DIMACS Technical Report 97-15
April 1997

Crowds: Anonymity for Web Transactions

(Preliminary Announcement)

by

Michael K. Reiter[1] Aviel D. Rubin[2]

AT&T Labs—Research, Murray Hill, New Jersey, USA
{reiter,rubin}@research.att.com

[1]Permanent Member
[2]Permanent Member

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs, Bellcore, and Bell Labs.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

In this announcement we introduce a system called Crowds for protecting users' anonymity on the world-wide-web. Crowds, named for the notion of "blending into a crowd", operates by grouping users into a large and geographically diverse group (crowd) that collectively issues requests on behalf of its members. Web servers are unable to learn the true source of a request because it is equally likely to have originated from any member of the crowd, and indeed collaborating crowd members cannot distinguish the originator of a request from a member who is merely forwarding the request on behalf of another. Our security analysis introduces *degrees of anonymity* as an important tool for describing and proving anonymity properties.

1 Introduction

The lack of privacy for transactions on the world-wide-web (WWW), or the Internet in general, is a well-documented fact (see [Bri97]). While encrypting communication to and from web servers (e.g., using SSL [SSL]) can hide the content of the transaction from an eavesdropper (e.g., an Internet service provider, or a local system administrator), the eavesdropper can still learn the IP address of the client and server machines, the length of the data being exchanged, and the time and frequency of exchanges. Encryption also does little to protect the privacy of the client from the server. A web server can record the Internet addresses at which its clients reside, the servers that referred the clients to it, and the times and frequencies of accesses by its clients. With additional effort, this information can be combined with other data to invade the privacy of clients even further. For example, by automatically `finger`ing the client machine shortly after an access from that machine and comparing the idle time for each user of the client machine with the server access time, the server administrator can deduce the exact user with high likelihood.

In this paper we introduce a new approach for increasing the privacy of web transactions and a system, called *Crowds*, that implements it. Our approach is based on the idea of "blending into a crowd", i.e., hiding one's actions within the actions of many others. To execute web transactions in our model, a user first joins a "crowd" of other users. The user's initial request to a web server is first passed to a random member of the crowd. That member can either submit the request directly to the end server or pass it to another randomly chosen member, and in the latter case the next member makes its choice independently. When the request is eventually submitted, it is submitted by a random member, thus preventing the end server from identifying its true initiator. Even crowd members cannot identify the initiator of the request, since the initiator is indistinguishable from a member that simply passed on a request from another.

In studying the anonymity properties provided by this simple mechanism, we introduce the notion of *degrees* of anonymity. We argue that the degree of anonymity provided against an attacker can be viewed as a continuum, ranging from no anonymity to complete anonymity and having several interesting points in between. We informally define these intermediate points, and for our "crowds" mechanism described above, we refine these definitions and *prove* anonymity properties for our system. We expect these definitions and proofs to yield insights into proving anonymity properties for other approaches, as well.

An intriguing property of Crowds is that a member of a crowd may submit requests initiated by other users. This has both negative and positive consequences. On the negative side, the user may be incorrectly suspected of originating that request. On the positive side, this property suggests that the mere availability of Crowds offers the user some degree of deniability for her observed browsing behavior, if it is possible that she was using Crowds. Moreover, if Crowds becomes widely adopted, then the presumption that the machine from which a request is received is the machine that originated the request will become decreasingly valid (and thus decreasingly utilized).

As already mentioned, we have realized our approach in a system called *Crowds*. We describe and motivate the design of the system here; the performance and scalability of the system will be detailed in the forthcoming full version of this paper. A considerable amount of our present effort involves tuning our system to achieve peak performance, while retaining an implementation that is highly portable.

The rest of this paper is structured as follows. In Section 2, we more precisely state the anonymity goals of our system and introduce the notion of *degrees* of anonymity. This gives us sufficient groundwork to compare our approach to other approaches to anonymity in Section 3. We describe and analyze the basic crowds mechanism in Section 4. We then discuss several implementation issues in Section 5.

2 Goals

2.1 Anonymity

As discussed in [PW87], there are three types of anonymous communication properties that can be provided: sender anonymity, receiver anonymity, and unlinkability of sender and receiver. *Sender anonymity* means

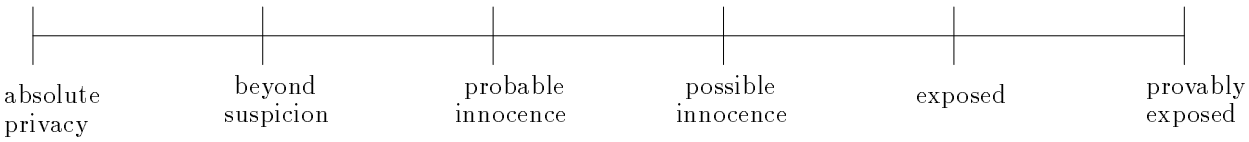

Figure 1: **Degrees of anonymity:** Degrees range from *absolute privacy*, where the attacker cannot perceive the presence of communication, to *provably exposed*, where the attacker can prove the sender, receiver, or their relationship to others.

that the identity of the party who sent a message is hidden, while its receiver (and the message itself) might not be. *Receiver anonymity* similarly means that the identity of the receiver is hidden. *Unlinkability of sender and receiver* means that though the sender and receiver can each be identified as participating in some communication, they cannot be identified as communicating *with each other*.

