in the routing layer.

The three logical constructs are used in the proxy layer. Each node manages a node map and deals with several channels and multiple Pastry messages.

- The **Pastry messages** facilitate message-based communication between a node's routing and proxy layer. The routing layer is able to send messages to other nodes. Pastry messages include Pastry's control messages as well as Polar-specific messages. The proxy layer deals with Polar-specific messages such as channel setup, channel failure and node discovery messages.

- The **channel** facilitates a communication between a client, multiple nodes and a Web server. An graphical illustration is given in figure 7.2 by the dashed lines. The arrows indicates the flow of the Web response.

- The **node map** is unique to every node. It maps a collection of IP addresses to the nodeids and public encryption keys of the respective nodes. This information is used during the construction of a Polar Onion.

The proxy layer establishes and maintains the anonymous communication channel. The tasks performed by the proxy layer are unique to Polar and hence comprises solely of the author's code. The routing layer on the other hand is not developed by the author but is an implementation of Pastry researched and developed at the University of Rice (`http://www.rice.com`).

The three routing layer modules represent a very coarse overview of FreePastry's functional entities. These three modules could possibly be broken down into more fine-grained modules. This dissertation is more concerned with the proxy layer and hence the focus is on the four proxy layer modules.

Insight into the functional operation of the modules is presented next.

7.4 Functional details

Three proxy layer modules are discussed in the following order: first the initiator module followed by the connector and finally the listener module.

The node map maintenance module was discussed in the previous chapter. A discussion on the implementation of the node map maintenance module

would add little to what has already been discussed. A more detailed discussion of this module is therefore not presented here.

7.5 Initiator module

A user wishing to anonymise his Web request should configure his client to redirect requests to a local Polar node, and not directly to the Web server. Polar listens to a pre-configured port and accepts client TCP/IP connections originating from the local machine. Note that once a connection has been accepted, Polar should immediately re-enter the listening phase, thereby supporting multiple concurrent connections. The two tasks of listening for and accepting connections are indicated by the first two steps in figure 7.3.

Once a connection has been accepted, the HTTP request is captured. The initiator module reads from the data stream and assumes that the content conforms to the HTTP /1.0 [13] or HTTP /1.1 [43] specification; the stream is read eight bytes (an octet) at a time and mapped to the ASCII [105] character set. This suffices to read and interpret characters for HTTP requests and responses [13, 43].

Simple parsing is done to capture a Web request: the character stream is scanned for an empty line (i.e. two successive carriage return and line feed characters). This indicates the end of an HTTP request. Channel setup can now commence.

First a number of addresses are selected from the node map. The node map additionally provides an appropriate nodeid, port number and public key for the respective node located at that IP address. The node selection algorithm is detailed in the previous chapter. The channel length can be user-specified or can be a random range-bound value. Future implementations should provide a simple user interface where a length can be specified.

Once nodes have been selected, the initiator can start constructing the tail and forward Onion. This procedure is also detailed in the previous chapter in algorithms 1 and 2 in section 6.8.1.

The two Onions are subsequently embedded in separate Pastry messages. The messages are addressed to the fist hop on the tail and forward path

