From Chaum to Tor and Beyond:
A Survey of Anonymous Routing Systems

Peter C. Johnson, Apu Kapadia
Department of Computer Science
Dartmouth College
Hanover, NH 03755
{pete,akapadia}@cs.dartmouth.edu

03 January, 2007

Abstract
The freedoms of speech and of assembly are generally considered bastions of civil liberty. The Internet poses a particularly prickly proposition for both those who wish to communicate and assemble, and those who seek to suppress such activities. On the one hand, the Internet provides a communication medium that virtually instantaneously connects any two points on the globe; and yet the endpoints of that communication are generally trivial to deduce. The former leads to all manner of communication; the latter enables an opposing party to quell that communication via various methods: legal attacks, denial-of-service attacks, offline attacks against participants, and so on.

In an attempt to provide the anonymity necessary to ensure the freedoms of speech and assembly, research has been conducted in the area of anonymous routing. The goal of these systems is to obscure both endpoints of an Internet communication from an adversary. This paper attempts to summarize the history of the area, with an emphasis on research trends and a look towards the future.

1 Introduction
Secret communication has existed in one form or another for thousands of years: Julius Caesar used his eponymous cipher to send messages to his generals. Encryption has gotten very good at guarding the content of secret messages, but it cannot hide the identity of the communicating parties. Often, the fact that particular entities have exchanged information—regardless of its content—is a significant security concern (e.g., citizens under oppressive governments browsing websites about democracy). The expected continued growth in popularity of the Internet as a communication medium highlights the need for anonymity, and raises its profile as a target.

Research into network anonymity began with David Chaum’s proposal for untraceable e-mail in 1981 [9] and continues to this day. This paper will start by laying some groundwork in the area of anonymous systems in Sections 2–4, then proceed to a survey of research into such systems in Section 5, as well as applications built upon them in Section 6, and attacks to which they are susceptible in Section 7, and conclude with some thoughts about the area in general and where it is headed in Section 8.

2 What is anonymity?
Pfitzmann and Waidner [23] identify three classes of anonymity.

Sender anonymity indicates the inability of any party other than the sender to deduce the source of a message.

Receiver anonymity indicates the inability of
any party other than the recipient to deduce the destination of a message.

Unlinkability of sender and receiver refers to the idea that, while it may be possible to deduce that two parties are using the same system, it is not possible to deduce that they are using it to communicate with each other.

Anonymity is generally discussed relative to the size of the anonymity set: that is, the set of entities that could have sent or received a particular communication. A user has more anonymity if his anonymity set is large, and less if it is small. Therefore, it is the job of the attacker to minimize the size of the anonymity set; and it is the job of the system designer and the honest user to maximize the size of the anonymity set.

Reiter and Rubin describe anonymity as a continuum [24] indicating the probability with which an attacker can accurately match a particular communication to a sender:

- A sender has absolute privacy if an attacker cannot tell when a message is sent or not sent.
- A sender is beyond suspicion when an attacker can see that a message is sent but cannot determine if the node in question is any more likely to be the originator than any other node.
- Subtly different than beyond suspicion, a sender is probably innocent if the sender is no more likely than not to be the originator of a message.
- In the case where there is a non-trivial probability that the sender is not the node in question, that node is possibly innocent.
- A sender whose communications can be identified is exposed.
- If the attacker can demonstrate this exposure to other entities, the sender is provably exposed.

This taxonomy can also be applied to receiver anonymity and unlinkability of sender and receiver. Furthermore, a particular user’s anonymity needs will vary by situation: sometimes a user might require probable innocence, and at other times require anonymity that is beyond suspicion.

2.1 Why anonymity?

It might be argued that encryption provides sufficient protection on the Internet, and that obfuscating the source and destination of traffic is excessive. This is not so, however, given the relative ease with which users can generally be tied to IP addresses, and the potentially sensitive nature of some websites.

For example, China blocked access to the entirety of Wikipedia [26] in 2004. Instead of outright blocking at the IP level, the Chinese government could instead see which users visit Wikipedia and punish them directly. In this case, an anonymizing network would prevent users from being connected with specific websites: all the authorities would know is that users were using the anonymizing network. The extension of this—that mere participation in the anonymizing network, as opposed to a small set of websites, is actionable—is explored in Section 8.

