APES deliverable 5
Tools for Technologies and Applications

Claudia Díaz, Vincent Naessens, Joris Claessens, Bart De Win,
Stefaan Seys, Bart De Decker and Bart Preneel

November 19, 2002
Executive summary

Anonymity and privacy are of paramount importance within the various electronic services provided in today’s expanding digital society. During previous work the anonymity requirements and properties of a wide range of applications have been analysed, and basic building blocks were distilled from existing solutions for anonymity. In this deliverable the appropriate privacy-enhancing technologies are chosen and incorporated in two different applications: privacy-preserving targeted advertising through web banners, and anonymous peer-to-peer networking. New tools and technologies are also presented.

A methodology to provide anonymity services is described. Ideally, the choice and composition of building blocks should be the output of an automatic algorithm that has a set of requirements as its input. The overall formalism needed in this approach is however still lacking and requires a substantial amount of research in the future. A more pragmatic approach is therefore followed.

A solution for privacy-preserving targeted advertising through web banners is proposed. The solutions allows users to make a balance between the exposure of their privacy and the personalization of the advertisement. The general idea of the solution lies in dynamically associating users with profiles according to their interests and/or demographics instead of to individual identifiers. In addition a number of security enhancements are suggested. The solution relies on an infrastructure for anonymous communication.

An architecture for anonymous P2P networking is proposed. The architecture is application-independent and is independent of the P2P model. The architecture separates peer-level and connection-level services and hides the implementation of anonymity functionality.

Anonymity is not a black-or-white issue. A model to measure the degree of anonymity is therefore developed. A specific model for applications such as the web banner system, as well as a generic model for anonymous communication, is presented. The degree of anonymity depends on the probabilities of having sent a message, and is measured with respect to a particular attacker.

A mix forms the core of an anonymous communication infrastructure. A new theoretical mix design is proposed that uses randomness in order to make message tracing more difficult, and that provides better resistance against the blending attack.

Finally, a proof-of-concept of both the web banner application and the P2P architecture is demonstrated.
Contents

1 Introduction 5

2 Methodology to select technologies 6
  2.1 Analysis ................................................................. 6
  2.1.1 A generic approach .................................................. 6
  2.1.2 A pragmatic approach ................................................. 9
  2.2 Tools ................................................................. 11
  2.2.1 Verification Tools .................................................. 11
  2.2.2 Composition Tools ............................................... 12

3 Applications: concepts and architectures 13
  3.1 Privacy-preserving web banners for targeted advertising ........ 13
    3.1.1 Web banners for targeted advertising .......................... 13
    3.1.2 Privacy-preserving web banners .................................. 15
    3.1.3 Enhancements ........................................................ 18
  3.2 Peer-to-Peer (P2P) .................................................... 19
    3.2.1 Taxonomy of P2P systems ......................................... 20
    3.2.2 Taxonomy of P2P models .......................................... 21
    3.2.3 Design goals ....................................................... 23
    3.2.4 Architecture ....................................................... 23
    3.2.5 Comparison with other systems .................................... 25

4 Developed tools, technologies and building blocks 27
  4.1 Model to measure the anonymity provided by a system. ............ 27
    4.1.1 System model ..................................................... 28
    4.1.2 Proposed measurement model .................................... 29
    4.1.3 Attack models ..................................................... 31
    4.1.4 Practical case: targeted advertising ............................ 31
4.2 Model to measure the anonymity provided by an anonymous communication system ............................................. 33
  4.2.1 Introduction ................................................................. 33
  4.2.2 System model ............................................................... 34
  4.2.3 Proposed measurement model .......................................... 36
  4.2.4 Degree of anonymity provided by the system ....................... 36
  4.2.5 Measuring the degree of anonymity provided by some systems ......................................................... 38
  4.2.6 Crowds ........................................................................ 40
  4.2.7 Onion Routing ............................................................... 44

4.3 The binomial mix .................................................................. 47
  4.3.1 Communication layer ...................................................... 47
  4.3.2 Mixes ............................................................................ 47
  4.3.3 The blending attack .......................................................... 48
  4.3.4 Description of the binomial mix ......................................... 48
  4.3.5 Guessing the number of messages contained in the mix ......... 50
  4.3.6 The blending attack on the binomial mix ......................... 51
  4.3.7 Average delay of a message .............................................. 54
  4.3.8 Additional measure: Timestamps ....................................... 54

4.4 Building block for Targeted Advertising .................................. 56
  4.4.1 Description ...................................................................... 56

4.5 Building blocks for P2P .......................................................... 57
  4.5.1 Anonymity library ............................................................ 57
  4.5.2 Identifiers and Locators .................................................... 58
  4.5.3 Peer level blocks ............................................................. 59
  4.5.4 Connection level blocks ................................................... 68

5 Detailed description of demonstrators ........................................ 70
  5.1 Targeted Advertising ............................................................ 70
  5.1.1 Web pages ...................................................................... 70
  5.1.2 Client ............................................................................. 71
  5.1.3 Banner Server ................................................................. 74

  5.2 Peer to peer (P2P) ............................................................... 75
  5.2.1 Overview of the application .............................................. 75
  5.2.2 Registration of a node ..................................................... 77
  5.2.3 Setting up a chat session .................................................. 77
  5.2.4 Subscribing to a session ................................................... 77
  5.2.5 Exchanging messages between nodes ................................. 78

  5.2.6 Storage of information in the peer system ......................... 78
6 Conclusions and future work

6.1 Conclusions ................................................. 80
6.2 Future work .................................................. 81
   6.2.1 Methodology ............................................. 81
   6.2.2 Privacy-preserving targeted advertising system .... 81
   6.2.3 Peer-to-peer networks ................................. 82
Chapter 1

Introduction

In deliverable 2 [D2] of this project, we analysed the anonymity requirements and properties of a wide range of applications. In deliverable 3 [D3] basic building blocks were distilled from existing solutions for anonymity. These generic components were examined, evaluated and structured. In this deliverable we return to the point of view of the application.

The appropriate privacy enhancing technologies described in deliverable 3 are chosen and incorporated in two different applications, namely privacy-preserving targeted advertising through web banners, and anonymous peer-to-peer networking. Specific tailor-made solutions are developed in order to cover all privacy requirements which cannot be solved by generic solutions. A methodology to provide anonymity services is described and followed. A method to evaluate to what extent anonymity is achieved is also presented.

In addition to this deliverable, a toolkit is implemented consisting of the individual building blocks that are needed by the two selected applications. These building blocks are integrated in two respective demonstrators as a proof of concept.

This deliverable is structured as follows. Chapter 2 discusses the methodology that is used to select particular privacy enhancing technologies in order to provide anonymity within a certain application. Chapter 3 then describes the two chosen applications in which anonymity is added. The concepts and architecture of the respective applications are presented. A number of new tools, technologies and building blocks have been developed for this purpose. These are discussed in Chapter 4. A more detailed description of the respective demonstrators is given in Chapter 5. Lastly, Chapter 6 gives a conclusion and outlines future work.
Chapter 2

Methodology to select technologies

In this chapter, we describe how different combinations of building blocks as defined in this and previous deliverables should be composed to meet a particular anonymity requirement. Since there is no practical elaboration on this matter yet, we will tackle the problem from a more theoretical viewpoint. Hereby, we first describe the general view we have on the solution to this problem. In a more pragmatic section, we then give some examples of how the different anonymity and other requirements could lead to the selection of particular building blocks. As such, we hope to sketch a very rough, but more concrete view on the generic approach of this methodology and at the same time give some examples of the rules that could serve as its input. Finally, we discuss several tools that will be required to enable the approach.

2.1 Analysis

2.1.1 A generic approach

From a generic point of view, the ideal approach to technology or building block selection is to have an automatic selection process (or algorithm). As input, this algorithm requires (a set of) user requirements. These user requirements are typically system-wide and they are described in a high-level, user understandable manner. For instance, a example user requirement could express the desire to have beyond suspicion sender anonymity in the system at all times. Besides these user requirements, other constraints that
are not specified by the user, such as specific block properties, are also part of the algorithm input. The output of the algorithm then consists of the description of an optimal\(^1\) technology/block composition.

Unfortunately, this generic approach to technology selection is very complex, for which we have not come up a satisfying solution yet. In this section, we will therefore describe a non-exhaustive list of issues that could be considered as input of the algorithm. Hereby, we distinguish between specific choices of the user and other properties that are more inherent for the technologies used.

**User specific requirements**

- Given a certain anonymity model, the user must first specify the anonymity requirements he wants to have enforced within the system. The options for the user evidently depend on the specific anonymity model used for this purpose. For the anonymity model described in deliverable ??, the user should specify the degree of anonymity and the role (sender, receiver, ...) for which it should be enforced.

- The user should explicitly specify the attack model for which the system should provide anonymity. For instance, a local attacker that can only eavesdrop communication on a local link involves less anonymity functionality compared to a global attacker that can eavesdrop on and modify all communication. In our opinion (see also deliverable ??), the following list of properties is important with regards to the attack model:

  - external vs. internal attackers. Hereby, an internal attacker is a party that executes as part of the system and tries to attack the system as such, while an external attacker is not part of the system as such.

  - active vs. passive attackers. A passive attacker will only eavesdrop and monitor communication within the system, while an active attacker will possibly also alter communication by modifying existing messages or even introducing or deleting messages.

  - single vs. colluding attackers. Single attackers will attack the system using only their view of the system, while colluding attackers may cooperate to exchange information to broaden their view.

\(^1\)Optimality is a generic term and the concrete meaning of the term can vary between execution time, execution complexity, minimization of communication, etc. In order to avoid overloading this discussion, we will make abstraction of this term.
• Specific hardware infrastructure and information on the particular network topology could influence the required strength of anonymity and hence also the selection of building blocks. For instance, when a connection actually consists of a controlled and a non-controlled part, blocks can be selected only to secure the non-controlled part. In general, this information will influence and can be used to improve the efficiency of the composed system, since a worst case scenario will always suffice for better scenario's too.

• A last issue concerns the fact that similar functionality can often be represented at different layers. For instance, encryption and fragmentation of communication can both occur at communication layer as at peer layer. For this purpose, the user could specify certain preferences.

Technology specific requirements

• Every building block has a certain effect within the light of anonymity. To enable checking the fulfillment of particular anonymity requirement through the selection of a particular building block, the particular block properties should be known for each building block. The representation of these block properties could include as well a functional description as a more high-level description of its purpose.

• Besides isolated block properties, dependencies might exist between different blocks. Dependencies address crucial relationships between blocks, such as the fact that a push block, disregarding its exact functionality, always requires a communication block. Dependencies relate with the core functionality (or implementation) of the block and can hence be identified by the block implementors. Besides dependencies, certain orderings can be specified for a specific selection of blocks. For instance, suppose an encryption and a fragmentation block is selected, should communication be first splitted and afterwards encrypted or the other way around? Orderings do not jeopardize the core execution of a set of building blocks, but they are required for multi party synchronization and understandability.

• Certain technologies consist of many separate parts or building blocks and a lack of one or more blocks renders the technology useless. For instance, all P2P technologies require a way to communicate with different peers in the system. As will shown later, we introduce generic communication blocks (push and pull) for this purpose. Now, when requesting a P2P execution environment, the absence of push and
pull blocks is crucial. Without them, the P2P system will not function properly. As illustrated by this example, the overall structure of certain technologies imposes demands and constraints on the block selection algorithm. These overall structures must be identified and fed to the algorithm. Remark that these structures are more apparent on higher-level layers where different types of blocks must cooperate to reach a higher goal. In that sense, communication layer blocks will rarely be bound by such structures.

For a practical elaboration of this approach, we see two major problems at this moment. A first problem is the large gap between user level requirement specification and its technical representation to be fed to the algorithm. Anonymity requirements should be specified in a user comprehensible way, such that he can clearly see whether his requirements really match the rules he writes down. For instance, the user wants to reason in terms of anonymity properties of communication, not in terms of specific strategies for push and pull blocks. Unfortunately, the technical translation of such requirements into block selection properties is far from trivial. A second problem is the overall architecture to capture and represent the list of above requirements. Probably, this requires properties per block (such as its dependencies, specific anonymity achievements, etc.) and more overall properties. For this problem, the structure, the formalism and the exact values to be used are not clear to us. We think that a generic approach is important, but it will require a lot of research in the future.

2.1.2 A pragmatic approach

In this section, we discuss three concrete examples for which we each time change the environment (and as such the input requirements) slightly to demonstrate the change in selected building blocks. In order to fully grasp the line of thought we try to sketch here, we advise the reader to read all examples first once in batch and then revisit the examples one by one as to find out why we choose each strategy (explicitly in- and excluding particular building blocks).

More concretely, all the examples relate to the enforcement of sender anonymity, for which we chose to focus on anonymity techniques at the communication level. Through the examples, we also suppose a passive attacker model that does hence not change the communication.

Example 1 In a first example, we suppose that the communication environment is protected and can as such not be eavesdropped. Furthermore, all
the hosts residing on the network are controlled in a centralized manner such that they can be trusted not to attack the system or spread any execution information about it.

In this type of system, sender anonymity can be achieved by the use of (at least) 1 forward block, similar to a Crowds-like system. There are no specific requirements as to which host should or cannot play the role of this forward block. Even the sender of the message can act as the forwarder. As long as this is not the only or primary communication scenario, the anonymity of the sender will not be compromised. Remark that this scenario is quite trivial to implement: either the sender sends the message directly to the recipient (in which case he might act as a forwarder), or the sender sends the message to another host, which will act as a forwarder. This host can then decide to forward the message to the recipient or to yet another host. So actually, the sender can control in some sense the degree of anonymity for this messages.

**Example 2** In this example, we retain the protection of the communication environment as in the previous example. However, we do allow hosts in the network to collaborate as to recover the senders’ identity.