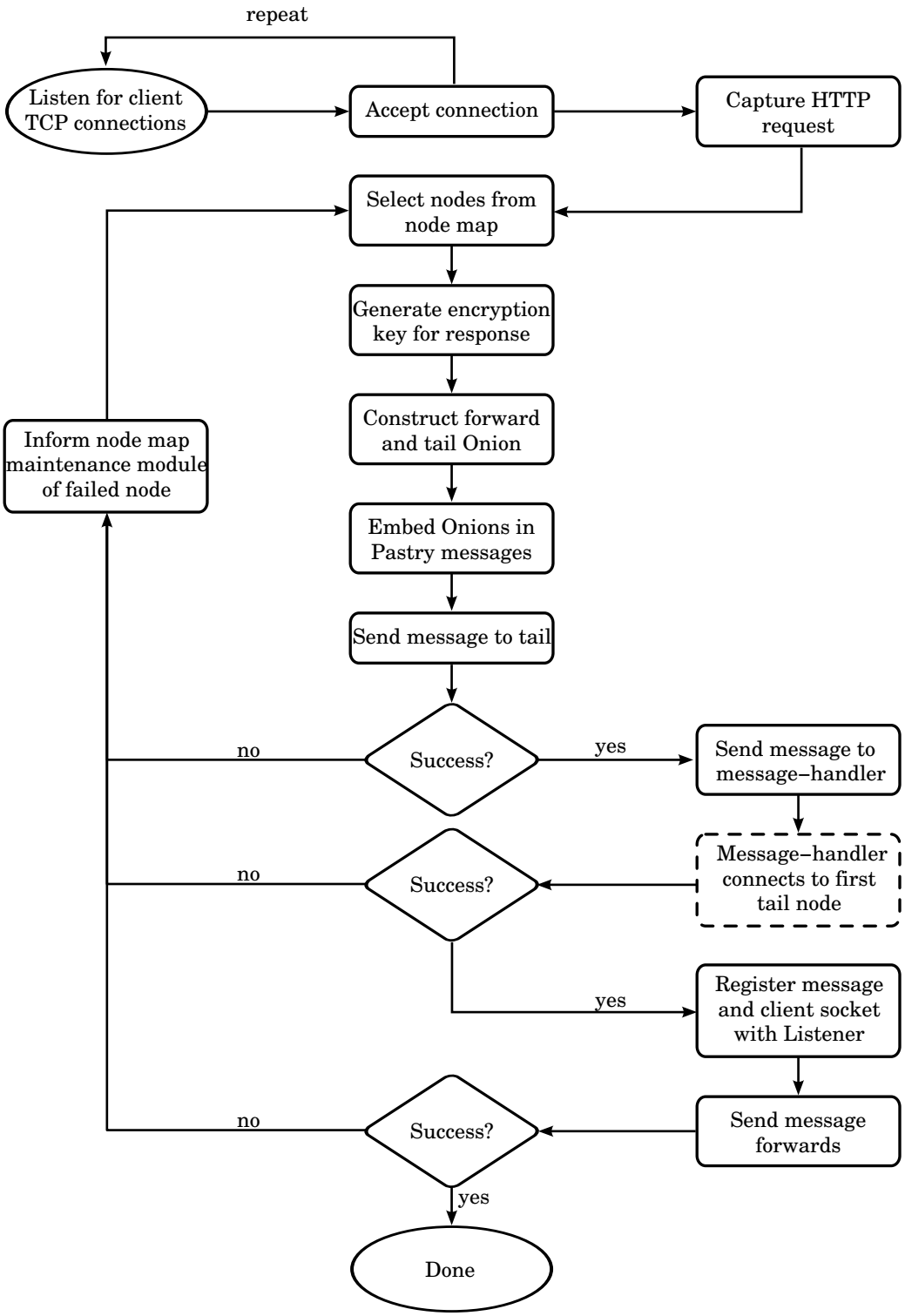


Figure 7.3: Functional flowchart of the initiator module

respectively. The routing layer is instructed to deliver the messages to the specified IP address and port number. The embedded Onions instruct the respective nodes to continue constructing the anonymous communication channel.

As per our discussion in section 6.6, in order to combat traffic analysis on a single node, the tail Onion should be sent prior to the forward Onion. The initiating node thus displays similar traffic patterns to other nodes on the forward channel.

The Pastry routing layer routes the messages directly to the specified node and not via a number of intermediate nodes; the target node's IP address is specified and not the nodeid. Although it is possible to bypass the routing layer and send the message using the proxy layer alone, it makes little sense to duplicate the messaging functionality already provided by the routing layer. Note that FreePastry allows messages to be addressed to a nodeid or IP address and port number.

It should be noted that the benefits of the Pastry routing overlay is best harnessed by the node map maintenance module. As discussed in section 6.9.4, the node map is populated by acquiring the Pastry leaf set of other nodes. This task is performed by the node map maintenance module. The initiator module merely depends on the accuracy of the node map, as well as the existing messaging capability of the routing layer.

Figure 7.3 indicates three possible fail-over points. At each point a conditional statements verifies whether message delivery or channel setup succeeded. Failure to send or connect to any node results in the node map maintenance module being notified. The respective entry is removed from the node map and the channel setup then needs to be re-initiated.

If no failure is encountered, the initiator module delegates the remaining tasks to the message-handler and the listener module.

Note that the initiating node connects to a tail node and instructs the forward node to connect to it. The message-handler actively establishes a connection to another node. The listener module is responsible for accepting connections initiated by the message-handler. The message-handler thus typically connect to a remote listener module. The listener module addition-

ally links an inbound and an outbound connection thus linking a preceding and a succeeding node to form a channel.

The initiator module is only active during the initial phase of a channel setup. Once the Onions have been dispatched, the tasks of the initiator are complete. It will only be called upon (again) if the channel setup fails. The client TCP connection remains open until the Web response is served to the client. Responsibility of maintaining the client connection is passed to the listener module.

The message-handler is discussed next followed by the listener module.

7.6 Message-handler module

The flowchart of the message-handler module is depicted in figure 7.4. The message-handler module responds to message-received events, triggered by the node's routing layer (or directly by the initiator module). The module reacts to three different types of messages: channel setup messages, channel failure messages and node discovery messages.

7.6.1 Node discovery messages

A node discovery message indicates that another node wishes to obtain the current node's leaf set. Node discovery messages are dispatched at regular and unpredictable intervals by the node map maintenance modules. The returned leaf sets are used by the maintenance module to populate the local node's node map. The reader is reminded that the leaf set contains a set of nodes that are closest to the current node in the circular nodeid space.

7.6.2 Channel failure messages

Channel failure message indicate channel construction on either the tail or the forward path has failed. The failure message is routed from the point of failure right through to the serving or dummy start node depending on whether the point of failure is located on the tail or forward path.