There are certainly uses for anonymizing networks on a less grand scale, as well. Consider the situation where a home user wishes to visit a website concerned with sensitive topics such as terminal diseases or sexual preferences. It is entirely likely that such a user would prefer others not to know of these visits, but the architecture of many residential networks is such that all traffic is visible to many other customers. Even though encryption will prevent a nosy neighbor from seeing the actual content transferred, a simple DNS lookup will often reveal the website from which the pages are originating. Perhaps more insidiously, even with encryption, the website operator knows which IP addresses are accessing its content and can potentially tie specific people to specific activities on the site.

It is against these threats that anonymizing networks primarily defend, though the very
structure of many such networks present further vectors for attack.

3 Adversaries

There are three main types of adversary to consider when designing or analyzing an anonymity system.

The first is a *global passive adversary* able to examine, save, and analyze all communication on the network with effectively limitless connectivity, storage space, and computation power. The goal of the global passive adversary is generally to learn about the network: identifying users, linking users with specific communications, linking users with specific destinations, and so on. It is currently unclear how realistic an adversary this is: Tor explicitly chooses not to defend against it (Section 3 of [11]), whereas Tarzan does (Section 1 of [13]). This sort of adversary may be impractical given the network, storage, and processing capabilities required to monitor an anonymity network of any reasonable size, though it is entirely possible that some governments have the resources to conduct surveillance of this scope.

Secondly, *malicious nodes* are part of the anonymizing network itself, able to examine all traffic that passes through them and decrypt all data for which they possess keys. These nodes may either be initially honest and later compromised by attackers, or dishonest nodes inserted directly into the network by the attackers themselves. Additionally, these nodes may collude via out-of-band communication channels, allowing them to share information and correlate data to divine the endpoints or content of a particular communication. Given that most systems allow nodes to freely join, this class of adversary must be accounted for.

Finally, *malicious external nodes* may mount attacks even though they are not part of the anonymity network. Such nodes could monitor known egress points or specific destinations (e.g., websites whose users they wish to track) to perform traffic analysis (see Section 7.1), or mount denial-of-service attacks by initiating an overwhelming number of anonymous connections. While the global passive adversary may not be a practical concern, an adversary wishing to target users of a specific web site could much more easily set up a few malicious nodes to monitor incoming traffic to that website. Additionally, a website itself could qualify as a malicious external node if it attempts to undermine the anonymity network to determine the identity of its users.

4 Basic Ideas

The main strategy behind all current network anonymity systems is to forward messages through a series of machines (commonly referred to as a *circuit*) in an attempt to obfuscate the true source, the true destination, or both. The key is to endow all nodes in the circuit—including the source and destination—with an acceptable level of plausible deniability.

Note that none of these systems deal with cleansing the communicated data itself of identifying information. Therefore, if an HTTP request sent through an anonymity network includes a cookie that identifies the user, that user loses anonymity to the webserver (though the webserver will not know the user’s IP address). Tor [11] specifically recommends the use of Privoxy to scrub outgoing HTTP requests of identifying information such as cookies.

The many similar features will be illuminated as individual systems are presented, though there are a number of differences worth mentioning that highlight various research directions.

4.1 Connection-less vs Connection-based

Chaum’s original proposal [9] attempts to anonymize the sending of e-mail and, while he does describe a method by which responses can also be sent anonymously, SMTP is effectively a one-and-done, one-way protocol. This is in contrast to protocols such as HTTP and SSH which are request/response based, interactively so in
the latter case, and requiring a sustained connection. Beginning with Onion Routing [29], systems support the notion of a bi-directional connection created by an *initiator*; the term “sender” is now less relevant because once the connection is formed, either end may both send and receive.

### 4.2 Cloud vs peer-to-peer

Most systems consist of a set of machines to which users connect over unsecured networks. This “cloud”, as we call it, (see Figure 1) then bounces their communication around internally, before allowing it to exit from an egress point to its ultimate destination. Freedom [8] and Tor [11] are clouds, with the benefit that administrators are able to vet nodes before they are allowed into the anonymity cloud.\(^1\)

Scalability may become an issue as the service becomes popular but not enough users consent to being part of the cloud.

Alternatively, other systems (such as Crowds [24], Tarzan [13], and MorphMix [25]) enforce a peer-to-peer architecture, in which all users are participants in the anonymization network.