A second aspect of anonymous communication is the attackers against which these properties are achieved. The attacker might be an eavesdropper that can observe some or all messages sent and received, collaborations consisting of some senders, receivers, and other parties, or variations of these [PW87].

To these two aspects of anonymous communication, we add a third: the *degree* of anonymity. As shown in Figure 1, the degree of anonymity can be viewed as an informal continuum. For simplicity, below we describe this continuum with respect to sender anonymity, but it can naturally be extended to receiver anonymity and unlinkability as well. On one end of the spectrum is *absolute privacy*: absolute sender privacy against an attacker means that the attacker can in no way distinguish the situations in which a potential sender actually sent communication and those in which it did not. That is, sending a message results in no observable effects for the attacker. On the other end of the spectrum is *provably exposed*: the identity of a sender is provably exposed if the attacker cannot only identify the sender, but can also prove the identity of the sender to others.

For the purposes of this paper, three intermediate points of this spectrum are of interest. One is *beyond suspicion*, which is slightly weaker than absolute anonymity. A sender's anonymity is beyond suspicion if though the attacker can see evidence of a sent message, the sender appears no more likely to be the originator of that message than any other potential sender in the system. A weaker degree is *probable innocence*; a sender is probably innocent if the sender appears no more likely to be the originator than not to be the originator. This is weaker than beyond suspicion in that the attacker may have reason to expect that the sender is more likely to be responsible than any other potential sender, but it still appears just as likely that the sender is not responsible. A weaker degree still is *possible innocence*; a sender is possibly innocent if, from the attacker's point of view, there is a nontrivial probability that the real sender is someone else. It is possible to describe these intermediate points for receiver anonymity and sender/receiver unlinkability, as well. When necessary, we define these intermediate points more precisely.

The default level of anonymity on the web for most information and attackers is *exposed*, as described in Section 1. Netscape Navigator and Internet Explorer are configured to automatically identify the client machine to web servers, by passing information including IP address, the host architecture, and the operating system in the request header. They also allow web servers to link multiple visits to the same sender, using "cookie" values that are stored on the client's machine.

2.2 What Crowds achieves

As described in Section 1, our system consists of a dynamic collection of users, called a *crowd*. These users initiate web requests to various web servers (and receive replies from them), and thus the users are our senders and the servers are our receivers. We consider the anonymity properties provided to an individual user against three distinct types of attackers:

- A **local eavesdropper** is an attacker who can observe all (and only) communication to and from the user's machine.
- **Collaborating crowd members** are other crowd members that can pool their information and even deviate from the prescribed protocol.
- The **end server** is the web server to which the web transaction is directed.

A local eavesdropper models, e.g., an eavesdropper on the local area network of the user, such as an administrator monitoring web usage at a local firewall. However, if the same LAN also serves the end server, then the eavesdropper is effectively global, and we provide no protections against it.

Table 1: Anonymity properties provided by Crowds

Attacker	Sender anonymity	Receiver anonymity
local eavesdropper	exposed	$P(\text{beyond suspicion}) \underset{n\to\infty}{\longrightarrow} 1$
c collaborating members, $n \geq \frac{p_f}{p_f - 1/2}(c+1)$	**probable innocence** $P(\text{absolute privacy}) \underset{n\to\infty}{\longrightarrow} 1$	exposed
end server	**beyond suspicion**	N/A

The security offered against each of these types of attackers is summarized in Table 1 and justified in the remainder of the paper. As indicated by the omission of an "unlinkability of sender and receiver" column from this table, our system serves primarily to hide the sender or receiver from the attacker. In this table, n denotes the number of members in the crowd and $p_f > 1/2$ denotes the probability of forwarding, i.e., when a crowd member receives a request, the probability that it forwards the request to another member, rather than submitting it to the end server. The boldface claims in the table are guarantees. The probability that a receiver's anonymity is beyond suspicion against a local eavesdropper, on the other hand, only increases to 1 asymptotically as the crowd size increases to infinity. Put another way, if the local eavesdropper is sufficiently lucky, then it observes events that expose the receiver of a web request, and otherwise the receiver is beyond suspicion. However, the probability that it views these events decreases as a function of the size of the crowd. Similarly, a sender's assurance of absolute privacy against collaborating members also holds asymptotically with probability one as crowd size grows to infinity (for a constant number of collaborators). Thus, if the collaborators are unlucky, users achieve absolute privacy. We provide a more careful treatment of these notions in Section 4.2.

Of course, against an attacker that is comprised of two or more of the attackers described above, our system yields degrees of sender and receiver anonymity that are the minimum among those provided against the attackers present. For example, if a local eavesdropper and an end server collaborate in an attack, then our techniques achieve neither sender anonymity nor receiver anonymity. Another caveat is that all of the claims of sender and receiver anonymity in this section, and their justifications in the remainder of this paper, require that message contents themselves do not reveal the sender's or receiver's identity.