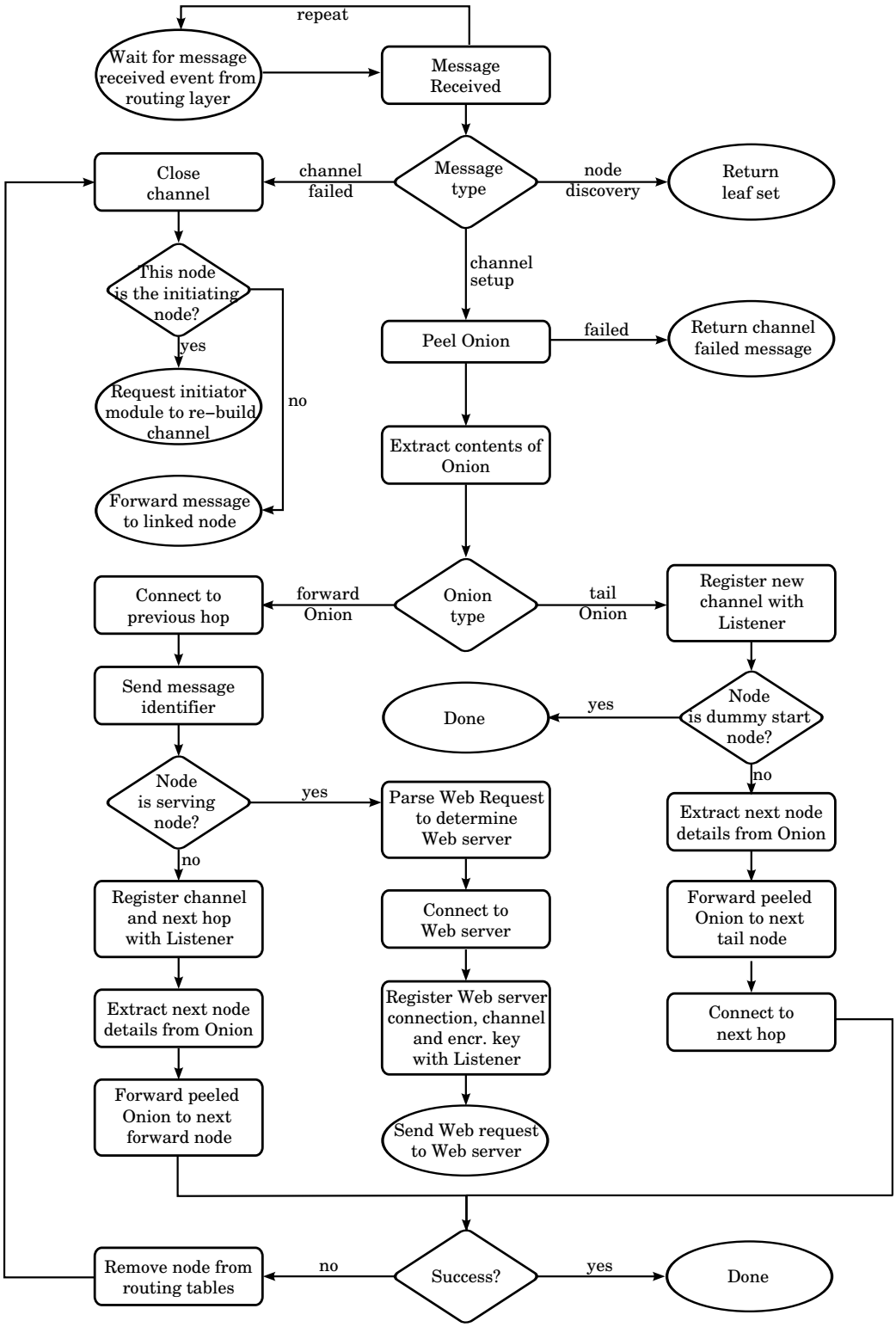


Figure 7.4: Functional flowchart of the message-handler module

Located somewhere in between is the initiating node. The initiating node re-initiates the channel; it chooses a new set of nodes and re-constructs and re-sends both Onions. This process is repeated until channel setup succeeds.

What implications the repetitive channel reconstruction has on possible identity exposure, has not been verified. This is left as future work.

7.6.3 Channel setup messages

Channel setup messages contain an embedded Onion. Failure to peel an Onion results in the current node returning a channel failure message.

Successfully peeling an Onion reveals its contents. The content could be one of three possibilities:

- an empty Onion – indicating the current node is the dummy start node
- the Web request and symmetric key – indicating the current node is the serving node
- details of the next hop as well as the next hop's Onion – indicating that the current node is an intermediate proxy

A discussion on the three possibilities is presented next. The respective functions performed by the message handler are depicted in figure 7.4 starting at the conditional statement labelled "Onion type".

Dummy start node

If the current node is the dummy start node then channel setup, for the tail, is complete. No further action is required by the dummy start node; it waits until the Web response is received or until the channel is closed. The Web response is simply discarded.

Serving node

If the current node is the serving node, a connection is established with the Web server. Determining which server the request is intended for is done by

parsing the Web request until the HTTP HOST header field is found. This header field specifies the host address of the Web server and is a mandatory header for HTTP requests [13, 43]. The format of the HOST header field is given as:

```
Host = "Host" ":" host [ ":" port ]
```

The `host` value represents a host name or IP address. The optional field `port` is the IP port number. Incorrectly formatted Web requests result in the channel being closed.

Once a connection has been established with the Web server, the connection as well as the symmetric encryption key is registered with the listener module. The connection and key is registered because the listener module is responsible for relaying the returned Web response via the appropriate channel. Note that the listener module links an inbound and outbound connection. The message-handler thus issues the request and passes the responsibility to the listener module.

The listener module is responsible for performing similar functions for intermediate proxies. Intermediate proxies are covered first before detailing the functions of the listener module.

Intermediate proxy

Intermediate proxies need to extend the channel by linking succeeding and preceding nodes.

By analysing figure 7.4 it can be observed how tail and forward Onions are treated slightly differently by the message-handler. In the previous chapter it was discussed how forward nodes display an outbound connection (to the previous node) followed by an inbound connection (from the succeeding node). Tail nodes display an inbound connection (from the preceding node) followed by an inbound connection (to the succeeding node).

The functions performed by the tail and forward nodes are thus quite similar, only the order differs slightly. Our implementation uses the message-handler module to perform outbound connections whilst inbound connections are accepted by the listener module.