Similar to the previous example, we can use one trusted forward block. However, in this case the sender itself cannot play the role of trusted forwarder, since the collaboration of all other (non-trusted) mixes in the network will jeopardize the anonymity requirement. We hence require the use of one external forwarder. Moreover, since not all hosts can be trusted, we must make sure to enforce the use of a trusted host within the communication path. Simply sending the message to a random host is clearly not sufficient in this case, since it might not be a trusted one. As a remedy, we could impose the use of an onion like structure to control the communication path used for the message. This onion must be created by the sender of the message and he must make sure to include the trusted host in the path.

To conclude, the infrastructure used in this example and in the previous one is very similar, but a change in requirements necessitates the introduction of an extra technique, the use of an onion. Remark that a Crowds like approach can also be used in this example, given that at least one external, or hence two forwarders in general are used for the communication.

**Example 3** For the last example, we also drop the requirement that the communication environment is protected. This means that links -and thus messages travelling on them- can be eavesdropped and acted upon.
Similar to example 2, we require in this case the use of at least one external, trusted forward block. Furthermore, since communication can be eavesdropped and messages can hence be followed on the basis of their content. Therefore, the messages must be encrypted and the encryption must be changed between the different forwarders (otherwise the content can still be followed). We hence require some kind of full onion-like system for which at least 1 trusted host is part of the communication channel or for which the path length is higher than the number of possibly concurrently colluding forwarders.

Through the description of these examples, several high-level rules can already be distilled that could help an algorithm to select building blocks. As we see it, a similar, but more extended process will serve as a kick off to the gathering of the different types of input as described in the previous section.

2.2 Tools

Designing an infrastructure for anonymous communication or storage/retrieval of data, is not a sinecure. Tools are necessary both to assist the design of the infrastructure and to verify its correctness.

Tools can be categorized in two major classes:

- **verification tools** prove that the chosen composition complies with the anonymity requirements;
- **composition tools** assist the designer with the selection of building blocks, the proper composition of these blocks and the deployment of the components over the available nodes.

The inputs for these tools are described in the "generic approach" section.

2.2.1 Verification Tools

Verification tools check the correctness of an implementation towards the anonymity requirements. The designer will have to compose the anonymity infrastructure manually, while the tool will formally prove that certain criteria are met. Hence, a formal logic for reasoning in that domain should be developed. Every building block needs to be assigned properties expressed in that logic.
2.2.2 Composition Tools

Composition tools help the designer select the basic building blocks, propose the proper composition of these building blocks into components and assist with the deployment of the components over the available nodes.

- a very elementary tool would present a set of (high level) solutions from which the designer will select some;

- a sophisticated tool would select and compose the building blocks, based on the anonymity requirements, attack & trust model, hardware infrastructure (network and platforms), properties and dependencies of the building blocks, and the overall structure. It should be clear that such a tool include a verifier in order to propose only correct designs. It is most likely that there is no unique solution. The tool should suggest alternatives or select one depending on secondary criteria: resources of the platforms, ...
Chapter 3

Applications: concepts and architectures

In this chapter we discuss the two applications that we have chosen. We outline the concepts and architecture of the privacy preserving targeted advertising system in section 3.1, and those of the peer-to-peer system in section 3.2.

3.1 Privacy-preserving web banners for targeted advertising

Here, we outline the privacy-preserving web banner application for targeted advertising, that has been developed in the scope of the project. The concept and issues of web banners are first described. We then present a system for privacy-preserving web banners. Last but not least, further enhancements of the system are proposed.

3.1.1 Web banners for targeted advertising

Model and participants

In a web banner system, there are basically 4 different parties:

Users. Users surf to web sites. The web pages include banners with advertisements that are related to that page and potentially to the personal interests of the user.
Web sites. Web sites want to include banners in their pages, in order to receive money from the Banner Server, per banner that is shown to the users through their pages. In addition, banners that are interesting for the user might make the web site more attractive.

Banner Server. The Banner Server's business model is to make money by serving banners to the appropriate category of users. Tracking of users is desired for choosing the appropriate banner (e.g., the same banner should be shown 3-7 consecutive times), and for marketing reports.

Banner Payer. Banner Payers will pay for the service offered by the Banner Server. Via the banners, they hope to attract users to their web sites, products and services.

Technical issues

Some technical issues regarding web banner systems are the following:

- Banner requests are associated with a unique ID. This can be done based on the IP address of the user and/or based on storing a unique ID in the cookie that is delivered to the user together with the banner.

- The unique ID is associated with a (user) profile, a database which contains information related to the user's identity and actions.

- The banner server receives requests, updates the profile according to the current request, and sends back the appropriate banner.

- There are two ways of inserting banners:

  - "invasive": the web site includes a link to the banner in the web page; the link possibly contains extra information regarding the category of the page; the actual banner request is directly performed by the user's browser.

  - "automatic insertion of code at server": banners are transparently added to the web pages at the server side, before sending the pages to the user; the actual banner request is performed by the server, and the banner is sent to the user together with the page.
Privacy and security problems

Current web banner systems have some privacy and security problems:

Privacy. Banner requests are usually associated with a unique identifier corresponding to individual users. Banner servers obtain an incredible amount of personal information with which they can build up complete profiles of the users’ activities on the web. This is (should be) an important concern for the users.

Security. Banner server and/or web sites possibly want to cheat in order to gain more money. For example, malicious web sites could perform numerous requests of their own web pages, so that it seems that many users received banners through these pages. The web sites can then claim more money from the banner server. The banner server could cooperate itself, and claim more money from the banner payers.

Privacy is an important concern of the users. Security is a concern of the other three parties. The solution that is presented in this document mainly intends to address the privacy issue.

3.1.2 Privacy-preserving web banners

A trivial solution for guaranteeing a user’s privacy is simply blocking banners (and cookies). However, we assume that users are potentially interested in receiving banners with advertisement of their interest. Obviously, banners can be made more personalized if more personal information is exposed. The privacy of the user should however still be guaranteed. More precisely, users should be able to make a balance between the exposure of their privacy and the personalization of the advertisement.

Concept

The general idea of the solution presented here consists of associating users with profiles according to their interests and/or demographics instead of to individual identifiers. In other words, individual users are not directly associated with a unique profile, but can (dynamically) belong to a group of users that are associated with the same profile. In this way, users do not expose their individual actions on the web, but can hide within the group of users with the same profile.
Figure 3.1 illustrates the proposed privacy-preserving web banner system. The four different parties and the interactions between them are indicated. The web banner system consists of the following steps:

1. The user requests a web page from a particular web site.

2. The web site delivers the requested web page. The web page contains a link to a banner. It is possible for the web site to include extra information about the web page (e.g., the interest category) with the link.

3. Upon receipt of the web page, the user’s browser will automatically request a banner from the banner server. The request includes the user’s profile (in a cookie) and the referring address of the web page (‘referer’ header).

4. Based on the user’s profile, the address of the web page, and the extra information provided by the web site, the banner server chooses the appropriate banner and sends it back to the user. Together with the banner, the banner server will send information with which the user’s profile can be updated.

5. The banner is shown to the user. The user’s profile is updated, i.e., the user’s next request will possibly include another profile associated with a different group of users.

6. At some point in time, the banner server will receive money from the banner payer for having served the banner. The web site will receive its share of the money.

All requests are performed over an anonymous network, as otherwise the user’s IP address is disclosed. The user’s IP address constitutes information that potentially discloses the users’ individual identity.

Locally on the user’s machine, software is running which acts as a proxy in between the user’s browser and the web sites and banner server. The proxy takes care of the user’s profile, and is also the access point to the anonymous network.

**Information theory and the degree of anonymity**

In Chapter 4, we propose a model with which the degree of anonymity can be quantified in applications in which users are divided into groups in
1 User requests web page
2 Web site delivers page; the page contains link to banner and possibly more info about category
3 User transparently requests banner; the request includes the user’s profile
4 Banner Server serves banner; information for updating the profile is included together with the banner
5 The banner is shown to the User; the user’s profile is updated
6 Banner Payer pays Banner Server; Web site receives part of the money

Figure 3.1: The privacy-preserving web banner system
which they are indistinguishable from each other. This model is particularly suited for measuring the quality of our solution. The model will help us to define appropriate user profiles, i.e., user profiles that allow a proper balance between targeted, personalized advertising, and exposure of individual privacy. In other words, the more profiles, the more functionality, while the more users with the same profile, the more privacy.

**Client-side profile update**

The profile update is based on the complete history of the user's actions. For privacy reasons, the update should thus obviously be performed at the client side.

As a consequence, the solution for privacy-preserving web banners might seem more suitable for the "invasive" way of inserting banners, as is the case in the description above. It should however also be possible to use the solution in the "automatic insertion" way.

**Server-side vs. client-side measurement**

In the current systems the appropriate banner is chosen by the banner server (i.e., server-side measurement). Juels [Jue01] proposed a solution in which the banner is chosen by client software (i.e., client-side measurement) and requested through a mix network. Our solution has both client-side and server-side properties. On the server-side, an appropriate banner is chosen corresponding to a particular profile. On the client-side, the appropriate profile is maintained and chosen by the user. The anonymous communication network in our solution is the equivalent of the mix network in the solution of Juels.

**3.1.3 Enhancements**

The privacy-preserving web banner system as described above, can be enhanced to improve security and functionality.

**Security**

Malicious parties could perform multiple fake banner requests in order to get more money. Current banner systems try to tackle this by for example checking if there are not too many requests from the same IP address, etc. Thus, some fraud detection is already included in these systems. Although these techniques are relatively easy to circumvent, they seem to be good
enough for current systems. In our system, the problem is more difficult
to solve, as for example the anonymous connections prevent to check the
originating IP address.

A more secure solution that works in the anonymous case and would
also be an improvement in the current systems, consists of making multiple
fake banner requests difficult in the first place, instead of (or in addition to)
having to check against it afterwards. For each banner request, some blinded
ticket could be required (e.g., ecash but with no financial value), or certain
‘client puzzles’ could be required to be solved (e.g., finding a collision of a
hash function). In the case of the blinded ticket, only a limited amount of
tickets would be issued to each user, enough for normal usage, but certainly
not allowing large-scale fraud. In the case of the client puzzles, the user
should solve some computationally intensive problem, which is easy enough
to allow normal browsing (e.g., one banner per 10 seconds), but which does
again not allow large-scale fraud. Note that this issue is related to Denial-
of-Service attacks.

**Intermediate banner service**

The Banner Server could be an intermediate party requesting banners to real
banner servers on behalf of the user. The Banner Server’s task is to map
the profile of the user to the unique ID required by the particular existing
banner system. The Banner Server is then an intermediate service providing
privacy-preserving banners to users. See also [Enc].

The advantage of this approach is that existing banner systems do not
have to adapt their internal system to be able to work with profiles instead
of unique IDs, even if they want to adopt privacy-preserving techniques.
The intermediate banner server does not have to be trusted but to map the
profile of the user to a corresponding relevant unique ID.

From a technical point of view, an intermediate banner server implies
that the client proxy program must re-route banner requests to the inter-
mediate server.

**3.2 Peer-to-Peer (P2P)**

A peer-to-peer (P2P) system is a distributed, often dynamic, architecture
where many identical participants (peers) cooperate using a decentralized
paradigm in order to achieve a common goal. The symmetric relationship
between the participating computers distinguishes this architecture from
other, typically asymmetric systems, like client-server.
The goals (and advantages) of these architectures are multifaceted. The most important goal of distributed systems is resource sharing. In P2P systems, shared resources typically consist of hardware (CPU, ...), software (apps, files, ...), and other information. Another goal, and probably the most important reason for their existence, is their inherent potential to provide anonymity. Due to the peer-to-peer organization and due to the lack of fixed dependencies, it is possible to distribute the trust in the system. As such, different anonymity characteristics can be achieved, for instance sender/receiver anonymity, content anonymity (storing/retrieving), query anonymity, etc. Clearly, the exact anonymity characteristics depend on the specific system.

The biggest disadvantage of P2P systems is the fact that a peer often does not know the true identity and the trustworthiness of other peers. This sometimes gives attackers the opportunity to misbehave and abuse the system, although more recently techniques are available to prevent this.

P2P is used as a case study to test a toolbox that provides anonymity services. P2P is chosen because of its nature: most of the anonymity services in the toolbox concentrate on anonymous connections, anonymous storage, anonymous information exchange, etc. This means that complex protocols for application specific aims such as voting protocols, auction protocols are not in the scope of the toolbox. The toolbox concentrates on more general services that can be reused in different applications.

3.2.1 Taxonomy of P2P systems

Distributed computing Distributed Computing achieves processing scalability by aggregating the resources of a large number of individual Internet PCs. Typically, distributed computing requires applications that are run in a proprietary way by a central controller. Such applications usually target massive multi-parameters systems, with long running jobs (months or years) using P2P foundations. One of the first widely visible distributed computing events occurred in January 1999, where distributed.net, with the help of several tens of thousands of Internet computers, broke the RSA challenge in less than 24 hours using a distributed computing approach. This made people realize how much power can be available from idle Internet PCs. Examples of such systems are Beowulf [BSS+95], MOSIX [BL85] and Condor.
File sharing  Content storage and exchange is one of the areas where P2P technology has been most successful. Multimedia content, for instance, inherently requires large files. Napster [Nap] and Gnutella [Kan01] have been used by Internet users to circumvent bandwidth limitations that make large file transfers unacceptable with classic mechanisms. Other examples are Oceanstore [KBC+00], Chord [DBK+01] [SMK+01] and Publius [WRC00].

Collaboration  Collaborative P2P applications aim at application level collaboration between users. The inherent ad-hoc nature of P2P technology makes it a good fit for user-level collaborative applications. These applications range from instant messaging and chat, to online games, and shared applications that can be used in business, educational and home environments. Examples are Kazaa [Kaz] and Groove [Gro].

Platforms  Operating systems are becoming increasingly relevant as environments for applications. Middleware solutions [Mic01] [JXT], such as Java Virtual Machines, or Web browsers and servers are the dominant environment that is of interest to users as well as to developers of applications. In that regard, it is likely that future systems will increasingly depend on some other sort of platform that will be a common denominator for users and services connected to the Web or in an ad-hoc network.