2.3 What Crowds does not achieve

Crowds makes no effort to defend against denial-of-service attacks by rogue crowd members. A crowd member could, e.g., accept messages from other crowd members and refuse to pass them along. In our system, such denial-of-service can result from malicious behavior, but typically does not result if (the process representing) a crowd member fails benignly or leaves the crowd. As a result, these attacks are detectable. Less detectable are active attacks where crowd members substitute wrong information in response to queries that they receive from other crowd members. Such attacks are inherent in any system that uses intermediaries to forward information, but fortunately they cannot be utilized to compromise anonymity directly.

3 Related work

There are two basic approaches previously proposed for achieving anonymous web transactions. The first approach is to interpose an additional party (a *proxy*) between the sender and receiver to hide the sender's identity from the receiver. The most popular such proxy today is the Anonymizer web server (`http://www.anonymizer.com/`). Crowds provides protection against a wider range of attackers than proxies do. In particular, proxy-based systems are entirely vulnerable to a passive attacker in control of the proxy, since the attacker can monitor and record the senders and receivers of all communication. Our system presents no single point at which a passive attack can cripple all users' anonymity. In addition, a proxy is typically a single point of failure; i.e., if the proxy fails, then anonymous browsing cannot continue. In Crowds, no single failure will discontinue all ongoing web transactions.

A second approach to anonymous web transactions is to use a *mix* [Cha81]. A mix is actually an enhanced proxy that, in addition to hiding the sender from the receiver, also takes measures to provide sender and receiver unlinkability against a global eavesdropper. It does so by collecting messages of equal length from senders, cryptographically altering them (typically by decrypting them with its private key), and forwarding the messages to their recipients in a different order. These techniques make it difficult for an eavesdropper to determine which output messages correspond to which input messages. A natural extension is to interpose a *sequence* of mixes between the sender and receiver [Cha81]. A sequence of mixes can tolerate colluding mixes, as any single correctly-behaving mix server in the sequence prevents an eavesdropper from linking the sender and receiver. Mixes have been implemented to support many types of communication, for example electronic mail (e.g., [GT96]), ISDN service [PPW91], and general synchronous communication (including web browsing) [SGR97].

The properties offered by Crowds is different from those offered by mixes. As described above, Crowds provide (probable innocence) sender anonymity against collaborating crowd members. In contrast, in the closest analog to this attack in the mix model—i.e., a group of collaborating mix servers—mixes do not provide sender anonymity but do ensure sender and receiver unlinkability. Another difference is that mixes provide sender and receiver unlinkability against a global eavesdropper. Crowds does not provide anonymity against global eavesdroppers. However, our intention is for a crowd to span multiple administrative domains, where the existence of a global eavesdropper is unlikely. Another difference is that mixes typically rely on public key encryption, the algebraic properties of which have been exploited to break some implementations [PP90].

Crowds unique properties admit very efficient implementations that typically outperform mixes. With mixes, the length of a message routed through a mix network grows proportionally to the number of mixes through which it is routed. More importantly, in a typical mix implementation, routing a message through a sequence of n mixes incurs a cost of n public key encryptions and n private key decryptions on the critical path of the message, which are comparatively expensive operations. Thus, since the unlinkability provided by mixes is tolerant of up to $n-1$ mixes colluding, increasing n improves anonymity but hurts performance. Privacy in Crowds can similarly be enhanced by increasing the number of times a request is forwarded among members before being submitted to the end server, but this typically impacts performance less because there are no public/private key operations and no inflation of message transmission lengths (beyond a small, constant-size header).

Another performance advantage of Crowds is that since each user actively participates in the function of the crowd, the throughput of a crowd grows in proportion to the number of users. As we will demonstrate in the full paper, this enables a crowd to scale almost limitlessly, in the sense that performance as seen by users remains roughly constant as new users join the crowd. With a fixed network of mixes, the performance of the system degrades with more users due to increased load.

4 Crowd basics

4.1 Overview

As discussed previously, a crowd can be thought of as a collection of users. A user is represented in a crowd by a process on her local machine called a *jondo* (pronounced "John Doe" and meant to convey the image of a faceless participant). The user (or a local administrator) starts the jondo on the user's machine, and then the user selects this jondo as her web proxy by specifying the machine and port number of the jondo in her web browser as the proxy for all services. Thus, any request coming from the browser is sent directly to the jondo.[1]

After a jondo is started and joins the crowd (this is discussed in Section 5.1), it initiates the establishment of a random *path* of jondos via which it carries its users' transactions to and from their intended web servers. More precisely, upon receiving the first user request from a browser, the jondo picks a jondo from the crowd (possibly itself) at random, and sends it the request. When this jondo receives the request, it flips a biased coin to determine whether or not to forward the request to another jondo. If the result is to forward, then the jondo selects a random jondo and forwards the request to it, and otherwise the jondo submits the request to the end server for which the request was destined. So, each request travels from the user's browser, through some number of jondos, and finally to the end server. A possible set of such paths is shown in Figure 2. Subsequent requests initiated at the same jondo follow the same path (except perhaps going to a different end server), and server replies traverse the same path as the requests, only in reverse.