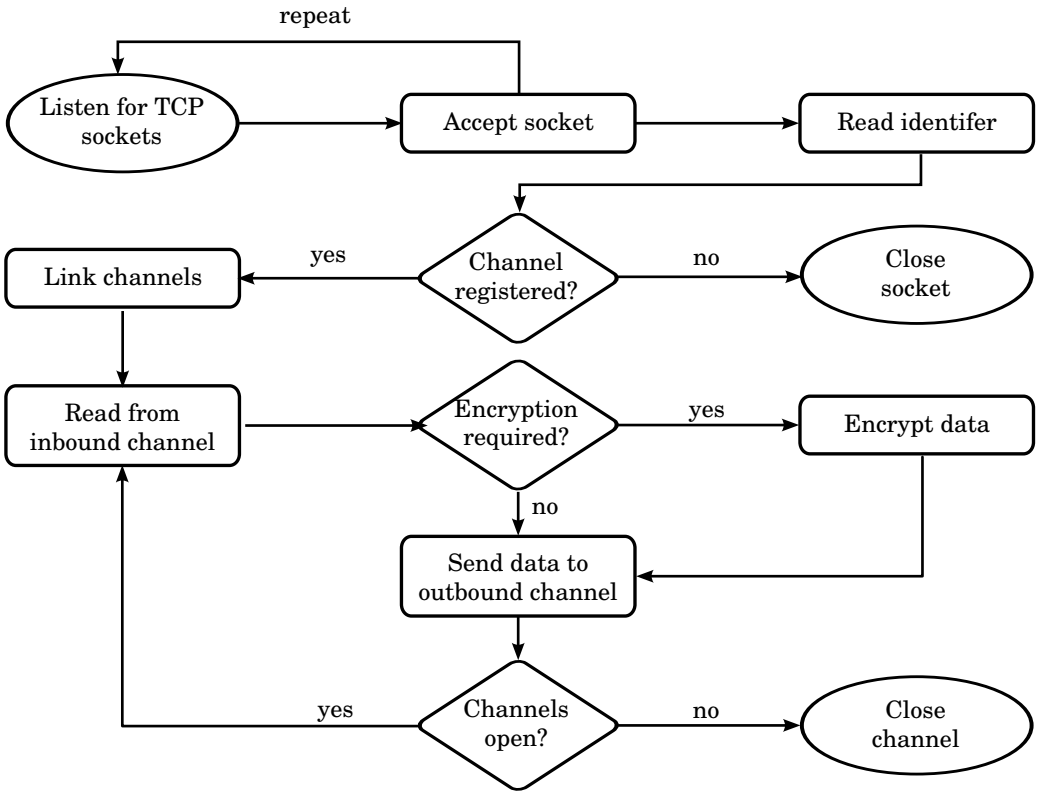


Figure 7.5: Flowchart of the listener module

The message-handler alerts the listener module of an expected inbound connection. The listener module additionally notes from which node the inbound connection is expected. Once the inbound connection had been accepted and identified, it can be linked to the outbound connection. The task of reading from the inbound connection and relaying to the outbound connection is performed by the listener module.

7.7 Listener module

The listener module completes the channel setup by joining the preceding and the succeeding node's channel. At the final node, the channel and the connection to the Web server is joined.

A channel is registered with the listener by either the initiator module

or the message-handler module. In both cases the respective message, used to create the channel, is registered as well. The message contains necessary details to distinguish one connection from the other.

A primary function of the listener module is to listen for inbound TCP connections and accept them.

Two nodes could potentially share multiple channels. To distinguish the one from the other, the Pastry message identifier is used. This identifier should present the first n bytes of data that is sent through the channel. The listener reads these bytes as a numeric identifier and determines if an appropriate channel has been registered. If this is the case, the inbound and the outbound channel can be linked. This means that each read on the inbound channel is followed by a write on the respective outbound channel. The listener module thus functions as a simple relay.

The listener module on the final node performs an additional task of encrypting the response using the symmetric key (which was embedded in the Onion). The listener module on the initiator module decrypts the data and passes it to the Web client.

Note that the initiating node links an inbound channel to two outbound channels: one to the client and one the the first tail node.

This completes the discussion on our implementation of Polar. Although other approaches are possible, we believe that the suggested implementation performs the required functionality in an optimal and logical manner.

7.8 Conclusion

This chapter presents a more technical discussion on our implementation of Polar. The reader should now have a detailed understanding of Polar's objectives and how it is proposed these objectives are achieved.

A final task is to conclude this dissertation. This is done by briefly summarising what has been discussed, what problems where identified and how they were addressed.

Chapter 8

Conclusion and future work

8.1 Introduction

Now that Polar has been fully presented, a re-evaluation of Polar's objectives and how Polar addresses these, is possible.

The author believes that Polar offers an adequate solution to the given problem statement. Achieving absolute and practical anonymity is a difficult; however, the contributions made by this dissertation are thought to be significant and should further assist in promoting anonymity research.

We conclude with a summary of this dissertation.

8.2 Summary

The introductory chapter highlighted the need for electronic privacy. It was argued that the lack of online privacy should be a cause for concern. Individual users risk being tracked and profiled by Web sites, ad-serving companies, organisational proxies, ISPs and other unauthorised observers.

Anonymity is appropriate if personalised services are not required. On the Internet, this should include traditional anonymous activities such as voting, counselling, whistle-blowing, refereeing and voicing political and other dissent. It should, however, additionally include performing Web searches and general (unpersonalised) Web browsing. The user's private browsing history

should be protected.

This dissertation covered connections anonymity technologies, with specific interest in anonymous Web browsing.

Problem statement

Anonymity solutions do exist but suffer from a number of deficiencies: they are not commonly used, are vulnerable to a host of attacks or are impractical or too cumbersome for daily use. Most anonymity solutions are centralised or partially centralised and require trust in the operators.

An extensive review of existing anonymity technologies was presented. It was noted how mixing techniques are more suitable for low-latency applications such as email and generally offer good levels of anonymity. Similar attempts for anonymous Web browsing have been less successful, largely due to the high delays associated with mixing.