3.2.2 Taxonomy of P2P models

This section overviews three common P2P algorithms and then compares their implementations in a few P2P systems.

Centralized directory model.  This model was made popular by Napster [Nap]. The peers of the community connect to a central directory where they publish information about the content they offer for sharing. Upon request from a peer, the central index will match the request with the best peer in its directory that matches the request. The best peer could be the one that is cheapest, fastest, or the most available, depending on the user's needs. Then a file exchange will occur directly between the two peers. This model requires some managed infrastructure (the directory server), which hosts information about all participants in the community. This can cause the model to show scalability limits, because it requires faster servers when
the number of requests increases, and larger storage when the number of users increases. However, Napster’s experience shows that - except for legal issues - this model is very strong and efficient.

**Flooded requests model.** The flooding model is a pure P2P model which requires no advertisement of shared resources. Instead, each request from a peer is flooded (broadcast) to directly connected peers, which themselves flood their peers etc., until the request is answered or a maximum number of flooding steps (typically 5 to 9) is reached. This model, for instance used by Gnutella [Kan01], requires a lot of network bandwidth, and hence does not prove to be very scalable, but it is efficient in limited communities such as a company network. To circumvent this problem, some companies have been developing super-peer client software, that concentrates lots of the requests. This leads to much lower network bandwidth requirement, at the expense of high CPU consumption. Caching of recent search requests is also used to improve scalability.

**Document routing model.** The document routing model, used by FreeNet [Fre] [CSWH99], is the most recent approach. Each peer from the network is assigned a random ID and each peer also knows a given number of peers. When a document is published (shared) on such a system, an ID is assigned to the document based on a hash of the document’s contents and its name. Each peer will then route the document towards the peer with the ID that is most similar to the document ID. This process is repeated until the nearest peer ID is the current peer’s ID. Each routing operation also ensures that a local copy of the document is kept. When a peer requests the document from the P2P system, the request will go to the peer with the ID most similar to the document ID. This process is repeated until a copy of the document is found. Then the document is transferred back to the request originator, while each peer participating the routing will keep a local copy. Although the document routing model is very efficient for large, global communities, it has the problem that the document IDs must be known before posting a request for a given document. Hence it is more difficult to implement a search than in the flooded requests model. Also, network partitioning can lead to an islanding problem, where the community splits into independent sub-communities, that do not have links to each other.
3.2.3 Design goals

In this section, we describe the goals we want to achieve with the P2P architecture. These goals originate from the need for a flexible framework that covers a wide range of P2P application domains and infrastructures and in which anonymity services can easily be built in.

Application independence. As described in section 3.2.2, many different applications can be built using P2P systems. In order to enable reuse for different applications, the P2P architecture must be designed independent from the application. We provide a modular, extensible architecture without references from the P2P system towards the application and define a general, reusable API with general concepts and operations to capture different application domains.

P2P model independence. In section 3.2.2, we described different P2P models. The architecture must be flexible to support these models. We try to achieve this goal by defining reusable blocks. Only the implementation of some blocks will depend on the model that is used. Moreover, the application should not be aware of the P2P model.

Connection independence. The architecture should separate peer level services and connection level services. By separating these services, we try to achieve a more flexible and modular design.

Hiding anonymity functions. The implementation of basic algorithms for achieving anonymity (such as a reordering algorithm) should be hidden towards peer level blocks and connection level blocks that provide some anonymity service.

3.2.4 Architecture

As shown in the figure below, the architecture of the anonymous P2P system consists of three layers: the application layer, the peer layer and the connection layer. The peer layer provides an API that can be called by the application layer. The connection layer also provides an API that can be called by the peer layer. Besides that layered design, an anonymity library contains the basic algorithms for implementing anonymity services. Peer level blocks and connection level blocks make a composition of some basic algorithms in the anonymity library to meet the anonymity requirements.
- **The application layer.** This layer represents the application. The application can communicate with the peer layer by calling the methods described in the peer level API. The API must be general enough to cover different application domains.

- **The peer layer.** This layer contains building blocks that can be identified in most P2P systems. These blocks are defined in such a way that they can be used in most P2P platforms. Some of these building blocks provide methods that can be called by the application layer. These methods are described in the peer layer API. For instance, methods are provided to push application data to other nodes in the system and methods are provided to pull information from other nodes to the application. Other blocks are used internally.

- **The connection layer.** This layer contains different types of connections. They provide an interface to the peer layer. The methods are described in a connection level API. This interface can be used by peer layer building blocks. The interface at that level must be designed so that different types of connections can be supported such as
(anonymous) synchronous connections and (anonymous) asynchronous connections.

The anonymity library. This library contains basic blocks to achieve anonymity such as encryption, reordering of messages, splitting of data, etc. The API provided by this basic can be called by the peer layer blocks and the connection layer blocks. By providing an API, the peer layer and the connection layer consider the basic blocks as black boxes. If the peer layer wants to reorder data, it uses a reordering basic block by calling the reorder method from the API without worrying about the implementation of the reordering algorithm. Peer layer blocks and connection layer blocks should make a good composition of the basic building blocks to achieve the desired degree of anonymity in the system. If the connection layer support anonymous connections, many anonymity properties are realised in the connection layer and less should be provided at peer layer. On the other hand, if the connection layer does not support anonymous connections, the components at peer layer are fully responsible for hiding the identity of a peer towards other peers.

3.2.5 Comparison with other systems

In this section, we compare the architecture with three other systems. The goal of this comparison is twofold. First, we show that the architecture is flexible enough to insert or reimplement existing systems in order to use them for P2P systems.

Onion routing. Onion routing [Oni] is an example of an anonymous connection system. Proxies already exist for electronic mail and web browsing. If a proxy is written for P2P systems, onion routing can be inserted in the architecture at connection level.

CROWDS. CROWDS [RR97] is also an anonymous connection system. CROWDS is in fact only designed for web browsing but it can be used for general connections. Messages are forwarded to other crowd members. This system can be used at connection level if a node participates in the crowd. Remark that participants not only consist of members of the P2P system but also of other members all over the internet. The crowds system can also be reimplemented at peer layer. In that case, each node registers as a crowd member and messages are only forwarded between members of the peer system.
TARZAN. TARZAN [FSCM02] is a anonymous connection system that uses peers as mixes. The mixes vary dynamically as they register to the system. Each user can start up a mix. But users that don’t start up a node can also send messages through peers. So, each peer in the P2P system can start up a mix to participate in the TARZAN system.
Chapter 4

Developed tools, technologies and building blocks

This chapter describes the tools for anonymity that have been developed. It includes:

- Anonymity models: we present two information-theoretical anonymity models. One of them is useful to measure the anonymity at the application layer (section 4.1), and the other one (section 4.2) can be used to measure the anonymity provided by an anonymous connection system (i.e., a mix network).

- Mix design: in section 4.3 we propose a new mix design. Mixes are the basical building block to provide anonymity at the connection level.

- Application-level filter: One of the implemented building blocks is an HTTP filter, which is described in section 4.4.

- Several building blocks for peer-to-peer systems: section 4.5 describes several building blocks that compose a peer-to-peer system.

4.1 Model to measure the anonymity provided by a system.

This section proposes a model that uses information theory in order to give a measure of the degree of anonymity. The model is suited for systems in which users belong to groups, and users who belong to the same group are indistinguishable. The proposed model has been published in [DCSP02].
4.1.1 System model

We consider a number of users, who are distributed into groups. Each user is identified as a member of a group when he generates requests (e.g., requests for a web page). We explain the model below into more detail, and we set the notation.

Users. Let $N$ be the number of users, $u_1, \ldots, u_N$.

Groups. The users are distributed into groups. All the users that belong to a group are indistinguishable. Let $M$ be the number of groups, $g_1, \ldots, g_M$. We assume that the number of users, $N$, is much larger than the number of groups. Users may belong to different groups at different moments. Each group $g_i$ contains $N_i$ users. We can see that:

$$N = \sum_{i=1}^{M} N_i .$$

Requests. Each user $u_j$ generates $r_j$ requests. We call $R$ the total number of requests produced by the set of users in a certain amount of time. To identify a particular request, we use the notation $R_i, (R_1, \ldots, R_R)$:

$$R = \sum_{j=1}^{N} r_j .$$

$R_{g_k}$ denotes the number of requests that belong to the same group $g_k$.

Connection level. We assume that all users are connected through a mix network to achieve anonymity at the connection level. This mix network works for example like Onion Routing. The attacker is not able to perform traffic analysis within the mix network, but he may be able to analyze the input and the output of the mix network. We consider real time applications. For this reason, the delay of the request cannot be too big. The attacker will know that a request that comes out of the mix network has been recently generated. In this model, we focus on the anonymity at the data level, and do not analyze the level of anonymity at the connection level, i.e., we assume the mix network to perfectly hide the link between source and destination of a particular request.
4.1.2 Proposed measurement model

Section 4.2 proposes a model to measure the anonymity at the connection level. Here, we introduce a model with which the degree of anonymity of a set of users grouped in several subsets, as defined above, can be quantified in an objective way.

Definition of anonymity

First of all, we should give a precise definition of anonymity. As in previous deliverables, we adopt the definition given by Pfitzmann in [PK00]. Anonymity is the state of being not identifiable within a set of subjects, the anonymity set.

Definition of the degree of anonymity

According to the previous definition, in a system with \( N \) active users, the maximum degree of anonymity is achieved when an attacker sees all users equally probable as being the originator of a request. In our case, this situation is achieved when there is only one group that contains all the users, and the number of requests generated by each user is the same.

Therefore, in our model the degree of anonymity depends on the distribution of probabilities and not on the number of users. This way, we are able to measure the quality of the system with respect to the anonymity it provides, independently from the number of users who are actually using it. Nevertheless, note that the number of active users should be large enough in comparison with the number of groups, in order to ensure that there are no groups that contain a small number of users. If a group contains a single active user, this user is no longer anonymous.

The proposed model compares the information obtained by the attacker after observing the system against the optimal situation from the anonymity point of view, in which all users seem to be equally probable as being the originator of the message, that is, in a system with \( N \) users, the situation where all users belong to the same group and make the same number of requests.

After observing the system for a while, an attacker may assign some probabilities to each sender as being the originator of a message, based on the information he has stored. For a given distribution of probabilities, the concept of entropy in information theory provides a measure of the information contained in that distribution. We will use entropy as a tool to calculate the degree of anonymity achieved by the users.
The entropy of the system after the attack will be compared against the maximum entropy (for the same number of users). In this way we get an idea of how much information the attacker has gained, or, in other words, we compare how distinguishable the sender is within the set of possible senders after the attack.

Let $X$ be the discrete random variable with probability mass function $p_i = Pr(X = i)$, where $i$ represents each possible value that $X$ may take. In this case, each $i$ will correspond to a user $u_i$. We denote by $H(X)$ the entropy of the system after the attack has taken place. For each user $u_i$, the attacker will assign a probability $p_i$. $H(X)$ can be calculated as:

$$H(X) = - \sum_{i=1}^{N} p_i \log_2(p_i) .$$

Let $H_M$ be the maximum entropy of the system we want to measure, for the actual number of users:

$$H_M = \log_2(N) ,$$

where $N$ is the number of users (size of the anonymity set).

The information the attacker has learned with the attack about a particular request can be calculated as:

$$H_M - H(X) .$$

We divide by $H_M$ to normalize the value. We then define the **degree of anonymity** provided by the system for a particular request $R_j$ as:

$$d_j = 1 - \frac{H_M - H(X)}{H_M} = \frac{H(X)}{H_M} .$$

For the particular case of one possible sender we assume $d_j$ to be zero. It follows immediately that $0 \leq d_j \leq 1$:

- $d_j = 0$ when a user appears as being the originator of a request with probability 1.
- $d_j = 1$ when all users appear as being the originator with the same probability.
**Average degree of anonymity**

The proposed model allows us to calculate the degree of anonymity obtained for each request. Given that during the attack \( R \) requests have been produced, we define the **average degree of anonymity** as:

\[
d = \frac{\sum_{j=1}^{R} d_j}{R}.
\]

This gives an accurate idea on the degree of anonymity provided by the system for request on average.

**4.1.3 Attack models**

The degree of anonymity depends on the probabilities of having sent a particular request that the attacker is able to assign to the users. The degree is therefore measured *with respect to* a particular attack: the results obtained are no longer valid if the attack model changes. Concrete assumptions about the attacker have to be clearly specified when measuring the degree of anonymity.

We consider a very powerful attacker, who can monitor all communication lines of the system and knows the number of active users in the system and the number of groups. The attacker also knows the group of the user that generated a particular request, and the number of requests produced by every user \( r_j \). The attacker wants to find out the identity of the user that generated a particular request. If there are several users that belong to a group, the attacker is not able to distinguish which member of the group generated the request. The attacker uses all the available information to assign probabilities of being the originator of the request to all the users. The attacker can record all traffic information in the system, and then, for each request, the attacker analyzes the outputs of the system in a period of time which is the maximum delay of the network. He uses this information to assign to every user a probability of having produced a particular request. During an attack, we assume that the number of users in the system, \( N \), is constant, and that the groups are static.

**4.1.4 Practical case: targeted advertising**

We can apply this model to the privacy-preserving targeted advertising system, described in the previous chapter.
Attacks

We want to protect the user from the banner server, so we assume that the attacker controls the banner server and can also observe the traffic between the users and the mix network.

We consider two cases, in the first one we assume that the attacker, given a user, does not know the group of the user (cannot access the profile), and in the other we assume the attacker knows this information.

**Attack 1**

This attacker has access to the information stored in the banner server (relationship profile–web page), and can also see how many requests are generated by each user. The attacker cannot see the profile of each user, so he cannot know the group to which the user belongs.

For each request, the attacker wants to find the user who generated it, so he will take all the users who made a request in a period of time equal to the maximum delay of the network and calculate the probability for each user $u_i$ as follows:

$$p_i = \frac{r_i}{R}.$$  

Where $R$ is the total number of requests produced in the period of time and $r_i$ is the number of requests produced by user $u_i$.