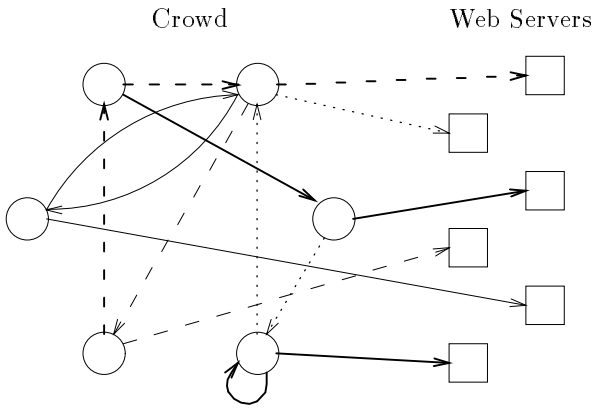

Figure 2: Paths in a crowd

A pseudocode description of a jondo is presented in Figure 3. This figure describes a thread of execution that is executed per received request. This description uses client-server terminology, where one jondo is a *client* of its successor (*server*) on the path, which might be a jondo or the destination web server. In particular, for each path, indicated by a *path id*, the value server[path_id] is the next hop on the path. To assign next hops for paths, each jondo maintains a set *Jondos* of jondos that it believes to be active (itself included). When it chooses to direct the path to another jondo, it selects the next hop uniformly

[1] The services that must be proxied are gopher, ftp, HTTP and SSL. Otherwise, e.g., ftp requests triggered by downloading a web page would not go through the crowd, and would identify the user. Java and Javascript must be disabled in the browser as well because, e.g., an applet could connect back to its server and identify the user.

```
(1)     client,request ← receive_request()
(2)     if (client = browser)
(3)         sanitize(request)                                  /* strip cookies and identifying headers */
(4)         if (my_path_id = ⊥)
(5)             my_path_id ← new_path_id()
(6)             server[my_path_id] ←R Jondos
(7)         send my_path_id||request to server[my_path_id]     /* send to first jondo */
(8)     else                                                   /* client is a jondo */
(9)         path_id ← remove_path_id(request)
(10)        if (path_id ∉ PathIds)                             /* new path id */
(11)            PathIds ← PathIds ∪ {path_id}
(12)            translate[path_id] ← new_path_id()             /* set "outgoing" path id */
(13)            coin ← coin_flip(p_f)                          /* tails with probability p_f */
(14)            if (coin = heads)
(15)                server[path_id] ← destination(request)     /* set next hop to web server */
(16)            else
(17)                server[path_id] ←R Jondos                  /* set next hop to random jondo */
(18)                request ← translate[path_id]||request      /* prepend path id to request */
(19)        else                                               /* path already exists */
(20)            request ← translate[path_id]||request
(21)        send request to server[path_id]
(22)    if (server[path_id] ∈ Jondos)
(23)        reply ← await_reply(∞)                             /* wait for reply or recognizable jondo failure */
(24)        if (reply = 'jondo failed')                        /* jondo failed */
(25)            send 'no reply' to client
(26)            Jondos ← Jondos \ {server[path_id]}            /* remove the jondo */
(27)            server[path_id] ←R Jondos                      /* assign a new random jondo for this path */
(28)        else                                               /* received reply from jondo */
(29)            send reply to client
(30)    else                                                   /* made request directly to web server */
(31)        reply ← await_reply(timeout)                       /* wait for reply, timeout, or recognizable server failure */
(32)        send reply to client                               /* send reply, timeout, or failure message to client */
```

Figure 3: Pseudocode description of a jondo

at random from this set (lines 6, 17, and 27); i.e., "$\leftarrow_R S$" denotes selection from the set S uniformly at random. Subsequent sections shed greater light on the operation of a jondo and the pseudocode description of Figure 3.

For technical reasons, it is convenient for the jondo at each hop in a path to hold a different path identifier for the path. That is, if a jondo receives a request marked with path_id from its predecessor in a path, then it replaces path_id with a different path identifier stored in translate[path_id] before sending the request to its successor (if a jondo). This enables a jondo that occupies multiple positions on a path to act independently in each position. If the path_id remained the same along the path, then the jondo would behave identically each time it received a message on the path, resulting in an infinite loop. Path identifiers should be globally unique; in our present implementation, new_path_id() returns a random 128-bit value.

Omitted from the description in Figure 3 is that fact that all communication on a path is encrypted with a *path key* (see Section 5.2), which all jondos on the path possess.

4.2 Security analysis

In this section we consider the question of what information an attacker can learn about the senders and receivers for web transactions, given the mechanisms we have described so far in Section 4.1. The types of attackers we consider were described in Section 2.

Local eavesdropper The mechanisms we have described typically prevent a local eavesdropper from learning the intended receiver of a request, because every message sent on a path, except for the final request to the end server, is encrypted with a secret path key. Thus, while the eavesdropper is able to view any message emanating from the user's machine, it only views a message sent to the end server (or equivalently a plaintext message containing the end server's address) if the user's jondo ultimately submits the user's request itself. Since the probability that the initiator's jondo ultimately submits the request is $1/n$ where n is the size of the crowd when the path was created, the probability that the eavesdropper learns the receiver decreases as a function of crowd size. Moreover, when the initiator's jondo does not ultimately submit the request, the local eavesdropper sees only the encrypted address of the end server, which we suggest yields receiver anonymity that is (informally) beyond suspicion.