Crowds [90] attempts anonymous Web browsing, not through mixing techniques, but by plausible deniability. It too suffers from a multitude of deficiencies, more notably: a partially centralised architecture, inefficient routing and weak levels of anonymity. Polar's aim is to offer a solutions to these deficiencies.

Polar aims to be (1) fully distributed, (2) offer adequate levels of anonymity and (3) enable users to browse the Internet anonymously without overly complex mixing techniques. Our proposed solution uses Pastry as the underlying peer-to-peer routing protocol. This allows Polar to operate in a highly dynamic peer network whilst being fully distributed. The author believes that Polar's level of anonymity exceeds that of Crowds and compares well to other existing solutions. Although Polar is vulnerable to a number of traffic analysis attacks, it is significantly difficult for an attacker to expose the identity of a Polar request.

Objectives

This dissertation aimed to contribute towards three aspects. First and foremost it wished to address our problem statement. Secondly, it aimed to

address the identified higher-level objectives. Lastly, it aimed to provide numerous technical improvements over similar existing techniques.

Chapter 4 derived five objectives from a set of problems identified by our literature review. Now that the Polar model has been detailed, a re-evaluation of these objectives is possible.

- **Low-latency:** Polar offers an anonymous Web browsing solution. Specific attempts are made at reducing latency. Embedding the Web request inside the Polar Onion allows the serving node to immediately issue the Web request. The serving node does not have to wait for the request to traverse the recently established channel.

- **Decentralised:** a structured routing overlay allows Polar to be fully distributed. This makes Polar particularly resistant against compulsion attacks (achieved through extortion, bribing or legal subpoenas). Pastry [96] was chosen because of it's ability to self-organise a highly dynamic network of peers.

- **Costs:** bandwidth and hardware costs are covered by participants. Polar is designed to be a free service that is self-maintained and operated by volunteers. This alleviates logistical costs and problems associated with obtaining and maintaining dedicated hardware, personnel and bandwidth.

- **Large user base:** participation is uncontrolled and open to anybody; no central authority controls the network. It was discussed how anonymity can only be provided if a user actively participates in the network – each user requires a local Polar node. Abusing the Polar service without active participation is futile.

- **Usability:** Polar's source routing allows the initiating node to choose the channel length. The channel length can be user-specified according to the desired strength of anonymity versus the expected latency. This feature is indifferent to what is offered by Onion Routing. However, Polar's tail allows the initiating node to be situated closer to the serving

node and simultaneously offer improved levels of anonymity compared to an approach without a tail.

Solution

Polar's channel setup is less expensive than that used by Tarzan [45]. The differential treatment of Web request and Web response allows Polar to treat requests as messages and the response as a data stream. The request can subsequently be embedded in an Onion.

Two round-trips, one for the channel setup and one for the transfer of the Web request, are no longer needed. A response can be issued as soon as the channel has been established.

A Polar Onion is used to establish a uni-directional channel from the serving node to the dummy start node. This differs to Onion Routing's bi-directional data stream from the initiating node to the serving node.

Onions prevent any number of corrupt nodes (except all) from compromising the Web request. All participating nodes have to successfully unwrap an Onion before the Web request is revealed. This is particularly effective against collusion attacks.

Once a communication channel has been established, identity exposure is no longer possible by tracing the returned Web response. An adversary can at most determine which nodes participate in a channel but can not infer which node originally issued the request.

Polar is not susceptible to network edge analysis as the original Onion Routing design. Onion Routing differentiates between Onion Routers and clients. Polar consists solely of peers. Unlike Onion Routing, a compromised Polar request does not necessarily expose the initiator. A peer can claim that it merely forwarded the request but did not issue it, thus, plausible deniability guarantees some level of protection.

A tail does increase the risk of collusion attacks between nodes on the forward or tail channel. More importantly, however, it prevents identity exposure through traffic analysis performed on a single machine. We perceive this threat to be far greater than the threat of colluding nodes. Polar

is therefore particularly resistant against simple log analysis performed by organisations, ISPs or external observers.

The less expensive channel setup additionally allows Polar to efficiently build a new channel for each request. A compromised channel only implicates one request. This differs to Crowds [90] and Tarzan [45] where a compromised channel implicates all transmitted Web requests.

8.3 Reflection

A number of people and factors where instrumental in the completion of this dissertation, both on an academic as well as a personal level.

On an academic level, two papers [111, 112] in particular assisted in formulating ideas that eventually led to the final Polar model. Papers were co-authored with Olivier [111, 112] and Neumann [111] and were published in peer-reviewed conference proceedings.

- H. Tillwick, T. Neumann, M.S. Olivier, H.S. Venter, and J.H.P Eloff. Polar: Proxies collaborating to achieve anonymous Web browsing. *Proceedings of the Fifth International Network Conference (INC2005)*, pages 317–324, Samos, Greece, July 2005. SM Furnell, PS Dowland and G Kormentzas (eds).

- H. Tillwick and M.S. Olivier. Towards a framework for connection anonymity. In *Research for a changing world Proceedings of SAICSIT 2005*, pages 113–122, White River, South Africa, September 2005. J Bishop and DG Kourie (eds).

The first paper [111] details the initial Polar model and provides the foundation upon which the revised model is built. The second paper [112] greatly assisted in structuring and analysing existing connection anonymity techniques. The knowledge acquired was subsequently used to enhance the original Polar model.

8.4 Future work

Now that the Polar model has been presented, it opens the possibility of a more extensive, practical and statistical analysis of Polar. Polar has not been formally analysed in a specification language. Such an approach could reveal additional deficiencies not identified in the theoretical design of Polar.