These two methods pose an interesting trade-off: in a cloud-based system, the anonymity set is huge because connections can originate from anywhere on the Internet, whereas the size of the anonymity set in a peer-to-peer system is strictly the set of peers. On the other hand, a peer-to-peer system has potentially far fewer points of failure, making it theoretically more resistant to, e.g., legal attacks.

Furthermore, peer-to-peer systems impose a greater burden on users, in that they must devote some of their resources (network bandwidth, processing power, and potentially storage) to other users of the system. In contrast, users of clouds need only connect to the cloud.

By extension, users of peer-to-peer anonymization networks could become implicated if some of their peers are using the system for illegal purposes. This could be either good or bad depending on the user’s perspective: distributing potential guilt strengthens the position of wrong-doers, whereas it increases the potential guilt of innocent users. While nodes in cloud systems are not necessarily directly guilty, they may become targets given their smaller numbers. Such potential exposure could, however, dampen enthusiasm of potential node owners to join the cloud, thus harming the scalability of the system.

### 4.3 Free-route vs cascade

Nearly all systems introduce the idea of a *circuit* to handle a single anonymized connection. This circuit consists of an ordered set of nodes (potentially with duplication, and possibly including itself) through which traffic is sent.

In a *free-route system*, such as Crowds [24] or Onion Routing [29], a circuit can consist of any nodes in the system, in any order.

In a *cascade system*, each node creates a set of connections to other nodes when it joins the network, and can only forward traffic over those connections. Examples of cascading systems include MorphMix [25] and Tarzan [13]. Limit-

---

\(^1\)The Tor paper claims this as a requirement (section 6.3 of [11]: “new nodes must be approved by the directory administrator before they are included”), though in practice it does not seem to be enforced: the instructions for setting up a Tor server do not mention obtaining permission from an administrator [2].
ing a nodes’ neighbor set reduces the potential anonymity of the system, though it is unclear by how much. Nikita Borisov has done some work attempting to quantify anonymity [7].

4.4 Who determines the route?

The initiator of an anonymous connection may itself define all nodes in the circuit at creation time (as in Onion [29]), they may be picked by subsequent nodes in the path (GAP [5] and Crowds [24]), or it may be a combination of the two (MorphMix [25]). Allowing other nodes to affect the circuit path requires some method to ensure the trustworthiness of those nodes, or to mitigate the effect of those nodes acting maliciously. In this case, the system is potentially quite vulnerable to colluding malicious nodes.

Salsa [21] allows nodes to select their own circuit paths, but coerces the node network into a specific structure in an attempt to improve selection of non-colluding nodes.

4.5 Trusted authorities

Freedom [8] and ANON [17] both require the presence of trusted authorities to provide anonymity. The ANON design dictates that because some parts of the network see raw IP addresses, “it is important that forwarders are properly protected from being compromised”. While they do not directly provide anonymity, Tor’s [11] directory servers are a trusted authority in that they store and disseminate the view of the anonymization network.

On the other hand, Chaum’s mixes [9] depend upon only one non-malicious node in the circuit; and MorphMix [25] introduces the idea of a witness in the circuit-creation process to reduce the chances of colluding nodes.

Consolidating trust in a few well-known entities heightens their attractiveness to attackers, as well as the potential harm that could be wrought should they become compromised. Freely distributing trust, however, potentially gives adversaries an easier vector for attack as they can insinuate themselves into the trust framework itself. These malicious internal nodes can analyze all traffic passing through them, and decrypt all traffic for which they have keys.

4.6 Complete network knowledge

Some systems require every client node to have complete knowledge of the network (while very different in structure, Tor [11] and Tarzan [13] are similar in this respect). This presents scalability issues in terms of memory, if not bandwidth and a bottleneck for potential directories. On the other hand, a system where every node is not aware of every other node could potentially be fragmented (such as GAP [5]), either maliciously or accidentally.

The intersection attack (described in Section 1 of [6]) takes advantage of an easily-accessible list of nodes: if an attacker is able to determine when a particular anonymous node is participating in the system, and gathers snapshots of the entire network at those times, the anonymous node in question must exist within the intersection of those snapshots. This is an example of an attacker reducing the anonymity set.

Furthermore, knowledge of only part of a network presents difficulties in evenly spreading anonymity across all nodes. Some systems (e.g., Tarzan [13] and Salsa [21]) impose a structure on the network to better select routes.