The initiator herself is exposed to a local eavesdropper, since we make no effort to hide correlations between inputs to and outputs from the initiating machine; i.e., the local eavesdropper observes that a request output by the initiator's machine did not result from a corresponding input.

Collaborating jondos Consider a set of collaborating (corrupted) jondos in the crowd. A single malicious jondo is simply a special case of this attacker, and our analysis applies to this case as well. Because each jondo can decrypt traffic on a path routed through it, any such traffic, including the address of the end server, is exposed to this attacker. The question we consider here is if the attacker can determine who initiated the path, and we are able to answer it with some positive (for the user) results.

To be precise, consider any path which is initiated by a non-collaborating member and on which a collaborator occupies a position. The goal of the collaborators is to determine the member that initiated the path. Assuming that the contents of the communication do not suggest an initiator, the collaborators have no reason to suspect any member other than the one from which they immediately received it, i.e., the member immediately preceding the first collaborator on the path. All other members are each equally likely to be the initiator, but are also obviously less likely to be the initiator than the collaborators' immediate predecessor. We now analyze how confident the collaborators can be that their immediate predecessor is in fact the path initiator.

Let H_k denote the event that the first collaborator on the path of a message occupies the kth position on the path, where the initiator itself occupies the 0th position (and possibly others), and define $H_{k+} = \bigvee_{k'>k} H_{k'}$. Let I denote the event that a collaborator on the path is immediately preceded on the path by the path initiator. Given this notation, the collaborators now hope to determine $P(I|H_{1+})$, i.e., given that a collaborator is on the path, what is the probability that the path initiator is the first collaborator's immediate predecessor? Refining our intuition from Section 2, we say that this immediate predecessor (and thus all noncollaborating members) have probable innocence if this probability is at most 1/2.

Definition 4.1 *Each noncollaborator has* probable innocence *(with respect to sender anonymity) if* $P(I|H_{1+}) \leq 1/2$.

In order to yield probable innocence for noncollaborators, certain conditions must be met in our system. In particular, let $p_f > 1/2$ be the probability of forwarding in the system (see Section 4.1), let c denote the number of collaborators in the crowd, and let n denote the total number of crowd members. The theorem below gives a sufficient condition on p_f, c, and n to ensure probable innocence for all noncollaborators.

Theorem 4.2 *If* $n \geq \frac{p_f}{p_f - 1/2}(c+1)$, *then all noncollaborators have probable innocence against* c *collaborators.*

Proof : Again, we want to show that $P(I|H_{1+}) \leq 1/2$ (if $n \geq \frac{p_f}{p_f - 1/2}(c+1)$). First note that

$$P(H_i) \;=\; \left(\frac{p_f(n-c)}{n}\right)^{i-1}\left(\frac{c}{n}\right)$$

This is due to the fact that in order for the first collaborator to occupy the ith position on the path, the path must first wander to $i-1$ noncollaborators (each time with probability $\frac{n-c}{n}$), each of which chooses to forward the path with probability p_f, and then to a collaborator (with probability $\frac{c}{n}$). The next two facts follow immediately from this.

$$P(H_{2+}) = \frac{c}{n}\sum_{k=1}^{\infty}\left(\frac{p_f(n-c)}{n}\right)^k = \left(\frac{c}{n}\right)\left(\frac{\frac{p_f(n-c)}{n}}{1-\frac{p_f(n-c)}{n}}\right) = \frac{p_f c(n-c)}{n^2-p_f n(n-c)}$$

$$P(H_{1+}) = \frac{c}{n}\sum_{k=0}^{\infty}\left(\frac{p_f(n-c)}{n}\right)^k = \left(\frac{c}{n}\right)\left(\frac{1}{1-\frac{p_f(n-c)}{n}}\right) = \frac{c}{n-p_f(n-c)}$$

Other probabilities we need are $P(H_1)=\frac{c}{n}$, $P(I|H_1)=1$, and $P(I|H_{2+})=\frac{1}{n-c}$. The last of these follows from the observation that if the first collaborator on the path occupies only the second or higher position, then it is immediately preceded on the path by any noncollaborating member with equal likelihood. Now, $P(I)$ can be captured as

$$P(I) = P(H_1)P(I|H_1)+P(H_{2+})P(I|H_{2+}) = \frac{c(n-np_f+cp_f+p_f)}{n^2-p_f n(n-c)}.$$

Then, since $I \Rightarrow H_{1+}$ we get

$$P(I|H_{1+}) = \frac{P(I \wedge H_{1+})}{P(H_{1+})} = \frac{P(I)}{P(H_{1+})} = \frac{n-p_f(n-c-1)}{n}$$

So, if $n \geq \frac{p_f}{p_f-1/2}(c+1)$, then $P(I|H_{1+}) \leq \frac{1}{2}$. □

As a result of Theorem 4.2, if $p_f=\frac{3}{4}$, then probable innocence is guaranteed as long as $n \geq 3(c+1)$. More generally, Theorem 4.2 implies a tradeoff between the length of paths (i.e., performance) and ability to tolerate collaborators. That is, by making the probability of forwarding high, the fraction of collaborators that can be tolerated approaches half of the crowd. On the other hand, making the probability of forwarding close to one-half decreases the fraction of collaborators that can be tolerated.