Polar has not been extensively benchmarked and tested. A proof-of-concept prototype was implemented but due to time and resource constraints, only limited testing was performed. We believe adequate statistical data can only be collected by observing a fully-deployed Polar network. This is because Polar requires a reasonably large user base from many diverse network locations.

A foreseeable practical problem is the initial deployment of Polar when the user base is small. Deploying, promoting and benchmarking Polar is left as future work.

In conclusion, we additionally hope that the contributions made by our model and framework will promote further advances in connection anonymity.

Glossary of Abbreviations

AS	Autonomous Systems
ASCII	American Standard Code for Information Exchange
API	Application Programmer Interface
CA	Certification Authority
CAN	Content Addressable Network
CPC	Cost-per-click
CIDR	Classless Inter-Domain Routing
DNS	Domain Name System
DHT	Distributed Hash Table
FIFO	First in, first out
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
ISO	International Standards Organisation
ISP	Internet Service Provider
LIFO	Last in, first out
LPWA	Lucent Personalized Web Assistant
MD5	Message Digest 5

OECD Organisation for Economic Co-operation

POP3 Post Office Protocol version 3

P2P Peer-to-peer

P3P Privacy for Preferences Project

PGP Pretty Good Privay

PKI Public Key Infrastructure

SIP Session Initiation Protocol

SOAP Simple Object Access Protocol

SMTP Simple Mail Transfer Protocol

SHA Secure Hash Algorithm

TCP Transmission Control Protocol

TTP Trusted Third Party

UDP User Datagram Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

Bibliography

[1] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the Economics of Anonymity. In Rebecca N. Wright, editor, *Proceedings of Financial Cryptography (FC '03)*. Springer-Verlag, LNCS 2742, January 2003.

[2] Hassan Aljifri and Diego Sanchez Navarro. Search engines and privacy. *Computer and Security*, 23(5):379–388, 2004.

[3] Nate Anderson. The ethics of using AOL search data. `http://arstechnica.com/news.ars/post/20060823-7578.htm`, August 2006. Last accessed 15 September 2006.

[4] Ross Anderson. The eternity service. In *Proceedings of Pragocrypt '96*, pages 242–252, Prague, Czech Republic, September 1996.

[5] Christer Andersson, Reine Lundin, and Simone Fischer-Hübner. Privacy-enhanced wap browsing with mcrowds - anonymity properties and performance evaluation of the mcrowds system. In *Proceedings of the Fourth annual ISSA 2004 IT Security Conference*, Johannesburg, South Africa, June 2004.

[6] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computer Surveys*, 36(4):335–371, 2004.

[7] Anonymizer. Anonymous surfing. `http://www.anonymizer.com/consumer/products/anonymous_surfing/`. Last accessed 1 May 2006.

[8] Annie I. Antón, Qingfeng He, and David L. Baumer. The complexity underlying jetblues privacy policy violations. Technical report, The Privacy Place, 2003.

[9] John Argyrakis, Stefanos Gritzalis, and Chris Kioulafas. Privacy enhancing technologies: A review. In *EGOV*, pages 282–287, 2003.

[10] Adam Back, Ian Goldberg, and Adam Shostack. Freedom systems 2.1 security issues and analysis. White paper, Zero Knowledge Systems, Inc., May 2001.

[11] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, April 2001.

[12] Jerry Berman and Deirdre Mulligan. Privacy in the digital age. *Nova Law Review*, 23(2), 1999. Work in progress.

[13] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0, 1996.

[14] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.

[15] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, July 2000.

[16] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[17] Wesley A Brandi and Martin S Olivier. On privacy and the Web. In *Proceedings of the Fourth Annual Information Security South Africa Conference (ISSA2004)*, Midrand, South Africa, June/July 2004. Information Security South Africa (ISSA). Published electronically.

[18] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20:1489–1499, 2002.

[19] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 299–314, New York, NY, USA, 2002. ACM Press.

[20] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[21] David L. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75, 1988.

[22] Reine Lundin Christer Andersson, Simone Fischer-Hübner. Enabling anonymity for the mobile internet using the mcrowds system. In *FIP WG 9.2, 9.6, 11.7 Summer School on Risks and Challenges of the Network Society*, 2004.

[23] Winnie Chung and John Paynter. Privacy issues on the Internet. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, pages 7–10, January 2002.

[24] I. Clarke, S.G. Miller, T.W. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with freenet. *Internet Computing, IEEE*, 6:40–49, 2002.

[25] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. `http://freenet.sourceforge.net`, June 2000.

[26] George Danezis. Mix-networks with restricted routes. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

[27] George Danezis. Better anonymous communications. Master's thesis, University of Cambridge, January 2004.

[28] George Danezis and Ross Anderson. The economics of censorship resistance. In *Proceedings of Workshop on Economics and Information Security (WEIS04)*, May 2004.

[29] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[30] Claudia Díaz and Bart Preneel. Reasoning about the anonymity provided by pool mixes that generate dummy traffic. In *Proceedings of 6th Information Hiding Workshop (IH 2004)*, LNCS, Toronto, May 2004.

[31] Claudia Díaz and Bart Preneel. Taxonomy of mixes and dummy traffic. In *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, Toulouse, France, August 2004.

[32] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Proceedings of PET 2002*, San Francisco, United States of America, April 2002. Springer-Verlag GmbH.

[33] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[34] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[35] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In "Security and Usability", an O'Reilly Media book, August 2005.

[36] Roger Dingledine, Nick Mathewson, and Paul Syverson. Reputation in P2P Anonymity Systems. In *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[37] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[38] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer-Verlag, 2002.