To calculate the degree of anonymity obtained for the request $R_j$ we apply the formula:

$$d_j = 1 - \frac{H_M - H(X)}{H_M} = \frac{H(X)}{H_M}.$$  

Where $H_M$ is the maximum entropy for a number of users equal to the total number of active users and $H(X)$ is the entropy calculated for the distribution obtained with the $p_i$.

In this case, the users who are more active just before the request arrives to the banner server appear more likely than the others.

To calculate the average degree of anonymity provided by the system, we calculate $d_j$ for each $R_j$ and then compute the average ($d$).

**Attack 2**

In this case the attacker also knows the group (profile) of each user. Once the attacker gets a request, he will only look at the active users that belong to a particular group.
Let us assume that the request the attacker is analyzing was generated by a user belonging to group $g_k$. This group contains $N_k$ users who made a request during the attack time.

The attacker will calculate the distribution of probabilities for each user $u_i$ that belongs to $g_k$ as:

$$p_i = \frac{r_i}{R_{g_k}},$$

and $p_i$ is zero for the users that belong to other groups. $R_{g_k}$ denotes the number of requests that arrived close to the request we want to attack and that have associated the group (profile) $g_k$.

The rest of the analysis is analogous to the previous case. Note that we still compare the obtained entropy with the optimal case ($H_M$ is the same), so the degree of anonymity we will obtain in this case is much less than in the previous one.

### 4.2 Model to measure the anonymity provided by an anonymous communication system

We introduce an information theoretic model that allows to quantify the degree of anonymity provided by schemes for anonymous connections. These schemes are relied upon in both the targeted advertising and the P2P applications. We consider attackers that obtain probabilistic information about users. The degree is based on the probabilities an attacker, after observing the system, assigns to the different users of the system as being the originators of a message. As a proof of concept, the model is applied to some existing systems. The model is shown to be very useful for evaluating the level of privacy a system provides under various attack scenarios, for measuring the amount of information an attacker gets with a particular attack and for comparing different systems amongst each other. The proposed model has been published in [DSCP02].

#### 4.2.1 Introduction

As explained earlier, a distinction can be made between connection anonymity and data anonymity. Data anonymity is about filtering any identifying information out of the data that is exchanged in a particular application. Connection anonymity is about hiding the identities of source and destination during the actual data transfer. The model presented in this section
focuses on the level of connection anonymity a system can provide, and does not indicate any level of data anonymity.

Information theory has proven to be a useful tool to measure the amount of information. We try to measure the information obtained by the attacker. A model is proposed, based on Shannon’s definition of entropy, that allows to quantify the degree of anonymity of an electronic system. This degree will be dependent on the power of the attacker.

4.2.2 System model

We focus on systems that provide anonymity through mixes. The system model we consider, thus consists of the following entities:

Senders. These are users who send (or have the ability to send) messages to recipients. These messages can be emails, queries to a database, requests of web pages, or any other stream of data. The senders can be grouped into the set of senders, that is also called the anonymity set. These are the entities of the system whose anonymity we want to protect.

During the attack, we consider the number of senders constant, and senders behaving as independent, identical, stationary stochastic Poisson processes. This is a standard assumption for modeling the behavior of users making phone calls. This means that all users send, in average, the same amount of messages, and the interval of time between one message and the next one follows a Poisson distribution.

Recipients. These are the entities that receive the messages from the senders. Recipients can be active (if they send back answers to the senders) or passive (if they do not react to the received message). Depending on the system there is a large variety of recipients. Some examples are web servers, databases, email accounts or bulletin boards where users can post their messages. The attacker may use the reply messages to gain information.

Mixes. These are the nodes that are typically present in solutions for anonymous connections. They take messages as input, and output them so that the correlation with the corresponding input messages is hidden. There are many different ways to implement a mix; if more than a single mix is used (which is usually done in order to achieve better security), there are several methods to route the message through a chain of mixes. In some of the systems, e.g., Crowds, the nodes do not have mixing properties as the
ones described by Chaum [Cha81]. In these cases the actual properties of the intermediate nodes will be mentioned.

Note that in some systems the intersection between the different sets might be non-empty (e.g., a sender could be at the same time a recipient or a mix).

Examples of systems that provide anonymous connections are Crowds [RR98] and Onion Routing [GRS99]. The proposed measurement model is shown to be suitable for these systems. It is however generally applicable to any kind of system.

**Attack model**

The degree of anonymity depends on the probabilities that the users have sent a particular message; these probabilities are assigned by the attacker. The degree is therefore measured with respect to a particular attack: the results obtained for a system are no longer valid if the attack model changes. Concrete assumptions about the attacker have to be clearly specified when measuring the degree of anonymity.

We briefly recall the attacker properties we consider:

- **Internal-External**: An internal attacker controls one or several entities that are part of the system (e.g., the attacker can prevent the entity from sending messages, or he may have access to the internal information of the entity); an external attacker can only compromise communication channels (e.g., he can eavesdrop or tamper with messages).

- **Passive-Active**: A passive attacker only listens to the communication or reads internal information; an active attacker is able to add, remove and modify messages or adapt internal information.

- **Local-Global**: A global attacker has access to the whole communication system, while a local attacker can only control part of the resources.

Different combinations of the previous properties are possible, for instance a global passive external attacker is able to listen to all the channels, while a local internal active attacker can control, for example, a particular mix, but is unable to get any other information.

In our model, an attacker will carry out a *probabilistic attack*. It has been pointed out by Raymond in [Ray00] that these attacks have not been thoroughly addressed so far. With such an attack, the adversary obtains
probabilistic information of the form *with probability* $p$, $A$ *is the sender of the message.*

### 4.2.3 Proposed measurement model

As explained above, we adopt the definition given by Pfitzmann and Köhntopp in [PK00]. Anonymity is *the state of being not identifiable within a set of subjects, the anonymity set.* A sender is identifiable when we get information that can be linked to him, e.g., the IP address of the machine the sender is using.

We only consider *sender anonymity.* This means that for a particular message the attacker wants to find out which subject in the *anonymity set* is the originator of the message. The *anonymity set* in this case is defined as *the set of honest\(^1\) users who might send a message.* It is clear that the minimum size of the anonymity set is 2 (if there is only one user in the anonymity set it is not possible to protect his identity).

Our definition for the degree of anonymity is based on probabilities: after observing the system, an attacker will assign to each user a probability of being the sender.

### 4.2.4 Degree of anonymity provided by the system

According to the previous definitions, in a system with $N$ users, the maximum degree of anonymity is achieved when an attacker sees all subjects in the anonymity set as equally probable of being the originator of a message. Therefore, in our model the degree of anonymity depends on the distribution of probabilities and not on the size of the anonymity set, in contrast with previous work [BPS00]. This way, we are able to measure the quality of the system with respect to the anonymity it provides, independently from the number of users who are actually using it. Nevertheless, note that the size of the *anonymity set* is used to calculate the distribution of probabilities, given that the sum of all probabilities must be 1.

The proposed model compares the information obtained by the attacker after observing the system against the optimal situation, in which all honest users seem to be equally probable as being the originator of the message, that is, in a system with $N$ users, the situation where the attacker sees all users as being the originator with probability $1/N$.

\(^1\)Users controlled by the attacker are not considered as part of the anonymity set, even if they are not aware of this control.
After observing the system for a while, an attacker may assign some probabilities to each sender as being the originator of a message, based on the information the system is leaking, by means of traffic analysis, timing attacks, message length attacks or more sophisticated attacks.

For a given distribution of probabilities, the concept of entropy in information theory provides a measure of the information contained in that distribution. We use entropy as a tool to calculate the degree of anonymity achieved by the users of a system towards a particular attacker. The entropy of the system after the attack is compared against the maximum entropy (for the same number of users). This way we get an idea of how much information the attacker has gained, or, in other words, we compare how distinguishable the sender is within the set of possible senders after the attack.

Let \( X \) be the discrete random variable with probability mass function 
\[ p_i = P_X(X = i), \]
where \( i \) represents each possible value that \( X \) may take. In this case, each \( i \) corresponds to an element of the anonymity set (a sender). We denote by \( H(X) \) the entropy of the system after the attack has taken place. For each sender belonging to the senders set of size \( N \), the attacker assigns a probability \( p_i \). \( H(X) \) can be calculated as:

\[
H(X) = - \sum_{i=1}^{N} p_i \log_2(p_i) .
\]

Let \( H_M \) be the maximum entropy of the system we want to measure, for the actual size of the anonymity set:

\[
H_M = \log_2(N) ,
\]

where \( N \) is the number of honest senders (size of the anonymity set).

The information the attacker has learned with the attack can be expressed as \( H_M - H(X) \). We divide by \( H_M \) to normalize the value. We then define the **degree of anonymity** provided by the system as:

\[
d = 1 - \frac{H_M - H(X)}{H_M} = \frac{H(X)}{H_M} .
\]

For the particular case of one user we assume \( d \) to be zero.

This degree of anonymity provided by the system quantifies the amount of information the system is leaking. If in a particular system a user or a small group of users are shown as originators with a high probability
with respect to the others, this system is not providing a high degree of anonymity.\footnote{On the other hand, note that any system with equiprobable distribution will provide a degree of anonymity of one, therefore a system with two senders will have $d = 1$ if both of them are assigned probability 1/2. This is because the definition of anonymity we are using is independent of the number of senders.}

It follows immediately that $0 \leq d \leq 1$:

- $d = 0$ when a user appears as being the originator of a message with probability 1.
- $d = 1$ when all users appear as being the originator with the same probability ($p_k = 1/N$).

### 4.2.5 Measuring the degree of anonymity provided by some systems

In this section we apply our proposed measurement model in order to analyze the degree of anonymity provided by some existing systems, in particular Crowds and Onion Routing.

**A simple example: mix based email.**

As a first example, let us consider the system shown in Fig. 4.1. Here we have a system that provides anonymous email with 10 potential senders, a mix network and a recipient. The attacker wants to find out which of the senders sent an email to this particular recipient. By means of timing attacks and traffic analysis, the attacker assigns a certain probability to each user as being the sender. The aim of this example is to give an idea on the values of the degree of anonymity for different distributions of probabilities.

**Active attack.** We first consider an active internal attacker who is able to control eight of the senders (that means that these eight users have to be
excluded from the anonymity set). He is also able to perform traffic analysis in the whole mix network and assign probabilities to the two remaining senders. Let $p$ be the probability assigned to user 1 and $1 - p$ the probability assigned to user 2.

The distribution of probabilities is:

$$p_1 = p \quad p_2 = 1 - p ,$$

and the maximum entropy for two honest users is:

$$H_M = \log_2(2) = 1 .$$

In Fig. 4.2a we show the variation of the degree of anonymity with respect to $p$. As we could expect from the definitions, we see that $d$ reaches the maximum value ($d = 1$) when both users are equiprobable ($p = 1/2$). Indeed, in this case the attacker has not gained any information about which of the two active users is the real sender of the message by analyzing the traffic in the mix network. The minimum level ($d = 0$) is reached when the attacker can assign probability one to one of the users ($p = 0$ or $p = 1$).

This simple example can be useful to get an idea on the minimum degree of anonymity that is still adequate. Roughly, we suggest that the system should provide a degree $d \geq 0.8$. This corresponds to $p = 0.25$ for one user and $p = 0.75$ for the other. In the following examples, we will again look at the probability distributions that correspond to this value of the degree, in order to compare the different systems. Nevertheless, the minimum acceptable degree for a particular system may depend on the anonymity requirements for that system, and we believe that such a minimum cannot be suggested before intensively testing the model.

We now consider a passive global external attacker who is able to analyze the traffic in the whole system, but who does not control any of the entities (the anonymity set is, therefore, composed by 10 users). The maximum entropy for this system is:

$$H_M = \log_2(10) .$$

The attacker comes to the following distribution:

$$p_i = \frac{p}{3} , \quad 1 \leq i \leq 3 \quad p_i = \frac{1 - p}{7} , \quad 4 \leq i \leq 10 .$$

In this case we have two groups of users, one with three users and the other one with seven. Users belonging to the same group are seen by the attacker as having the same probability.
In Fig. 4.2b we can see the variation of $d$ with the parameter $p$. The maximum degree $d = 1$ is achieved for the equiprobable distribution ($p = 0.3$). In this case $d$ does not drop to zero because in the worst case, the attacker sees three users as possible senders with probability $p = 1/3$, and therefore he cannot identify a single user as the sender of the message. The reference value of $d = 0.8$ is reached when three of the users are assigned probability $p_i = 0.25$, and the remaining seven users are assigned probability $p_i = 0.036$.

4.2.6 Crowds

**Overview of the system.** Crowds [RR98] is designed to provide anonymity to users who want to access web pages. To achieve this goal, the designers introduce the notion of “blending into a crowd”: users are grouped into a set, and they forward requests within this set before the request is sent to the web server. The web server cannot know from which member the request originated, since it gets the request from a random member of the crowd, that is forwarding the message on behalf of the real originator. The users (members of the crowd) are called *jondos*.

The system works as follows: when a *jondo* wants to request a web page it sends the request to a second (randomly chosen) *jondo*. This *jondo* will, with probability $p_f$, forward the request to a third *jondo* (again, randomly chosen), and will, with probability $(1 - p_f)$ submit it to the server. Each *jondo* in the path (except for the first one) chooses to forward or submit the request *independently* from the decisions of the predecessors in the path.

Communication between *jondos* is encrypted using symmetric techniques, and the final request to the server is sent in clear text. Every *jondo* can ob-
Figure 4.3: Example of a Crowds system with 7 jondos

serve the contents of the message (and thus the address of the target server), but it cannot know whether the predecessor is the originator of the message or whether he is just forwarding a message received by another member.

Note that for this system the mizes are the jondos, and they do not have some of the expected characteristics. In particular, they do not make any effort to hide the correlation between incoming and outgoing messages.

Attacker. We calculate the degree of anonymity provided by Crowds with respect to collaborating crowd members, that is, a set of corrupted jondos that collaborate in order to disclose the identity of the jondo that originated the request. The assumptions made on the attacker are:

- **Internal**: The attacker controls some of the entities that are part of the system.