The value of $P(H_{1+})$ derived in the proof of Theorem 4.2 shows that $P(H_{1+}) \to 0$ as $n \to \infty$ if c, p_f are held constant. Assuming that collaborators cannot observe a path on which they occupy no positions, it follows that the probability of achieving *absolute privacy* for the user grows to one as the size of the crowd grows. The rate of this growth, however, can be slow if p_f is large.

End Servers We now consider the security of our system against an attack by the end server only. Because the web server is the receiver, obviously receiver anonymity is not possible against this attacker. However, the security of who initiated the session is quite strong. In particular, since the end server is never the first hop of a path (see Section 4.1), the end server is equally likely to receive each request from any group member. Therefore, we have

Theorem 4.3 *Every crowd member is beyond suspicion against an end server.*

It is interesting to note that this analysis, as opposed to that for collaborating jondos, does not require that p_f be any particular value, and indeed increasing expected path length beyond two offers no additional assurance of anonymity against an end server.

4.3 The importance of static paths

Early in the design of Crowds, we were tempted to make paths much more dynamic than they are in the present system, e.g., by having a jondo use a different path for each of its users, per time period, or even per user request. The advantages of more dynamic paths include the potential for better performance via load balancing among the crowd. In this section, however, we caution that dynamic paths tends to decrease the anonymity properties provided by the system against collaborating jondos. The reason is that the probable innocence offered by Theorem 4.2 vanishes if the collaborators are able to link multiple distinct paths as being initiated by the same user. Collaborating jondos might be able to link paths from the same unknown user based on related path content or timing of communication on paths. To prevent this, we made paths static, so the attacker simply does not have multiple paths to link to the same user.

To see why multiple linked paths initiated by the same user could compromise the user's anonymity, note that collaborating jondos have a higher probability of receiving each path initiation message from the initiator of the path than from any other individual member (see the proof of Theorem 4.2). Multiple paths from the same user therefore pinpoint the user as the one from which the collaborators most often receive the initiating messages. Put another way, if the collaborators identify paths $P_1, \ldots, P_k$ from the same (unknown) initiator, then the expected number of paths on which a collaborator is directly preceded by the path initiator is $\mu = k(\frac{n-p_f(n-c-1)}{n})$. By Chernoff's bound, the probability that the first collaborator is immediately preceded by the initiator on substantially fewer of these paths is small: the first collaborator is immediately preceded by the path initiator on fewer than $(1-\delta)\mu$ paths with probability only $e^{-\mu\delta^2/2}$. Thus, the initiator is identified with high probability.

Again, it is for this reason that a jondo sets up one path for all its users' communications, and this path is altered only when failures are detected in the path. Indeed, paths are only rerouted when the failure of a jondo is unmistakenly detected, i.e., when the jondo executes a *fail-stop* failure [SS83]. In our present implementation, such failures are detected by the TCP/IP connection to the jondo breaking or being refused. A jondo does not reroute a path based on simply timing out on the subsequent jondo in the path (see line 23 of Figure 3). While this increases our sensitivity to denial-of-service attacks (see Section 2.3), it strengthens our promise of anonymity to the user.

A reasonable question, however, is whether a malicious jondo on a path can feign its own failure in hopes that the path will be rerouted through a collaborator, yielding information that incriminates the path initiator. Fortunately, the answer is "no." If a jondo in a path fails (or appears to fail), the path remains the same up until the predecessor of that faulty jondo, who reroutes the remainder of the path randomly (line 27 of Figure 3). Since the collaborating jondos cannot distinguish whether that predecessor is the originator or not, the random choices made by that predecessor yield no additional information to the collaborators.

5 Implementation issues

5.1 Crowd membership

There are many aspects of maintaining the membership of a crowd that we still must address. The membership maintenance procedures are important because they must inform crowd members of who is in the crowd, and because they must limit access to the crowd. More precisely, from Section 4.2, it is clear that the number of collaborating jondos must be kept small relative to the total number of jondos. If anyone could set up a jondo anywhere, then a single attacker could easily launch enough collaborating jondos so that $n < \frac{p_f}{p_f - 1/2}(c+1)$, at which point Theorem 4.2 no longer offers protection. It is worth noting that a similar attack could be mounted against a typical mix network, unless there are mechanisms in place to limit what mix servers can be introduced into the network.

There are many schemes that could be adopted to manage membership of the crowd. Existing *group membership protocols*, tolerant either of benign (e.g., [Cri91, RB91, MMA91]) or malicious [Rei96] faults, can be used for maintaining a consistent view of the membership among all jondos, and the members could

use voting to determine whether an authenticated prospective member should be admitted to the crowd. Indeed, a similar approach has been adopted in prior work on secure process groups [RBvR94]. While providing robust distributed solutions, these approaches have the disadvantages of being heavyweight and of providing semantics that are arguably too strong for the application at hand. In particular, a hallmark of these approaches is a guaranteed consistent view of the group membership among the group members, whereas it is unclear whether such a strong guarantee is required here.