[39] D. Eastlake and P. Jones. US secure hash algorithm 1 SHA1, 2001.

[40] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Trans. Inter. Tech.*, 3(1):1–27, 2003.

[41] AMA ePolicy Institute Research. 2005 Electronic Monitoring & Surveillance Survey. `http://www.amanet.org/research/pdfs/EMS_summary05.pdf`, 2005. Last accessed 1 May 2006.

[42] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*, Washington, DC, USA, October 2004.

[43] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999.

[44] FIPS. Secure hash standard. In *Federal Information Processing Standards Publication 180-1*. US Department of commerce/NIST, National Technical Information Service, April 1995.

[45] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM conference on computer and communications security*, pages 193–206, Washington, DC, USA, 2002. ACM Press.

[46] Michael J. Freedman, Emil Sit, Josh Cates, and Robert Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 121–129. Springer-Verlag, 2002.

[47] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (cidr): an address assignment and aggregation strategy, 1993.

[48] Eran Gabber, Phillip B. Gibbons, David M. Kristol, Yossi Matias, and Alain Mayer. Consistent, yet anonymous, Web access with lpwa. *Commun. ACM*, 42(2):42–47, 1999.

[49] Andrew Goldberg and Peter N. Yianilos. Towards an Archival Intermemory. In *Proceedings of IEEE Advances in Digital Libraries (ADL 1998)*, pages 147–156, Santa Barbara, CA, 1998.

[50] I. Goldberg, D. Wagner, and E. Brewer. Privacy-enhancing technologies for the Internet. In *COMPCON '97: Proceedings of the 42nd IEEE International Computer Conference*, page 103, Washington, DC, USA, 1997. IEEE Computer Society.

[51] Ian Goldberg. Privacy-enhancing technologies for the Internet, II: Five years later. Workshop on Privacy Enhancing Technologies 2002, April 2002.

[52] Ian Goldberg and David Wagner. Taz servers and the rewebber network enabling anonymous publishing on the World Wide Web. *First Monday electronic journal*, 3(4):1–14, May 1997.

[53] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, 1999.

[54] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, pages 137–150, London, UK, 1996. Springer-Verlag.

[55] Yong Guan, Xinwen Fu, Riccardo Bettati, and Wei Zhao. An optimal strategy for anonymous communication protocols. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 257, Washington, DC, USA, 2002. IEEE Computer Society.

[56] Ceki Gülcü and Gene Tsudik. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96*, pages 2–16. IEEE, February 1996.

[57] P. B. Harris, C. M. Werner, B. Brown, and D. Ingebritsen. Relocation and privacy regulation: a cross-cultural analysis. *Journal of Environmental Psychology*, 15(3):311–320, September 1995.

[58] Sabine Helmers. A brief history of anon.penet.fi - the legendary anonymous remailer. `http://www.december.com/cmc/mag/1997/sep/helmers.html`, September 1997. Last accessed 29 January 2006.

[59] Johan Helsingius. Johan Helsingius closes his Internet remailer. `http://www.fitug.de/news/1997/penet.html`, 30 August 1996. Last accessed 8 September 2005.

[60] IBM. Privacy in a connected world. White paper, IBM, May 2002.

[61] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: a decentralized peer-to-peer Web cache. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 213–222, Monterey, California, 2002. ACM Press.

[62] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, El Paso, Texas, USA, 1997. ACM Press.

[63] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of Information Hiding Workshop (IH 1998)*. Springer-Verlag, LNCS 1525, 1998.

[64] David M. Kristol. Http cookies: Standards, privacy and polititcs. *ACM Trans. Inter. Tech.*, 1(2):151–198, November 2001.

[65] Frans A Lategan and Martin S Olivier. PrivGuard: A model to protect private information based on its usage. *South African Computer Journal*, 29:58–68, 2002.

[66] Shelly Lowery. Google adsense rewards content with advertising revenue. `http://www.web-source.net/google_adsense.htm`. Last accessed 9 April 2006.

[67] Salvador Mandujano and Clay Shields. Confidentiality and anonymity analysis of on-line payment protocols. In *Congreso del Dìa Internacinal de la Seguridad en Cómputo*, pages 45–52, Mexico City, Mexico, November 2000.

[68] Aviel D. Rubin Marc Waldman and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, Web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, Denver, Colorado, USA, August 2000. USENIX.

[69] Stephen T. Margulis. Privacy as a behavioral phenomenon. In *Man-Environment Interactions: evaluations and applications, Part II*. Hutchinson and Ross, 1974.

[70] Stephen T. Margulis. Privacy as a social issue and behavioral concept. *Journal of Social Issues*, 59(2):243–261, July 2003.

[71] David Martin, Hailin Wu, and Adil Alsaid. Hidden surveillance by web sites: Web bugs in contemporary use. *Commun. ACM*, 46(12):258–264, 2003.

[72] David Mazières and M. Frans Kaashoek. The Design, Implementation and Operation of an Email Pseudonym Server. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS 1998)*. ACM Press, November 1998.

[73] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003.

[74] Mojo Nation. Mojo nation technical overview. `http://surfvi.com/simota/others-papers/MojoNation_TechnialOverview.html`, February 2000. Last accessed 22 January 2006.

[75] Patricia B. Newell. A systems model of privacy. *Journal of Environmental Psychology*, 14:65–78, 1994.

[76] OECD. Oecd guidelines on the protection of privacy and transborder flows of personal data. Technical report, Organisation for Economic Co-operation and Development, September 1980. Last accessed 2 April 2006.