- **Passive**: The corrupted jondos can listen to communication. Although they have the ability to add or delete messages, they will not gain extra information about the identity of the originator by doing so.

- **Local**: We assume that the attacker controls a limited set of jondos, and he cannot perform any traffic analysis on the rest of the system.

Degree of anonymity. Figure 4.3 shows an example of a crowds system. In this example the jondos 1 and 2 are controlled by the attacker, i.e., they are collaborating crowd members. A non-collaborating jondo creates a path that includes at least one corrupted jondo\(^3\). The attacker wants to know which of the non-collaborating jondos is the real originator of the message.

---

\(^3\)If the path does not go through a collaborating jondo the attacker cannot get any information.
Generally, let $N$ be the number of members of the crowd, $C$ the number of collaborators, $p_f$ the probability of forwarding and $p_i$ the probability assigned by the attacker to the jondo $i$ of having sent the message. The jondos under the control of the attacker can be excluded from the anonymity set. The maximum entropy $H_M$, taking into account that the size of the anonymity set is $N - C$, is equal to:

$$H_M = \log_2 (N - C) .$$

From [RR98] we know that, under this attack model, the probability assigned to the predecessor of the first collaborating jondo in the path (let this jondo be number $C+1$) equals:

$$p_{C+1} = \frac{N - p_f(N - C - 1)}{N} = 1 - p_f \frac{N - C - 1}{N} .$$

The probabilities assigned to the collaborating jondos remain zero, and assuming that the attacker does not have any extra information about the rest of non-collaborators, the probabilities assigned to those members are:

$$p_i = \frac{1 - p_{C+1}}{N - C - 1} = \frac{p_f}{N} , \quad C + 2 \leq i \leq N .$$

Therefore, the entropy of the system after the attack will be:

$$H(X) = \frac{N - p_f(N - C - 1)}{N} \log_2 \left[ \frac{N}{N - p_f(N - C - 1)} \right] + \frac{N - C - 1}{N} \log_2 \left[ \frac{N}{p_f} \right] .$$

The degree of anonymity provided by this system is a function of $N$, $C$ and $p_f$. In order to show the variation of $d$ with respect to these three parameters we chose $p_f = 0.5$ and $p_f = 0.75$, and $N = 5$ (Fig. 4.4a), $N = 20$ (Fig. 4.4b) and $N = 100$ (Fig. 4.4c). The degree $d$ is represented in each figure as a function of the number of collaborating jondos $C$. The minimum value of $C$ is 1 (if $C = 0$ there is no attacker), and the maximum value of $C$ is $N - 1$ (if $C = N$ there is no user to attack). For the case $C = N - 1$ we obtain $d = 0$ because the collaborating jondos know that the real sender is the remaining non-collaborating jondo. We can deduce from the figures that $d$ decreases with the number of collaborating jondos and increases with $p_f$. The variation of $d$ is very similar for systems with different number of users. Regarding the tolerated number of collaborating jondos to obtain $d \geq 0.8$, we observe that for $p_f = 0.5$ the system does not tolerate any corrupted jondo; for $p_f = 0.75$ the system tolerates: for $N = 5$ users, $C \leq 1$, for $N = 20$ users, $C \leq 4$, and for $N = 100$ users, $C \leq 11$. 

42
Figure 4.4: Degree of anonymity for Crowds
In [RR98] a degree of anonymity is defined as \(1 - p_{\text{sender}}\), where \(p_{\text{sender}}\) is the probability assigned by the attacker to a particular user as being the sender. This measure gives an idea of the degree of anonymity provided by the system for a particular user, and it is complementary with the proposed degree. It is interesting to compare the results obtained by Reiter and Rubin in [RR98] with our results (for the same attack model): they consider that the worst acceptable case is the situation where one of the jondos is seen by the attacker as the sender with probability \(1/2\). Therefore, they come to the conclusion that, for \(p_f = 0.75\), the maximum number of collaborating jondos the system can tolerate is \(C \leq N/3 - 1\). For the chosen examples we obtain: for \(N = 5\) users, \(C = 0\), for \(N = 20\) users, \(C \leq 5\), and for \(N = 100\) users, \(C \leq 32\).

Degree of anonymity from the point of view of the sender. We have calculated the degree of anonymity of a user who sends a message that goes through a corrupted jondo, but this only happens with probability \(C/N\) each time the message is forwarded to another jondo. We have to take into account that the first jondo always forwards the message to a randomly chosen jondo of the crowd, and subsequent jondos forward with probability \(p_f\) to another jondo, independently from previous decisions. The probability \(p_H\) of a message going only through honest jondos is:

\[
p_H = \frac{N - C}{N} (1 - p_f) \sum_{i=0}^{\infty} \left( \frac{N - C}{N} p_f \right)^i = 1 - \frac{C}{N - p_f(N - C)}.
\]

If a message does not go through any collaborating jondo, the attacker will assign all honest senders the same probability, \(p_i = 1/(N - C)\), and the degree of anonymity will be \(d = 1\) (the maximum degree is achieved because the attacker cannot distinguish the sender from the rest of honest users).

4.2.7 Onion Routing

Overview of the system. Onion Routing [GRS99] is a solution for application-independent anonymous connections. The network consists of a number of onion routers. They have the functionality of ordinary routers, combined with mixing properties. Data is sent through a path of onion routers, which is determined by an onion.

An onion is a layered encrypted data structure, that is sent to an onion router. It defines the route of an anonymous connection. It contains the next hop information, key seed material for generating the symmetric keys.
that will be used by the onion router during the actual routing of the data, and an embedded onion that is sent to the next onion router.

The data is encrypted multiple times using the symmetric keys that were distributed to all the onion routers on the path. It is carried by small data cells containing the appropriate anonymous connection identifier. Each onion router removes/adds a layer of encryption (using the symmetric keys, generated from the key seed material in the onion) depending on the direction of the data (forwards/backwards).

**Attack model.** Several attack models have been described by Reed, Syverson and Goldschlag in [GRS99]. In this example we consider an attacker who is able to narrow down the set of possible paths. The attacker obtains, as a result of the attack, a subset of the anonymity set that contains the possible senders. We do not make any assumption on the attacker, but that he does not control any user of the system. We make abstraction of the attack, but, in order to illustrate the example, it could be carried out performing a brute force attack, starting from the recipient and following all the possible reverse paths to the senders. Another alternative is that the attacker controls some of the onion routers, and he is able to eliminate a group of users from the anonymity set.

**Degree of anonymity.** Figure 4.5 gives an example of an Onion Routing system. There are in total seven users in this system. We assume that the attacker managed to exclude users 6 and 7 from the set of possible senders.

Generally, let \( N \) be the size of the anonymity set; the maximum entropy for \( N \) users is:
\[ H_M = \log_2(N) \].

The attacker is able to obtain a subset of the *anonymity set* that contains the possible senders. The size of the subset is \( S \) (\( 1 \leq S \leq N \)). We assume that the attacker cannot assign different probabilities to the users that belong to this subset:

\[ p_i = \frac{1}{S} \quad \text{for} \quad 1 \leq i \leq S \quad \text{and} \quad p_i = 0 \quad \text{for} \quad S + 1 \leq i \leq N. \]

Therefore, the entropy after the attack has taken place, and the degree of anonymity are:

\[ H(X) = \log_2(S) \quad \text{and} \quad d = \frac{H(X)}{H_M} = \frac{\log_2(S)}{\log_2(N)}. \]

Figure 4.6 shows the degree of anonymity with respect to \( S \) for \( N = 5 \), \( N = 20 \) and \( N = 100 \). Obviously, \( d \) increases with \( S \), i.e., when the number of users that the attacker is able to exclude from the *anonymity set* decreases. In order to obtain \( d \geq 0.8 \): for \( N = 5 \) users we need \( S \geq 3 \); for \( N = 20 \) users, we need \( S \geq 12 \); and for \( N = 100 \) users, we need \( S \geq 40 \).

When comparing \( N - S \) to the number of collaborating *jondos* \( C \) in the Crowds system, it seems that Onion Routing is much more tolerant against ‘failing’ users/ *jondos* than Crowds. This is because the remaining ‘honest’ users/ *jondos* have equal probability (for this attack model) in the Onion Routing system, while in Crowds there is one *jondo* that has a higher probability than the others.
4.3 The binomial mix

In this section we describe a new theoretical mix design that uses randomness in order to make more difficult for an attacker to trace a message that goes through the mix.

4.3.1 Communication layer

Internet applications need to provide anonymity at every layer. If the communication layer is not anonymized an attacker can observe the IP address of the user and with this information disclose his identity. Therefore, an anonymous communication infrastructure is needed at the communication layer for anonymous applications.

4.3.2 Mixes

The typical way of implementing an anonymous communication infrastructure is a mix network. A mix is a router that takes several inputs and outputs them in a way that it is difficult to find a correlation between inputs and outputs. Some of the typical working modes of mixes are explained below:

Threshold mix This mix collects \( n \) inputs, then reorders them and flushes them.

Timed mix This mix collects inputs for \( t \) seconds, then reorders them and flushes.

Threshold pool mix This mix collects \( n \) inputs, then selects (randomly) \( n - f \) messages which are flushed, and keeps \( f \) messages in the pool.

Timed pool mix This mix collects inputs for \( t \) seconds. Then, it leaves \( f \) messages (randomly selected) in the pool, and flushes the rest.

Cortrell mix This mix gets flushed every \( t \) seconds. A fraction \( f \) of randomly chosen messages remain in the pool of the mix; but if there are less than a minimum number of messages in the pool at the time of flushing, it outputs no messages.
Stop-and-go mixes These mixes work in a substantially different fashion than previous designs. Users generate a delay that follows an exponential distribution and the mix keeps them for this period of time. It is necessary to have a sufficiently high traffic load in order to achieve anonymity. Note that if the traffic load is low, messages may not be mixed at all.

4.3.3 The blending attack
Mixes are vulnerable to the blending attack. The attack can be deployed as follows. First, the attacker fills the mix with his own messages, preventing the users’ messages from entering the mix. Then, he lets the target message into the mix. In the output, there will be only one unknown message, since the attacker can use several techniques in order to recognize his own messages. The attacker can trace the message from sender to recipient using this attack, breaking the anonymity of the system. The blending attack is also called the n-1 attack.

4.3.4 Description of the binomial mix
We propose a new mix design that is more resistant to the n – 1 attack and we analyze the effort required by the attacker in order to trace a message that goes through a single binomial mix.

The binomial mix collects messages for a fixed period of time, the timeout, T. The messages are stored in a pool of size \( N_{\text{max}} \), in random positions. If a new message arrives when the pool is full, it will be dropped. Every timeout the mix runs the selection algorithm; which results in a number of messages being taken from the pool and forwarded to the next mix or to the final recipient.

Selection algorithm. The messages are taken from the pool using a binomial distribution. The algorithm works as follows: let \( p \) be the probability of taking a message from the pool. A biased coin (the probability of head is \( p \) and the probability of tail is \( 1 - p \)) is thrown for each message, if the result is head, then the message is taken from the pool; otherwise it stays there for, at least, one more round.

Note that each message is withdrawn from the pool with probability \( p \), independently. This is a series of Bernoulli trials that results in a binomial distribution. If \( n \) is the number of messages contained in the pool at a particular round, then the number of messages selected, \( s \), follows a binomial distribution.
**Fixed** \( p \). Using a fixed \( p \) has a serious shortcoming: lack of flexibility. We should take into account that the traffic load can have large variations. If the chosen \( p \) is too big then the mix can be empty or almost empty in low traffic load conditions, which is dangerous because reduces the size of the anonymity set (the message is mixed with very few messages, or not mixed at all). On the other hand, choosing a small \( p \) is not convenient if the traffic load is high, because messages would have an unnecessary large delay and could even be dropped due to a lack of memory in the pool.

**The \( P(n) \)-function.** We would like to have a \( p \) that adapts to the traffic load. It should have a small value when there are few messages in the pool, and a big value when the pool is reasonably filled up. Therefore, we can define the probability of sending a message as a function of the number of messages contained in the pool. We call this function \( P(n) \), where \( n \) is the current number of messages in the pool. The result of this function is be \( p \), the probability of sending messages in this round.

The function \( P(n) \) should have a smooth growth in order to avoid large variations in \( s \) that could reveal information about \( n \). We suggest the use of a cumulative distribution function of the normal distribution. If we want to guarantee that a minimum number of messages, \( N_{min} \), will be in the pool, we can define \( P(n) \) to be 0 for \( n \leq N_{min} \). Note that the function should be rescaled in order to limit the maximum value of \( p \). An example is shown in Fig. 4.7.
Active attack. The number of messages sent, $s$, is a function of the number of messages contained in the pool, $n$. On average, $s = nP(n)$; but $s$ follows a binomial distribution, which has a variance equal to $np(1 - p)$, where $p$ is the result of the function $P(n)$. The property of the mix is that by observing $s$ the attacker does not obtain much information about the value of $n$. The effort required to estimate $n$ is analyzed in section 4.3.5. The usefulness of this property becomes more clear when binomial mixes are used as nodes of a mix network implementing a dummy traffic policy.

4.3.5 Guessing the number of messages contained in the mix

We analyse the information obtained by a passive attacker that observes the input and output of the binomial mix. Then we explain how the attacker can combine the information obtained in multiple observations and give an estimate of the number of rounds needed to accurately guess $n$. The results we are showing have been computed with a simulator that uses a non-optimized $P(n)$-function. Therefore, better results could be achieved by using a different $P(n)$-function.

Observation of one output.

When the attacker is allowed to observe only one output, the only available information he has is $s$. We have constructed a simulator that calculates the probabilities of every value of $n$ after observing that the mix outputs $s$ messages.

Given $n$, we can calculate the probability of sending $s$ messages with the following formula, according to the binomial distribution:

$$p(s|n) = \frac{n!}{s!(n-s)!}p^s(1-p)^{n-s}, \quad (4.1)$$

where $p$ is the result of the function $P(n)$.