In our present implementation we have therefore opted for a simpler, centralized solution. Membership in a crowd is controlled and reported to crowd members by a process called the *blender*. In order to make use of the blender (or thus the crowd), the user must establish an *account* with the blender, i.e., an account name and password that the blender stores. When the user starts a jondo, the jondo and the blender use this shared password to authenticate each other's communication. As a result of that communication (and if the blender accepts the jondo into the crowd; see below), the blender adds the new jondo (i.e., its IP address, port number, and account name) to its list of members, and reports this list back to the jondo. In addition, the blender also reports back a list of shared keys, each of which can be used to authenticate another member of the crowd. The blender then sends each key to the other jondo that is intended to share it (encrypted under the account password for that jondo) and informs the other jondo of the new member.

Each member maintains its own list of the crowd membership. This list is initialized to that received from the blender when the jondo joins the crowd, and is updated when the jondo receives notices of new or deleted members from the blender. The jondo can also remove jondos from its list of crowd members, if those jondos are persistently unresponsive. This allows for each jondo's list to diverge from others' if different jondos have different perceptions of what jondos are responsive. This appears to have little qualitative effect on our security analysis of Section 4.2, unless attackers are able to delay communications between correct jondos to the extent that each removes the correct jondos from its view.

In our present implementation, the blender serves as the primary point at which access control to the group is enforced. We are experimenting with policies to limit the possibility that a large group of collaborators are present in the group at any time. In particular, we are considering a policy of allowing at most one jondo in the crowd per account name, and applying heuristics to limit the number of jondos in geographic proximity to each other. This, combined with administrator efforts to limit each user to one account name, should lessen the possibility of a substantial group of collaborators. We will report on the enforcement and success of these policies in the full paper.

A disadvantage of this approach to membership maintenance is that the blender is a trusted third party for the purposes of key distribution and accurate membership reporting. As a result, the blender is best executed on a secure machine, e.g., with login access available only at the console. However, note that web communication is *not* routed through the blender, and thus a successful passive attack on the blender does not immediately imply loss of anonymity properties (unlike the Anonymizer; see Section 3). In addition, the failure of the blender does not interfere with ongoing web transactions.

5.2 Performance

Performance has been a primary motivating factor behind the design of Crowds and, as discussed in Section 3, is one of the main factors that caused us to depart from mixes [Cha81]. Here we only briefly sketch some of the factors that determine performance in our system. An extensive discussion of performance and scalability will appear in the full paper.

The additional latency imposed by Crowds in a request/response interaction is roughly the latency associated with the path from the initiating jondo to the jondo that submits the request to the web server. In our early implementations, the latency associated with this portion of the path was dominated by the cost of encrypting and decrypting requests and replies on this path. To limit this latency as much as possible, we employ a *path key*, which is a key shared among all jondos on a path. A path key is created by the jondo initiating the path, and each jondo on a path passes it to the next jondo by encrypting the key with a key it shares with the next jondo (see Section 5.1). The existence of a path key enables requests to be encrypted at the jondo initiating the request, decrypted by the last jondo in the path, and passed by intermediate

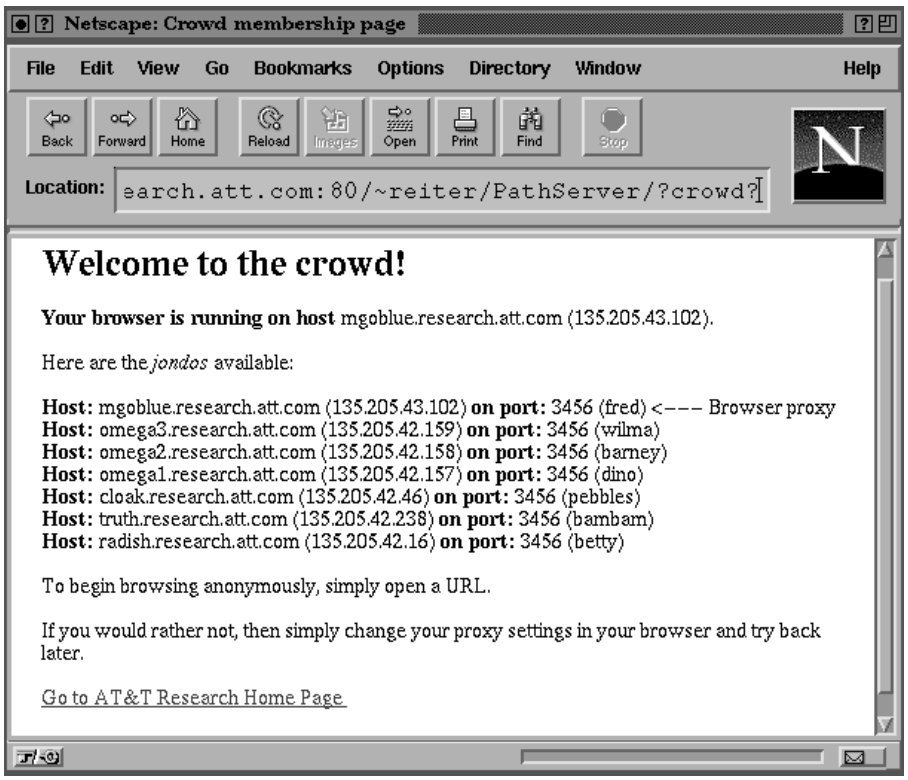

Figure 4: **Crowd query:** A crowd query shows the jondos that are available, and indicates which one is acting as the user's HTTP proxy for the browser.