4.7 Latency

Chaum’s original proposal [9] was intended to anonymize e-mail, and was not only accepting of delay, but utilized it specifically to enhance anonymity. More recent systems (e.g., Tarzan [13] and Tor [11]) attempt to provide anonymity for low-latency application such as web browsing, IRC, and remote shell access.

While reducing latency is generally desirable from a user perspective, it can make traffic analysis easier [18].
5 Systems

In the following descriptions, $P_n$ and $S_n$ denote the public and private keys of $n$, respectively; $P_n\{m\}$ is the encryption of a message $m$ with the key $P_n$; $DH_n$ is node $n$’s half of the Diffie-Hellman handshake; a comma denotes concatenation.

5.1 Mixes (1981)

In 1981, David Chaum proposed the first system [9] to provide anonymity for sending and receiving e-mail; all subsequent systems carry vestiges of his original design.

Anonymity is provided by a set of machines called “mixes”, which may be both used as a cloud or a peer-to-peer system in which some of the mixes themselves generate traffic. Each entity in the system—mixes and nodes wishing to send e-mail—creates a public/private key-pair for message encryption.

In order for a node $a$ to send a message $m$ to a node $b$, $a$ first selects an ordered set of mix nodes of length $n$—the circuit $M$—from a well-known list (e.g., provided by a webserver or a usenet post). $a$ appends the address of $b$ to $m$ and encrypts it using $b$’s public key, appends the address of the last mix $M_n$ in the circuit, and encrypts the whole with $M_n$’s public key. $a$ recursively adds addresses and encryption until finally encrypting with the public key of the first mix. $a$ sends this message to mix $M_1$, which strips the first layer of encryption, revealing the next hop and a further encrypted message. The message thus traverses the circuit, a layer of encryption stripped at each step, until $b$ is left with the plaintext $m$.

In the simple case of one mix in the circuit, $a$’s message to $M_0$ looks like this:

$$P_{M_0}\{P_b\{m\}, IP_b\}$$

For three mixes, $a$ instead sends this to $M_0$:

$$P_{M_0}\{P_{M_1}\{P_{M_2}\{P_b\{m\}, IP_b\}, IP_{M_2}\}, IP_{M_1}\}$$

Note that a mix only knows its immediate predecessor and successor on a particular message’s circuit. It cannot tell whether it is the first or last mix in the circuit because the encrypted message it is forwarding may contain more hops or it may not. However, if the immediate source or destination of a message is not present in the global list of mixes (i.e., a cloud rather than a peer-to-peer network), a mix knows that it is the first or last mix in the chain, respectively, and it knows either the sender or the receiver.

Chaum also presents a method for $a$ to allow $b$ to respond without revealing the address of $a$.

If $a$ wishes $b$ to respond, but does not wish to reveal $IP_a$, $a$ can include $P_{M_0}\{IP_a\}$ in the message. $b$ uses this encrypted return address in a response to $a$, which traverses the same circuit in reverse. When the message reaches the final mix, $M_0$ can decrypt $P_{M_0}\{IP_a\}$ using its private key and forward the message to $a$.

To prevent a global passive adversary from correlating messages as they enter and exit a mix via traffic analysis, mixes buffer messages and output them in lexicographically ordered batches. Batching messages also prevents an adversary from tracking messages of a particular size through the network because a particular message will be included in different batches on entry and exit from any one node. Similarly, because the message is individually encrypted for each mix in the circuit, a message loses its anonymity only if the entire circuit is malicious. Chaum recommends that all senders also act as mixes to protect the initial ingress to the mix network.

While batching is a reasonable technique for mitigating traffic analysis of delay-tolerant protocols such as e-mail, it is less viable when attempting to anonymize low-latency protocols such as SSH. This is because the delay induced by a routing node to gather enough other traffic to generate a batch may degrade performance beyond usable bounds.

5.2 Onion Routing (1997)

In 1997, Syverson et al. formalized and enhanced the mix idea with Onion Routing [29]. Onion routers form a cloud, just as in mixes, and strip or add encryption as a message tra-
verses its circuit (the name references an onion and its many layers of skin). All nodes (client and routers alike) have public/private keypairs, and some mechanism is required to disseminate these keys.