But the attacker does not know $n$, he has to estimate $n$ from the observation of $s$. Bayes’ rule can be applied to reverse the formula and compute the probability of each $n$.

$$p(n|s) = \frac{p(s|n)}{\sum_{i=s}^{N_{max}} p(i|n)} . \quad (4.2)$$

\footnote{Given that the attacker does not have any \textit{a priori} information he must assume, initially, that any possible value of $n$ between $s$ and $N_{max}$ is equally probable.}

50
The attack is implemented as follows: the attacker observes $s$ and assumes that the $n$ that generated this output is at least $s$ and at most $N_{\text{max}}$. In order to compute the probability of $n$ taking a particular value, say 100, we apply equation 4.1 using this value for $n$, and then substitute the result in equation 4.2. We also need to calculate the result of equation 4.1 for this $n$ and every possible value of $s$.

Using this formula the attacker can obtain the probability of each value of $n$ given than the mix has sent $s$ messages. The practical results show that the attacker cannot guess the value of $n$ with probability greater than 15%. We have also calculated the 95% confidence interval and found that, typically, it contains between 12 and 30 different values of $n$. This is due to the large value of the variance of a binomial distribution.

**Number of rounds needed to estimate $n$ with 95% confidence.**

We have implemented a passive attack in the simulator in order to have an estimate on the number of rounds required by the attacker to guess with probability 95% the correct value of $n$.

Given that every round is independent from the others, we can multiply the results of every round, taking care of shifting the terms we are multiplying as many positions as the difference between the $n$ of the first round of attack, $n(0)$, and the current $n$, $n(t)$. This difference is known to the attacker because he can count the incoming and outgoing messages.

The attacker, according to the results of the simulations, needs typically close to 200 rounds of observation. This number could be improved by choosing a more appropriate $P(n)$-function. In terms of time, he will have to wait the number of rounds times $T$.

**4.3.6 The blending attack on the binomial mix**

As we have seen in the previous section, a passive attacker needs a substantial number of rounds of observation in order to accurately guess the current $n$. Therefore, it does not seem to be practical to deploy a blending attack using the same strategy as with classical pool mixes.

In this section we describe first the attack model, then the steps needed in order to deploy a blending attack and, finally, we analyze the results.

**Attack model.**

The attacker we consider controls all communication lines (global attacker). He can not only observe all incoming and outgoing messages, but also delay
the messages of the users and insert messages (active attacker). The attacker
does not have access to the contents of the mix, i.e., the mix is a black box
for the attacker (external attacker). In order to test the effectiveness of the
design, we consider a setup with only one mix.

The flooding strategy.

The goal of the attacker is to trace a particular message (the target message)
that is sent by a user to the mix. The actions of the attacker can be divided
into two phases: the emptying phase and the flushing phase.

The emptying phase. During this stage of the attack, the goal of the
attacker is to remove all unknown messages contained in the pool, while
preventing new unknown messages from going into the mix. In order to force
the mix to send out as many unknown messages as possible in each round, the
attacker sends to the mix $N_T$ messages, where $N_T$ is the minimum number of
messages that guarantees that the $P(n)$ function takes its maximum value,
$p_{\text{max}}$. If the attacker wants to empty the mix with probability $1 - \epsilon$, then
he will have to flood the mix for $r$ rounds.

The formula that can be used to estimate the number of rounds needed
to flush all unknown messages with probability $1 - \epsilon$ is:

$$
(1 - (1 - p_{\text{max}})^r)^n \geq 1 - \epsilon
$$

Where $n$ is the number of messages contained in the pool. If the attacker
does not have any information about $n$ he will have to assume $n = N_{\text{max}}$.

Cost of emptying the mix. We compute the cost, $C_E$, of this phase of
the attack taking into account the following:

- Number of messages the attacker has to send to the mix.
- Time needed to complete the operation.
- Number of messages the attacker has to delay.

Number of messages the attacker has to send to the mix. In the
first round the attacker has to send $N_T$ messages, to ensure that the function
$P(n)$ takes its maximum value, $p_{\text{max}}$, and therefore the probability of each
message leaving is maximum. In the following rounds, it is enough to send
as many messages as the mix output. Note that if $n + N_T$ is bigger than

52
$N_{max}$, then some messages will be dropped and the mix will contain $N_{max}$ messages.

Thus, for the first round the attacker sends $N_T$ messages, and the following rounds he sends $(N_T + n)p_{max}$ messages in average. The total number of messages sent during this process is:

$$\text{Number of messages sent} = N_T + (r - 1)(N_T + n)p_{max} \quad (4.4)$$

**Time needed to complete the operation.** This is a timed mix, so the attacker has to wait $T$ units of time for each round. Therefore, the total time needed is $rT$ time units.

**Number of messages the attacker has to delay.** Assuming that the users generate messages following a Poisson distribution with parameter $\lambda$, the attacker has to delay, in average, $\lambda rT$ messages.

**The flushing phase.** Once the mix has been emptied of unknown messages, the attacker sends the target message to the mix. Now, he has to keep on delaying other incoming unknown messages and also send messages to make the mix flush the target.

The number of rounds needed to flush the message is, in average, $r = \frac{1}{p_{max}}$. The cost of this phase is computed according to the previous parameters.

**Number of messages the attacker has to send to the mix.** Assuming that the attacker carries out this phase of the attack immediately after the emptying phase, the number of messages needed in the first round is $(N_T + n - 1)p_{max}$, and in the following ones $(N_T + n)p_{max}$. The total number of messages is:

$$p_{max}(N_T + n - 1 + (r - 1)(N_T + n)) \quad (4.5)$$

The other two parameters are computed in the same way as in the emptying phase, taking into account the new value of $r$.

**Guessing the number of messages within the mix with an active attack**

The attacker can use the flooding strategy (emptying phase only) in order to determine the number of messages contained in the pool of the mix. This
attack is much faster than the one described in 4.3.5, although it requires more effort of the attacker.

**Probabilistic success.**

Note that, due to the probabilistic nature of the binomial mix, the attacker only succeeds with probability $1 - \epsilon$. Therefore, with probability $\epsilon$ there is at least one unknown message in the mix. In this particular case, the attacker can detect his failure if during the flushing phase more than one unknown message leaves the mix in the same round (and there is no dummy traffic policy), which happens with probability $p_{max}^2$ for the case of one unknown message staying during the emptying phase (the most probable case). With probability $p_{max}(1 - p_{max})$ the target message leaves the mix alone, and the attack is successful. Also with probability $p_{max}(1 - p_{max})$, the other unknown message leaves the mix first, and the attacker follows a message that is not the target without noticing. Finally, with probability $(1 - p_{max})^2$, both messages stay in the pool and the situation is repeated in the next round.

### 4.3.7 Average delay of a message.

Assuming that the population of users generate messages following a Poisson distribution with mean $\lambda$ messages per time unit, and given that the mix flushes messages every $T$ time units, the average number of messages going into the mix per round is $\lambda T$. Assuming that the mix outputs as many messages as it gets (that is, the $P(n)$ function and $N_{max}$ are designed in such a way that the probability of dropping messages because of a lack of space in the mix is very small), the average number of messages sent per round is $s = \lambda T$. We know that $s = nP(n)$, therefore, we have to find $n$ such that $nP(n) = \lambda T$. This number can be found recursively.

Given that the average number of rounds that a message spends into the mix is $1/P(n)$, where $n$ has to be computed as stated above, the average delay of a message going through the binomial mix is $\frac{1}{P(n)}$ time units.

### 4.3.8 Additional measure: Timestamps.

Additional measures, like timestamps, can be used in order to prevent the blending attack. This idea has already been proposed by Keadogan et al. in [KEB98] for the Stop-and-Go (SG) mixes.
SG mixes work in a different way than pool mixes: users, after choosing the path of mixes, generate a timestamp for each mix in the path that follows an exponential distribution. The message is encrypted several times, each time with the key of one of the mixes. Once an SG mix has received and decrypted a message, it keeps it in the memory a period of time equal to the delay indicated by the user. Then, it forwards the message to the next mix.

**Link Timestamps.**

In our design, the user cannot generate timestamps for every mix in the path, because he does not know how long the message is going to be delayed in each mix. Therefore, we propose the use of link timestamps: the user generates a timestamp for the first mix and, in each of the following hops, the mix puts the timestamp on the message once the message has been taken from the pool and is going to be sent.

When a mix receives a timestamp that is too old, it drops the message. With this policy, the attacker has limited time to delay messages: if he delays the target message too long it will be dropped, and the attacker will not have any means to disclose the recipient of the message.

Using this measure prevents the attacker from delaying the target message at his will, and the attacker does not have means to deploy a blending attack (unless he knows that the message is going to be sent by the user in advance, and can empty the mix before). Therefore, in this scenario the binomial mix provides protection against the blending attack. Furthermore, the anonymity provided by the binomial mix will not be threatened by a change in the traffic load while this change, if large enough, can affect the anonymity provided by a SG mix (since SG mixes only delay messages).

**Drawbacks.**

The use of timestamps presents practical problems, and this is the reason why we have not included them in the basic design. The most serious problem is the synchronization of clocks. If the different computers (both users and mixes) have a deviation in their clocks, many valid messages are dropped. All entities could be synchronized using a time server, but then the security of this time server becomes an issue.

Also, timestamps are not so effective if we are dealing with corrupted mixes: a corrupted mix can put a fake timestamp on a message and give the attacker extra time to empty the following mix in the path.
4.4 Building block for Targeted Advertising

As part of the implementation of the privacy-preserving Targeted Advertising system, we have developed an HTTP filter. HTTP filters can be used in order to protect the privacy of the users while they are surfing the Internet. These filters can intercept cookies, or check if private information is being sent to the web sites.

4.4.1 Description

It consists of a proxy that intercepts the communication between the browser and the Internet. In this prototype, the functionality of the proxy is still limited. The filter has been designed to intercept and modify a particular cookie.

The profile cookie is used by a banner server. This banner server is intended to serve banners related to the users’ interests while avoiding identification. The profile cookie contains declared information (that the user may choose not to give) and information about the history of the user. In particular, it keeps record of the type of pages accessed. The user can delete or change this information when he desires to do so.

When the filter detects that the cookie profile is going to be sent, it calls a Java class that computes the actual value of the profile, according to the information stored in the user’s computer.

The banner server returns a cookie that contains the update of the user’s profile, which is the last item to be added to the history of the user. At this stage, the filter intercepts this information, and calls another Java program that takes care of the update.

The functionality of the filter could easily be extended in order to deal with other cookies, or to check whether personal information is being sent or not.

The client proxy program is essentially a privacy filter for the HTTP application. It can be enhanced to filter all HTTP headers, and in particular to maintain cookies that are not banner-related.

Note that the proxy should also filter the extra information provided by the web site to the banner server. Web sites that cooperate with the banner server could misuse this side-channel to leak information about the user’s identity (the web site possibly has this information, e.g., in the case of a subscription).
4.5 Building blocks for P2P

The previous chapter contains an overview of the architecture for P2P systems and its layers. In this section, we give a detailed description of the building blocks at each layer. First, we discuss the basic building blocks that are part of the anonymity library. A composition of these building blocks is made by peer level building blocks and connection level building blocks to provide anonymity in the system. Second, the identifier and locator concepts are explained. These concepts are used in the P2P system at different layers. They are orthogonal to the building blocks at peer level and at connection level. Peer level blocks are discussed in the third section. Finally, connection level building blocks are discussed.

4.5.1 Anonymity library

The anonymity library contains basic building blocks for anonymity. These building blocks are split up in two categories: building blocks for changing appearance and building blocks for changing the message flow. We give some examples of blocks that belong to each of these categories. They are discussed in detail in deliverable 3.

- Changing appearance. The intention of these blocks is to change the appearance of messages. Encryption, padding, information substitution and compression belong to these category of basic blocks.

- Changing flow. These blocks change the message flow. Examples of these blocks are reordering, latency, dummy traffic, filtering, ...

By providing an interface for each of these blocks, we hide the implementation to the services that use these blocks. For instance, a service that wants to reorder messages is not concerned about the algorithm that is used to reorder messages. The implementation of an encryption algorithm is also hidden towards the service that uses encryption.

When providing a service, a good composition of building blocks is essential to provide a system that meets the anonymity requirements. Appearance changing blocks will be used together with blocks that change the message flow. They must be composed in a good order. Basic blocks are composed both at peer layer and at connection layer. If the connection layer already supports reordering of messages, it makes no sense that the peer layer reorders messages twice. If the connection layer provides anonymous connections (by making a good connection level composition of basic blocks
or by using an existing connection level system such as onion routing), less blocks should be composed at peer layer. However, building blocks can appear at both layers. Encryption can be inserted as a basic block between connections as well as peers. Information can be substituted at peer layer and at connection layer.

4.5.2 Identifiers and Locators

When we make abstraction of the application domains and architectures of P2P systems, we can introduce the concepts of a node and a unit. We first outline the difference between these two concepts. Thereafter, we associate locators with each of these concepts.

**Node - NodeIdentifier** A node is the fundamental component of a P2P system. Each node is associated with an IP address and port to receive incoming calls. Each node in a P2P system also has a unique nodeIdentifier which represents the node. In pure P2P systems, every node is a peer and vice versa. In hybrid systems, other (assisting) components can also be peers. For instance, in hybrid P2P systems central servers are also nodes. Central servers also have a unique nodeIdentifier.

**Unit - UnitIdentifier** A unit declares a resource in a P2P system. A unitIdentifier is a unique identifier to a unit. The concept of a unit depends on the application domain. In file storage systems, each file is a unit. Each file has a unitIdentifier. A chat session is a unit in chat applications, a process in a unit in distributed computing, etc. We distinguish between single units and composed units.

- **Single unit.** A single unit is a unit that is atomic and can not be split up further. For instance, a file that is stored at one node is a single unit. A single unit is located at a particular place within a node.