jondos without encrypting or decrypting. Similarly, replies are encrypted at the last jondo in the path, and decrypted only at the jondo where the request was initiated. The cryptographic operations are performed using an efficient stream cipher, allowing the encrypting and decrypting streams for the reply to be generated while the jondos are waiting for the reply from the web server. The result is very low additional latency in a request/response interaction.

5.3 User interface

When a user points her browser to the jondo on her machine, she can specify a *crowd query*, by adding `?crowd?` to the end of any URL. This returns a list of all of the active jondos in the crowd, according to the jondo pointed to by the browser. The information includes the account name, the IP address and the port number. An example is shown in Figure 4. When a user is browsing via the crowd, the word *Crowd:* is prepended to the title of each page. Thus, a user can check whether or not she is using the crowd by looking at the tile of the documents in the browser. (Of course, an attacker can add this word to the title of any document to fool the user, and so this alone should not be relied upon.)

5.4 Firewalls

Firewalls [CB94] present a problem for Crowds. Like all network servers, jondos are identified by their IP address and port number. Unfortunately, most corporate firewalls do not allow incoming connections on ports other than a few well-known ones. Thus, firewalls represent a barrier to wide-scale inter-corporation adoption of Crowds.

Since most firewalls are configured to allow outgoing connections on any port, it is still possible for a jondo to initiate a path that goes outside the firewall and eventually to web servers. However, the firewall gives the first jondo on the path outside that domain a way to verify that the initiating machine resides within the domain; it simply tries to open a connection back to its predecessor on the path, and if that fails, then the path must have originated in the predecessor's domain. Therefore, crowd members that reside behind a firewall are not offered the same anonymity properties as those that are not.

It is conceivable that if Crowds becomes widespread, and there is demand for a special reserved port, that firewalls can open this port and allow jondos to communicate. Until then, Crowds will be most useful across academic institutions, as a service provided by Internet service providers, and within large corporations.

6 Status and future work

At the time of this writing, we have completed an initial implementation of Crowds and are experimenting with its use in AT&T. Our present implementation is approximately 700 lines of Perl code, not including implementations of cryptographic primitives and administrative functionality. We are preparing the system for release to the Internet community free-of-charge. By releasing the system, we hope to gain experience with which we can refine and expand the system.

An area of ongoing work is techniques to more effectively defend against local and global eavesdroppers. In particular, we are developing a method for mimicking the browsing behavior of a human user closely. By co-locating an implementation of this method with every web browser, we hope to gain *possible innocence* for sender anonymity, even against an attacker that narrows the initiator of a path down to a single machine. Another area of ongoing work is to generalize the Crowds paradigm to provide anonymity in other services and forms of synchronous communication.

References

[Bri97] S. Brier. How to keep your privacy: Battle lines get clearer. *The New York Times*, January 13, 1997.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM* 24(2):84–88, February 1981.

[CB94] W. Cheswick and S. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Publishing Company, 1994.

[Cri91] F. Cristian. Reaching agreement on processor group membership in synchronous distributed systems. *Distributed Computing* 4:175–187, 1991.

[GT96] C. Gulcu and G. Tsudik. Mixing email with Babel. In *Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security*, February 1996.

[MMA91] L. E. Moser, P. M. Melliar-Smith and V. Agrawala. Membership algorithms for asynchronous distributed systems. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 480–488, May 1991.

[PP90] A. Pfitzmann and B. Pfitzmann. How to break the direct RSA-implementation of MIXes. In *Advances in Cryptology—EUROCRYPT '89*, pages 373–381, Springer-Verlag, 1990.

[PPW91] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-Mixes: Untraceable communication with very small bandwidth overhead. In *GI/ITG Conference: Communication in Distributed Systems*, pages 451–463, Springer-Verlag, 1991.

[PW87] A. Pfitzmann and M. Waidner. Networks without user observability. *Computers & Security* 6(2):158–166, 1987.

[Rei96] M. K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering* 22(1):31–42, January 1996.

[RBvR94] M. K. Reiter, K. P. Birman, and R. van Renesse. A security architecture for fault-tolerant systems. *ACM Transactions on Computer Systems* 12(4):340–371, November 1994.

[RB91] A. M. Ricciardi and K. P. Birman. Using process groups to implement failure detection in asynchronous environments. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 341–351, August 1991.

[SGR97] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, May 1997.

[SS83] R. D. Schlichting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems* 3(1):222-238, August 1983.

[SSL] Kipp E. B. Hickman and Taher Elgamal. The SSL Protocol. *Internet draft draft-hickman-netscape-ssl-01.txt*, 1995.