The onion routing network topology is pre-defined (section 5.6 of [29]); when a new onion router joins the network, it establishes a socket connection to each of a subset of other onion routers—its neighbors—authenticated using the keypair of each node. A circuit can only be created through neighbor links, thus Onion Routing is a cascade system.

A client wishing to send anonymous traffic runs an onion proxy, which provides application-specific interfaces to the onion routing network. The client invokes an application, which requests an anonymous connection of the onion proxy. The onion proxy, knowing the network topology, selects a circuit and constructs an onion describing each hop, such that the first onion router will strip off the first layer of encryption, revealing the address of the second onion router, and so on. The final node in the circuit establishes a connection to the destination, and success is returned to the onion proxy.

Data is then sent from the application, through the onion proxy (which encapsulates it within an onion), and down the circuit (stripping off layers as it goes), and finally to the destination. Replies are the reverse: the destination sends data to the final onion router in the circuit, and the onion proceeds back up the circuit (adding layers of encryption as it goes), until it arrives at the onion proxy, which can remove all the encryption and pass the data to the application.

Similar to mixes, each onion router buffers some onions and re-orders them on output with the intention of foiling traffic analysis, though the buffering is less significant than in mixes because onion routing aims to be a relatively low latency system. To prevent an adversary from tracking a message through the network based on its size, all messages are padded to an identical length, and random bytes are added to a message whenever data is stripped (e.g., when an onion router removes the address of the next hop).

The main contribution of Onion Routing is the newly-placed emphasis on broad application support and low-latency, real-time connections. The authors have created onion proxies that support anonymization of RLOGIN, HTTP, and SMTP (section 5.3). This highlights the transition away from message-based anonymization as in Chaum’s mixes, and towards generic stream-level encapsulation. Additionally, Onion Routing is far more explicit about including multiple hops in a circuit to counter malicious onion routers.

5.3 Crowds (1998)

Reiter and Rubin’s Crowds paper [24] is important if for no other reason than it provides an excellent foundation for thinking about anonymous networking, what it means to be anonymous, and metrics for measuring anonymity. They propose a continuum describing anonymity: absolute privacy, beyond suspicion, probable innocence, possible innocence, exposed, and provably exposed.

The goal of Crowds is to provide senders with “probable innocence” anonymity against colluding members of the crowd. A stated non-goal is sender and receiver unlinkability against a global eavesdropper: they attempt to mitigate the possibility of such an adversary by encouraging crowds to span multiple administrative domains. Additionally, Crowds is also the first system that requires users to become part of the cloud—and hence is better classified as a peer-to-peer system—whereas mixes and Onion Routing merely recommended it.

Each user runs a piece of software called a jondo; upon starting, the jondo joins the crowd by authenticating to a “blender”, the response to which is a list of all other nodes in the crowd. The jondo acts as an HTTP proxy (thus limiting Crowds use to anonymizing web traffic) to which the user points her browser. Circuit creation proceeds by the local jondo contacting another jondo at random; the second jondo then flips a biased coin and either forwards the connection to another random node or becomes
the egress point. The “probability of forwarding” \( p_f \) is used as a knob to tune the anonymity provided: a higher \( p_f \) results in better potential resistance to colluding nodes at the cost of more potential latency. This is elaborated upon as part of the discussion of the Predecessor Attack in Section 7.2.

A jondo does not create a circuit immediately upon first use. At set intervals, the blender signals all nodes in the mix to terminate and reform circuits, thus allowing new nodes to create circuits without being easily traceable. Each circuit is protected by a single symmetric key, distributed by the user’s jondo at circuit creation time. This presents the drawback that any jondo in the circuit can see the contents of the message, though they cannot prove both who sent it and where it is destined.

Crowds enjoys a theoretical scalability improvement over the previous, cloud-based, systems. Because circuit length is dependent upon the probability of forwarding, and the number of circuits is the same as the number of jondos in the crowd, the total number of hops among all circuits should scale linearly with the size of the crowd.

5.4 Freedom (2000)

Freedom [8], run by Zero Knowledge Inc., is the sole commercial system covered in this survey. It is an IP-level service in which users purchase anonymity on a monthly basis; a user paying more per month is allowed to anonymize more anonymous applications. Visitors to http://www.freedom.net (as of 18 December, 2006) are informed that the service is no longer available.