- **Composed unit.** A unit can also be a composition of other units. If a file is split in different parts that are stored at different servers, these parts are also units. A chat session is also a composition of different units that participate in the session. A process created for distributed computing is composed of nodes where the computation takes place.
NodeLocator A nodeLocator is information to contact a node in the P2P system. In P2P systems without anonymity, a locator can be an IP address and port. A locator can also be a path to the node. A reply onion is an example of a nodeLocator. Remark that a nodeLocator refers to one single node in the system. It is also important to note that one node can have different nodeLocators. These are different ways to contact a node.

UnitLocator A unitLocator is an information structure to contact a unit in the P2P system. We both discuss unitLocators for single units and unitLocators for composed units.

Single units. The location of a unit consists of a node and a place of the unit within that particular node. In systems without anonymity, a nodeLocator consists of the address of the node and an offset that declares the place of the unit within that node. If the location of a unit must be anonymous, the nodeLocator can be a transformed form of the node and the place of the unit within that node. Files stored at an anonymous node can be retrieved if the unitLocator does not reveal information about the physical location of the unit. Another example is an encrypted url (used by TAZ servers). TAZ servers [TAZ] dispose encrypted urls (unitLocators) that are used to retrieve an anonymous webpages (units).

Composed units. If a unit is a composition of other units, the structure of the unitLocator is more complex:

- **Set of unitIdentifiers.** The locator for a composed unit is a set of unitIdentifiers. A chat session unitLocator is a composition of unitIdentifiers. These unitIdentifiers refer to the participants of the chat session. Each unitIdentifier maps to a unitLocator (i.e. the location of the participant).

- **Set of unitLocators.** The locator for a composed unit is a set of other unitLocators. If a file is distributed between several nodes, the unitLocator to retrieve that file is a composition of the unitLocators. Each unitLocator is a path to a part of the file.

4.5.3 Peer level blocks

In this section, we try to give an overview of building blocks that return in many peer platforms. We outline the functionality of each block and try to
give an API that can be called by the peer platform. The building blocks have two important properties. First, the building blocks are application independent. This means that the blocks can be used for file sharing, distributed computing, collaboration, ... Second, the intention is to reuse these blocks in other systems such as client-server systems, agent systems, ... The implementation of some of these building blocks depends on the P2P model or the anonymity service we want to provide. We can divide the presented blocks in two main categories: blocks that do not call the connection level API (registration block, storage block, mapping block and trust block) and blocks that call the connection level API (push block, endpush block, pull block, reply block and forward block). We discuss each block, explain its functionality and define its interface. The interface contains a layout for constructing an instance of each block and some methods that can be invoked on that block.

Registration block

This block registers a node in the peer system. The interface consists of a constructor to create a new registration block and a method to register a node.

- **Register().** When the application starts up, a new register block is created.

- **registerNode().** This method is invoked immediately after the creation of the register block. The implementation of this block depends on the P2P model. The node can be registered at a central server or at a neighbouring peer. It looks up the right nodeIdentifier (central server or neighbouring node) and calls a push block to send the registration request to the right node.

Storage block

A storage block stores units. A typical example of a unit is a file of a part of a file that is stored. A storage block is created when a node starts up. Each node has one storage block. There are methods to add data, remove data and retrieve data. Remark that each data that is stored is associated with an unitIdentifier.

- **Storage().**
• add(Data data, UnitIdentifier unitIdentifier).

• remove(UnitIdentifier unitIdentifier).

• Data retrieve(UnitIdentifier unitIdentifier).

Mapping block
A mapping block has two main functionalities. First, a mapping block creates locators for nodes and units. Second, it keeps a table that maps locators to identifiers. Just like storage blocks, a mapping block is created when a node starts up. New locators are added to the mapping block if other nodes send locators to that node or if the node that is associated with the mapping block creates new locators.

• Mapping().

• NodeLocator createNodeLocator(). A nodeLocator can be created by a node in its registration phase. This method does not require the node as a parameter because we suppose the node creates a nodeLocator for his own node. If there is a need for a node to create a locator for another node, an additional method "createNodeLocator(NodeIdentifier nodeIdentifier)" is required.

• UnitLocator createUnitLocator(UnitIdentifier unitIdentifier)
A unitLocator is made for the unit that corresponds with the unitID. This locator can later be pushed together with the identifier. For instance, if a file is pushed to another location, a unitLocator must also be pushed so that other nodes can retrieve the file. If a node creates a chat session unit, the node must push a locator so that interested members can subscribe to the session.

• Locator lookup(Identifier identifier). This method looks up a locator for an identifier. A nodeLocator is returned if the parameter is a nodeIdentifier and a unitLocator is returned if the parameter is a unitIdentifier.

• add(Identifier identifier, Locator locator). This method adds an identifier and a corresponding locator to a mapping table. This information can be pushed by other nodes or the node can contact another node to ask a locator that fits with the identifier.
• **remove(Identifier identifier)**. This method removes an identifier and his locator from the mapping table. This is useful if the locator is not valuable any more (for instance if the location of a unit has moved or if the unit/node does not exist anymore).

**Trust block**

While the advantages of a P2P system are obvious, the distributed nature and the independence between its different peers allow the introduction of misbehaving entities into the system. A typical example of such misbehavior is the phenomenon of freeloading in a file sharing system, where certain peers communicate predominantly in the incoming direction and as such profit from all data available of the system without inserting any new data in it on their behalf. To overcome this kind of problems, different types of trust models are being used in P2P implementations. Among them, a very decentralized one keeps for instance records for every of the peers he has had contact with in the past and its behavior (what actions does he do, does he act as promised, ...). When for a new connection request, no information is known locally, a peer could contact well-known other peers to get such information. In another model, this data is kept in a more centralized way and every action within the system requires hence updating this information repository.

The purpose of this building block is to provide a generic interface for different types of trust models. The relatively straightforward interface consists of the following methods:

• **Trust()**

• **getTrustLevel(NodeIdentifier n)**. For a certain node, the trust level can be queried. The result of this query is expressed using some metric. A straightforward metric could consist of an integer representation between certain bounds. However, other, more advanced metrics could be necessary somehow.

• **updateTrustLevel(NodeIdentifier n, Action a)**. After the execution of a certain action, the internal state of the trust model should be updated to reflect this new action. This is the purpose of this method. Remark here that the action still has to be defined. Similarly, a straightforward representation could be a general purpose string. Some trust models might however require more advanced action definitions.
Push block

This block pushes data to other nodes in the system. A push block is created the first time data is pushed. Before data is actually sent over a connection, it can be reordered, delayed, encrypted... Thus, a push block makes a composition of basic blocks that are in the anonymity library. An identifier is related to a push block. This identifier is the node to which information should be pushed. We can set this identifier to another value if data must be pushed to another node. If the push block detects some connection to the node that maps the identifier, the data is sent over that connection. Otherwise, a new connection is created to the node that maps the identifier. The push block keeps a table of open connections to other nodes. Data can be pushed by several entities. First, the application can push data. For instance, some text can be pushed to another node in a chat system. This text is application data. Second, a registration block can push data to register a node in the system. Third, a trust block can push information about the reliability of other nodes.

The push block can be called by a file sharing system when some node wants to push a file towards another node. In a chat application, messages can be pushed to the participants of the chat session. Locators and identifiers can also be pushed (for instance to a central server). Sometimes, not all data should be pushed the same way. For instance, a file can be split in several parts that are sent to several locations. If a chat message is pushed to another node in the system, the message should not be split. A different push block is created when data is handled in a different way before it is actually sent.

- **Push().** This constructor is called in two cases. A first case is when data will be pushed to some default location (for instance a central server). A second case is when the initiator of the push block does not care where the information is pushed.

- **Push(NodeIdentifier nodeIdentifier).** This constructor is called when some data must be pushed to the node with the corresponding nodeIdentifier. It is not required that the node that calls this method has the corresponding nodeLocator to that node. In that case, the data is actually sent to a node that has the nodeLocator that fits with the nodeIdentifier.

- **Push(NodeLocator nodeLocator).** This constructor is called when data must be pushed to the node with the corresponding nodeLocator.
• **pushData(Data data)** This method pushes the data to another peer.

• **pushData(data data, Identifier identifier)**. This method pushes data to the node that maps the identifier.

• **void setIdentifier(Identifier identifier)**. The identifier to which data should be pushed is set to another value.

**EndPush block**

This block receives data that arrives at a peer. A method is provided to construct an object of this type and a method to retrieve data. Data can be received from different connections. A connectionIdentifier denotes from which connection we should retrieve data. This identifier can be changed if data is received from another connection. The retrieved information is passed to a receiver object. For instance, data can be passed to an application object, a trust block, a storage block, etc...

• **EndPush(ConnectionIdentifier connId, Receiver receiver)**. The connectionIdentifier in this block is a reference to the connection from which it receives data. The receiver is the entity that actually receives the data. A storage block can be a receiver of data but also the application. Therefore, these block must implement the Receiver interface. These two parameters can be changed by the methods "setReceiver" and "setConnectionIdentifier".

• **setReceiver(Receiver receiver)**.

• **setConnectionIdentifier(ConnectionIdentifier connectionIdentifier)**.

• **Data retrieveData()** This method actually retrieves data from the connection that corresponds the connectionIdentifier that is passed in the constructor.

**Pull block**

A method of this building block is called when a node requests information. For instance, a node can pull a file. A node can also pull information from a message board or pull a locator from a central server. The information that is pulled is always associated with a unit. Before the request is sent, it can be reordered, encrypted, etc... So, a pull block also consists of a
compositional blocks in the anonymity library. Just like a push block, a pull block is created the first time we want to pull data. Different entities can pull data. For instance, the application can pull a file. A trust block can also pull trust information from other nodes. The “setIdentifier” method is called when we want to pull data from another identifier.

A pull block can be seen as a combination of a push block and an endpush block. The push block sends the request and the endpush block retrieves the answer to that request. Different implementations are possible. We mention two main categories.

- Synchronous pull. The request and the answer are sent on the same connection (see Fig. 4.8.). The connection must be open until the pull block receives an answer.

- Asynchronous pull. The request and the answer are sent on different connections (see Fig. 4.9.). In this case, the pull block must know which answer corresponds to which request. A request identifier must be passed.
We define the constructor and the methods of a pull block:

- **Pull().** A pull block is created the first time a request is made.

- **Data pullRequest(Request request, UnitIdentifier unitIdentifier).** A node calls this method if it wants to request data associated with the unit that has the corresponding identifier. The node does not have the locator to that node. Therefore, the node can first pull the nodeLocator (from a central server) and call the next method. Another option is to send the request together with the identifier to a server that knows the mapping. That server forwards the request.

- **Data pullRequest(Request request).** A request is made to the node that fits the current identifier.

- **Data pullRequest(Request request, UnitLocator unitLocator).** This method is called when the node has a locator to the node. The node returns a reply to the request.

- **setIdentifier(Identifier identifier).** The identifier is set to another value.
Reply block

A reply block has an endpush block and a push block. The request is received by an endpush block, handled by the reply block and the answer is returned by a push block. The reply block can call methods on other blocks to handle the request. For instance, a reply block can ask for a file to the storage block. The reply block can also ask for trust information to the trust block. Replies can be sent along the same connection or along another connection (2 variants to discuss). Another implementation will be required depending on the variant.

- Synchronous reply. The request and answer make use of the same connection.
- Asynchronous reply. The request and answer use a different connection. A requestIdentifier must be passed with the answer.

We define the constructor and method of a reply block:

- Reply(). This constructor is called the first time a request is sent to a node.
- handleRequest(). This method handles and incoming request from an endpush block and sends the answer to a push block.

Forward block

A forward block just forwards data (including requests and answers). An instance of this block is created the first time data should be forwarded by a node. Data is retrieved, handled by a forward block (encrypted, reordered ...) and forwarded towards another node. We distinguish between a simple forward block and a bidirectional forward block.

- Simple forward block. This variant (see Fig. 4.10.) consists of an endpush block and a push block. The information that the block receives is just forwarded. A file can be forwarded to another node.
- Bidirectional forward block. This is a composition of a reply block and a pull block (see Fig. 4.11.). A node sends a request to a forward block. That block must forward the request, retrieve the answer and send the answer to the original node.

67
We define the constructor and method of a forward block:

- **Forward()**. The creation of a forward block.
- **forwardData()**. Forwarding incoming data.

### 4.5.4 Connection level blocks

Data is sent over connections. Connections can be categorized in two ways: synchronous connections versus asynchronous connections and anonymous connections versus connections that are not anonymous. These two categories are orthogonal to each other.

- **Synchronous connection**. Data can be sent in two ways over a synchronous connection. Each endpoint of a synchronous connection can send and retrieve data. A synchronous connection is created by a pull block if the request and answer are sent over the same connection.

- **Asynchronous connection**. With an asynchronous connection, one endpoint sends data and the other endpoint receives data. A push block
can make use an asynchronous connection. Remark that a pull block can also send the request over an asynchronous connection. The reply block should send the answer to that request over another asynchronous connection. The initiator of the request must append reply information in addition to the request so that the reply block can set up a connection to the initiator of the request.

- Non-anonymous connection. A connection is non-anonymous if the endpoints reveal information about their identities.

- Anonymous connection. A connection is anonymous if the endpoints of a connection are anonymous to each other.

Remark that the connection layer does not necessarily support anonymous connection. Then, the anonymity requirements should be achieved by the peer layer. If we want to provide anonymous connections, a existing implementation can be inserted at this level such as onion routing, Crowds, Hordes ... However, we can implement our own anonymous connection system by making a good composition of basic blocks that are in the anonymity library. A static method is provided to set up connections. This method needs a nodeLocator (for instance, an onion) to set up a connection to another node in the peer system. The interface is outlined below:

- static ConnectionIdentifier setup(NodeLocator nodeLocator).
- Connection(NodeLocator nodeLocator).
- send(Data data).
- Data receive(Data data).
- close().
Chapter 5

Detailed description of demonstrators

This chapter provides a more detailed description of the implementation of the web banner system for privacy-preserving targeted advertising (section 5.1), and the peer-to-peer framework (section 5.2).