[77] Martin S Olivier. Database privacy. *SIGKDD Explorations*, 4(2):20–27, 2003.

[78] Martin S Olivier. Flocks: Distributed proxies for browsing privacy. In Gary Marsden, Paula Kotzé, and Ayodele Adesina-Ojo, editors, *Proceedings of SAICSIT 2004 — fulfilling the promise of ICT*, pages 79–88, Stellenbosch, South Africa, October 2004. South African Institute for Computer Scientists and Information Technologists.

[79] Martin S Olivier. Distributed proxies for browsing privacy — a simulation of Flocks. In Judith Bishop and Derrick G Kourie, editors, *Research for a changing world — Proceedings of SAICSIT 2005*, pages 104–112, White River, South Africa, September 2005.

[80] Rolf Oppliger. Privacy protection and anonymity services for the World Wide Web (WWW). *Future Generation Computer Systems*, 16:379–391, February 2000.

[81] Sameer Parekh. Prospects for remailers. *First Monday*, 1(2), August 1996.

[82] A Pfitzmann and M Waidner. Networks without user observability. *Computer Secururity*, 6(2):158–166, 1987.

[83] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudeonymity - a proposal for terminology. In *International workshop on Designing privacy enhancing technologies*, pages 1–9, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[84] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems*, pages 451–463, February 1991.

[85] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, Newport, Rhode Island, USA, 1997. ACM Press.

[86] Sandro Rafaeli, Marc Rennhard, Laurent Mathy, Bernhard Plattner, and David Hutchison. An architecture for pseudonymous e-commerce. In *Proc. of the AISB'01 Symposium on Information Agents for Electronic Commerce*, pages 33–42, York, United Kingdom, 21-24 March 2001. AISB.

[87] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, San Diego, California, United States, 2001. ACM Press.

[88] Jean-François Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29, Berkeley, CA, USA, July 2000. Springer-Verlag, LNCS 2009.

[89] Joseph Reagle and Lorrie Faith Cranor. The platform for privacy preferences. *Commun. ACM*, 42(2):48–55, 1999.

[90] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[91] Michael K. Reiter and Aviel D. Rubin. Anonymous Web transactions with crowds. *Commun. ACM*, 42(2):32–48, 1999.

[92] Marc Rennhard and Bernhard Plattner. Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 91–102, New York, NY, USA, 2002. ACM Press.

[93] RFC793. Transmission control protocol, September 1981. DARPA Internet Program Protocol Specification.

[94] USA Rice University, Houston. Freepastry version 1.4.4. `freepastry.org`, October 2006. Last accessed 10 October 2006.

[95] Andreas Rieke and Thomas Demuth. Janus: Server anonymity in the World Wide Web. In *Conference Proceedings EICAR International Conference*, pages 195–208. U. E. Gattiker, 2001.

[96] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, Banff, Alberta, Canada, 2001. ACM Press.

[97] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350. Springer-Verlag, 2001.

[98] Mark Sakalosky. Doubleclick's double edge. `http://www.clickz.com/experts/crm/analyze_data/article.php/1455141`, September 2002. Last accessed 3 April 2006.

[99] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[100] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In Fabien Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.

[101] Andrei Serjantov and Richard E. Newman. On the anonymity of timed pool mixes. In *Proceedings of the Workshop on Privacy and Anonymity Issues in Networked and Distributed Systems*, pages 427–434. Kluwer, May 2003.

[102] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In *Proceedings of ESORICS 2003*, October 2003.

[103] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423,623–656, 1948.

[104] Clay Shirky. What is P2P and what it isn't. `http://www.openp2p.com/pub/a/p2p/shirky1-whatisp2p.html`, November 2000. Last accessed 2 February 2006.

[105] K. Simonsen. Character mnemonics and character sets, June 1992.

[106] W. Richard Stevens. *TCP/IP illustrated (vol. 1): the protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[107] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[108] Moritz Strasser and Alf Zugenmaier. Personalization through mask marketing. *hicss*, 07:219b, 2003.

[109] Hongfei Sui, Jianxin Wang, Jianer Chen, and Songqiao Chen. The cost of becoming anonymous: on the participant payload in crowds. *Inf. Process. Lett.*, 90(2):81–86, 2004.

[110] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. *IEEE journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[111] H. Tillwick, T. Neumann, M.S. Olivier, H.S. Venter, and J.H.P Eloff. Polar: Proxies collaborating to achieve anonymous Web browsing. In *Proceedings of the Fifth International Network Conference (INC2005)*, pages 317–324, Samos, Greece, July 2005. SM Furnell, PS Dowland and G Kormentzas (eds).

[112] H. Tillwick and M.S. Olivier. Towards a framework for connection anonymity. In *Research for a changing world Proceedings of SAICSIT 2005*, pages 113–122, White River, South Africa, September 2005. J Bishop and DG Kourie (eds).

[113] Chonggang Wang and Bo Li. Peer-to-peer overlay networks: A survey, April 2003.

[114] Joss Wright and Susan Stepney. The many faces of anonymity - characterising anonymity systems, dec 2004.

[115] Joss Wright, Susan Stepney, John A. Clark, and Jeremy L. Jacob. Designing anonymity - a formal basis for identity hiding. Internal yellow report, York University, York, UK, dec 2004.

[116] Joss Wright, Susan Stepney, John A. Clark, and Jeremy L. Jacob. Designing Anonymity: An Overview. Technical report, York University, York, UK, February 2005.

[117] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. Defending anonymous communication against passive logging attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[118] Matthew K. Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Trans. Inf. Syst. Secur.*, 7(4):489–522, 2004.

[119] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, University of California at Berkeley, 2001.