5.1 Targeted Advertising

In this section we describe the implementation of the demonstrator of a targeted advertising system. We describe the entities that participate in the system and their tasks. We also give some information about implementation details.

5.1.1 Web pages

The web pages contain a link to the banner server. When a user accesses a page that contains a banner provided by the banner server, the page sends to the user a link to the banner server. A parameter called infotype is included in the link. This parameter is used by the banner server to know to which category belongs the page (e.g., "COMPUTERS", "SPORTS", "BOOKS", etc.).

A list of links to all the pages that contain banners can be found in: http://www.cosic.esat.kuleuven.ac.be/banners/.

The banner server takes a decision on the banner to send to the user that is a function of the profile (taking into account the history and the declared information) of the user and the category of the page.
5.1.2 Client

The user is running a proxy at the client side. This proxy intercepts all the information between the browser and the web. When the proxy detects an access to the banner server it calls a Java program that creates the profile to be sent to the banner server. Once this proxy has created the appropriate headers containing the information expected by the banner server, it sends the information through another proxy that connects with a mix network [JAP]. The mix network is used to achieve anonymity at the connection level.

The user maintains locally information about the last 1,000 accesses. This information is used by a Java program at the client side to construct a simplified profile that is sent to the banner server. The simplified profile contains the three categories that appear more often in the user’s history. This simplification of the profile is done for two reasons: first, because the full profile would give too much information about the user (the number of combinations is too large, so every user would be distinguishable from the others and, therefore, traceable); and second, to improve the performance, because this way less bytes of information are sent, and the algorithm to choose a banner at the banner server is simplified. The banner server does not need to know the full history of the user to take a decision on the banner to send, it is enough to know which are the categories of more interest for the user.

Also, the user can choose to provide some declared information, such as age range, country, interests and gender. The user can change this information at any time if he desires to do so, and he can also choose which information he wants to provide, if any.

The user can access his profile information through a graphical interface. This graphical interface shows to the user the information that is actually being sent in the cookie. He can change the declared information fields, delete the history and change the weight that the items of the history have. With this last action, he can give more or less relative weight to the more recently visited web pages.

In the first version of the implementation (version 0.1) the cookie sent to the banner server has the following format:


where:

- 01: is the number of the version,
- PROFILE: is the type of message (this one contains a profile),
• $C1$ value: value contains the category with more weight in the history of the user,

• $C2$ value: value contains the category which is second in weight in the history of the user,

• $C3$ value: value contains the category which is third in weight in the history of the user,

• $I1$ value: value contains the first interest of the user,

• $I2$ value: value second interest of the user,

• $I3$ value: value third interest of the user,

• $A$ value: value age range,

• $C$ value: value country,

• $G$ value: value gender,

• $O$ value: value other information.

The fields $C1$, $C2$ and $C3$ will have an empty value if the history of the user does not contain enough items.

The fields $I1$, $I2$, $I3$, $A$, $C$, $G$ and $O$ contain declared and optional information, if the user did not choose to send it, value will be empty.

The banner server sends a response to this request. This response has the following format: "profile=01UPDATE.CATEGORY", where "CATEGORY" is the category of the page that contained the link to the banner server (the most recent page in the history of the client). The proxy calls a Java program that updates the history of the client, and forwards the banner to the browser.

The user has full control over the information that is sent to the banner server. He may change or delete his profile information by modifying the file that contains this information, manually or using the graphical user interface.

**Speed of changes of the profile**

The proxy can extract the profile from the history of the user is three different ways, depending on the relative weight of the most recent items of the history. We can define three speeds of changes as follows:
**Slow.** In this mode, all items if the history count 1 point. Each time a string appears in the array of the history, we sum a point to its respective counter, regardless of the position in the history. Eventually, the strings that have more points are put in the cookie and constitute the profile of the user.

**Medium.** Here we count the items which are more recent as having more weight. So our profile will contain information that is more representative of our recent history than the previous mode. A string that is within the 100 most recent items gets 1 point; if it is between the 101 and 200 then it gets 0.9 points, if it is between the 201 and 300 then it gets 0.8 points, and so on. The last 100 items of the history (the oldest ones) get only 0.1 points.

**Fast.** This mode makes the profile even more dynamic. The algorithm gives one point to the 10 most recent items. The 40 following items (11 – 50) get 0.5 points. The 50 following ones (51 – 100) get 0.25 points. The items with positions between 101 and 200 get 0.1 points. And the rest (201 – 1000) get only 0.01 points.

**The browser is not trusted**

Some additional security measures are taken in order to avoid that a malicious browser can build a profile of the user and, therefore, track it. In order to avoid this problem, the proxy does not let the browser see neither contents of the profile cookie sent to the banner server, nor the update information sent from the banner server to the user.

The proxy, together with the client-side programs, is the only element which can construct the profile of the user. This way a user could be using a browser in an untrusted computer and still use the proxy remotely in order to manage his profile information. At this stage, the profile information is stored in clear, but encryption should be used in later versions in order to effectively protect the private information.

Also, the proxy performs a series of security checks. For example, it makes sure that the browser is not sending other cookies with the same name (profile) that may confuse the server.
5.1.3 Banner Server

The banner server is implemented in PHP. When it gets a request from a user, it retrieves the profile contained in the cookie. After checking the format of the cookie, it decides which banner to send to the client, as a function of the category of the page that is currently being viewed by the user and the information contained in the profile.

The banner server sends to the user the selected banner and a cookie that contains the update of the user’s profile (with the category of the page that is currently viewed by the user).

The banner server logs the profile of the user together with the time and date, the page from which the request was originated and the banner that has been sent to the user.

Algorithm to select the banner

In this first version, the algorithm used is quite simple. We give points to each category, depending on the field in which it is included, in particular:

- $C1$: 3 points.
- $C2$: 2 points.
- $C3$: 1 point.
- $I1$: 4 points.
- $I2$: 3 points.
- $I3$: 2 points.
- Category of the page being currently viewed: 5 points.

Note that, in this version, we do not use the values of the other fields of the profile in order to take a decision. This could be done if, for example, one banner is considered interesting for people within a particular range of age.

If a field (e.g., $I3$) does not contain any value, it will not be taken into account. If the same category appears in $C1$ and $I2$, for example, it will have 6 points. Then we choose the category that has more points and we select the banner that is associated with it. This association is kept in a file that can be modified by the banner server administrator whenever it is needed. If the chosen category does not appear in the file, then the next one
with more points will be chose, and the procedure is repeated. Eventually, if none of the categories is present in the file, a default banner will be sent to the user.

Users who do not send a profile (this can be due to the fact that it is the first time that they access the system or because they do not use the proxy) will get a banner as a function of the category of the page that contains the link to the banner server.

5.2 Peer to peer (P2P)

5.2.1 Overview of the application

For the demonstrator, a chat application is built on top of the P2P architecture. In this section, we give an overview of the functionalities of the application and its anonymity requirements. In the next sections, we discuss four scenarios.

In the chat application, we distinguish three central concepts of the application domain: chat sessions, participants of a chat session and files that are related with a chat session. We discuss these units:

- **Chat session.** A chat session is a unit that has two unitLocators. The first unitLocator is used by other nodes that want to participate into the chat session. When a node registers to the chat session, it receives its unitIdentifier for that chat session and a second unitLocator. This unitLocator contains a path to the participants of the chat session and a path to the files that are related to the chat session. As the number of participants or files grows, the unitLocator is also extended.

- **Participant.** When a node wants to participate into a chat session, it creates a unit. The node disposes a unitLocator in its registration phase and receives its unitIdentifier from the chat session administrator.

- **File.** Participants can also dispose files and make them part of the chat session. If a chat session is set up to discuss a medical problem, a participant can dispose an interesting paper about that topic. The participant splits the file and pushes each part of the file to a different node. At the same time, the participant sends a locator for the file to the other participants. The chat session unitLocator of each participant is extended with a file unitLocator.
Every client that wants to participate in the P2P system starts up a node. At this time, the node is anonymously registered at a central server. For the demonstrator, we use the central server model. The new node sends a nodeLocator over an anonymous connection. The central server returns a unique identifier for that new node and information about the other nodes and units. Other nodes are informed about the new node.

These methods are implemented at application layer:

- **UnitIdentifier setupSession()**. A node in the system can decide to initiate a chat session. Each session is a unit in the P2P system. The session is registered at the central server by pushing a unitLocator to that server. This central server informs other nodes about the new session.

- **UnitLocator subscribe(UnitIdentifier unitIdentifier)**. A node can subscribe to a session. The request is sent to the chat session unit by using its unitLocator. The chat session unit returns an identifier for the new participant. Moreover, the chat session unit returns an identifier and locator for each participant and file in the session. It also informs other participants about the new user.

- **sendMessage(Data data, Identifier identifier)**. If a user wants to send a message to a participant in the chat session, it uses its identifier. The identifier is mapped to a locator at peer level where the message is pushed to the other participant. If we don’t specify any identifier, the message is broadcast to every participant of the current chat session.

- **Data receiveMessage()**. This method is called when a message arrives from another user.

- **sendFile(File file)**. A client can also anonymously dispose files that are associated with a particular session.

- **File retrieveFile(UnitIdentifier)**. Every node that participates the session can retrieve a file with the corresponding unitIdentifier.

In the following sections, we discuss each of these operations in more detail. We discuss what happens at peer layer when a chat session is set up, how a node can subscribe to a session, how messages are exchanged between nodes and how files are stored in the peer system.
5.2.2 Registration of a node.

At the registration phase, the node sends a registration request to the central server. The new node gives a nodeLocator to the central server and the server returns a unique identifier for that node. The central server stores the nodeLocator and the nodeIdentifier in its mapping block and informs the new user about the other nodes by setting up an anonymous connection to it. For this purpose, it uses the locator the new node gave to the server. It also informs the other nodes about the new node.

5.2.3 Setting up a chat session.

If a node sets up a new chat session, it pushes its unitIdentifier and unitLocator to the other nodes in the system. This way, the other nodes are informed about the new unit. When a session is started up, only one nodeLocator is associated with the session. This is the node that has started up the session.

5.2.4 Subscribing to a session.

If a new node subscribes to the session, it pushes a participation request to the chat session unit by using its unitLocator. The new participant sends its nodeIdentifier to the unit. The unit informs the other participants about the other node by sending the nodeIdentifier to them. The unit also informs the new node about the other participants and files in the session.
5.2.5 Exchanging messages between nodes.

When nodeIdentifiers are associated with a session, nodes can exchange messages. At application level, the node composes a message and adds an identifier. This identifier is mapped to a locator at peer level. If there is some open connection to that node, the message is pushed over that connection. Otherwise, a new connection is created.

5.2.6 Storage of information in the peer system.

If a node wants to dispose a file into the peer system, a push block (Push1 on figure) first splits the file in several parts and asks the mapping block for locators. Then, each part of the file (with a corresponding identifier) is pushed to a different node by another type of push block (Push2). Each node stores a part of the file in its storage block. The locator for the file is pushed (Push2 type) to the corresponding chat session. This unit pushes the locators for the file to the participants of the chat session. Each node that wants to retrieve the file uses the unitLocator. Remark that we have two instances of push blocks in the system. The first push block just splits a unit and a second push block actually sends the file to the right node.
Figure 5.2: File storage in the peer system.
Chapter 6

Conclusions and future work

6.1 Conclusions

Automated tools can help in guaranteeing and assessing the privacy of applications based on various input, both predefined and user dependent. Unfortunately, the complexity of such a tool is very high and until now we could not construct a satisfying solution yet.

We have proposed several tools at a theoretical and practical level. These tools are building blocks that can be used by different applications.

Using the building blocks mentioned above, we have designed and implemented two demonstrators. One of them implements a privacy-preserving web banner system and the other one a peer-to-peer system.

As it has been shown with the privacy preserving web banner system, it is possible to provide customized services while protecting user’s privacy.

Users should have control on their personal data, and be able to choose which data they want to provide to other parties.

The P2P architecture captures the functionality of a typical peer to peer system in a more general way. As such, we are able to use a peer to peer system for different types of applications. Moreover, different peer to peer implementations can be used as privacy subsystem for the same application.

Through the construction of this architecture, we gained a clear understanding in the structure and implementation of several peer to peer systems, which will stimulate their evaluation and reuse.

An anonymous communication system is needed for both demonstrators. This system provides anonymity at the connection level, protecting information such as the IP address of the computer of the user.
6.2 Future work

6.2.1 Methodology

Regarding the composition methodology of building blocks, we have not completed the analysis phase of this problem. Due to the complexity of this problem, we cannot accurately predict the effort needed to complete this task. However, we want to stress the importance and applicability of such methodology and we hence want to continue this research track during the following years.

6.2.2 Privacy-preserving targeted advertising system

For the proposed targeted advertising system, we propose the following enhancements:

- User profiles could be stored in P3P format [P3P]. The client proxy could be enhanced with generic P3P processing capability.

- In order to make this design compatible with existing banner systems, an intermediate banner server could be implemented. The task of this banner server would be to map the profile to a unique ID (this does not mean that each user would be assigned a unique ID, but that each group of users with the same profile would be considered as a single user by the banner server).

- The data of the user can be stored in an encrypted form, and the user could authenticate himself in order to access this data. These mechanisms add extra protection to the data of the user.

- The proxy could be extended to filter other private information. Also, it could act as an access point to the anonymous communication system (that is, implement the functionality currently provided by the JAP Proxy).

- Malicious web sites could fake requests in order to obtain money from the banner server. Client puzzles, blinded tickets, etc, are mechanisms that are useful in many applications in which usage restrictions are enforced while still guaranteeing anonymity.
6.2.3 Peer-to-peer networks

The demonstrator was limited to a single application and a single type of peer to peer system. In order to better assess the current architecture, an implementation of other application types and an integration of other peer to peer systems would be very useful.

One particularly interesting building block is the trust block. While some peer to peer systems already have some notion of trustworthiness regarding other peers, this is still a problem with no adequate general solution yet. In the future, we would like to study this problem in more depth and develop new techniques to tackle this problem.
Bibliography