The network consists of Freedom Server Nodes, used in anonymizing circuits, and Freedom Core Servers, which provide network directories, pseudonym creation, and gateways to external services such as Usenet and e-mail. The Freedom Server Nodes (also called “Anonymous Internet Proxies” or AIPs) behave essentially as onion routers: a user contacts the Core Servers to get a list of AIPs and public keys, and a user constructs a free-route circuit through the network.

The Core Servers also allow users paying for the appropriate service tier to anonymize traffic such as Usenet and e-mail. After tunneling through the network of AIPs to the Core Servers, the latter then makes a connection to an NNTP server or SMTP server and further tunnels the user’s data. This is an extreme example of the cloud network: there is a single egress point for Usenet and e-mail, which makes it a prime target for someone wishing to shut down anonymous access thereto.

Even barring the single point of failure, the commercial nature of this system makes it a far riper target—not to mention one much simpler to litigate—than the open networks presented in the rest of this paper. Additionally, given the success of Tor [11] and the demise of Freedom, it seems that people are happier to pay for anonymity by substantively helping it themselves, rather than with a monthly fee.

5.5 MorphMix (2002)

MorphMix [25] further develops the peer-to-peer strategy for anonymity; the authors claim that traditional networks lack scalability and are susceptible to traffic analysis because of the small number of nodes. A peer-to-peer design is not without its drawbacks, however, especially as the authors wish the system to be accessible to as many users as possible. They identify several points in this vein: lack of a global public key infrastructure (PKI) makes key distribution difficult; the ephemeral nature of many users’ presence online disrupts long-standing circuits; “normal” internet users do not have large amounts of bandwidth to spare. Beyond addressing these shortcomings, they aim for a system providing low latency and resistance to traffic analysis.

A node \( a \) connects to the network by learning about some nodes from an external source (e.g., a list from a publicly-accessible webserver, a usenet post, a trustworthy friend, etc.) and establishing a connection with each, encrypted using a separate symmetric key. These nodes are the neighbors of \( a \), and the connections
thereto form links through which a forwards data; thus MorphMix is a cascade system, as opposed to free-route. Beyond its neighbors, a may have knowledge of other nodes in the network, though it cannot forward to them directly.

Node a also generates a public/private key-pair. During circuit creation, a is made aware of the public key of each node in the circuit. To prepare a message, a recursively encrypts the payload with the public key of each node in the circuit, and encrypts the header (specifying the next hop) with the symmetric key of the link through which the circuit runs. As the message traverses the circuit, each node strips a layer of encryption on the payload, and replaces the header with the address of the next link encrypted with the symmetric key shared by that neighbor. Because a does not know the symmetric keys for every link in the circuit, it cannot arbitrate the complete circuit as in systems such as Onion Routing [29]. Additionally, a is not required to have knowledge of all other nodes in the network; therefore, how can a create a circuit and minimize the possibility of malicious nodes hijacking its connection? To solve this problem, the authors of MorphMix introduce a third party, the witness, to aid in selecting nodes for circuits.

Establishing the first hop in a circuit is straightforward, as a can just negotiate a key-pair with one of its neighbors. For a to extend its circuit beyond node b (which may be any node in the circuit), it first selects a witness w from the nodes it knows about, and encodes its half of the Diffie-Hellman key-exchange with w’s public key, along with a nonce:

\[ P_w\{nonce, DH_a\} \]

It sends this, along with w’s IP address to b. b selects from among its neighbors a set of potential next hops with capacity for another anonymous connection and presents this list to w via a link-encrypted connection, along with a second nonce. w picks a node c from the list at random, establishes link-encryption with c, and sends to c a re-encrypted form of a’s Diffie-Hellman handshake, along with the new nonce from b, and b’s IP address:

\[ P_c\{nonce_2, DH_a\}, IP_b \]

c is now able to send its half of the key-exchange, \(DH_c\) back to a through b. Note that b never saw \(DH_a\) in the clear, and hence can’t forge \(DH_c\). The use of the witness mitigates the possibility of b extending the circuit through another malicious node, using a sort of cut-and-choose method in which an untrusted source presents a set of choices and a neutral party chooses from them. This method depends, among other things, upon b providing at least some honest nodes in its selection set to w, which may be a precarious assumption.

If w and b collude, a circuit from a can be extended through other colluding nodes and a’s anonymity is destroyed. To detect this, a keeps a history of which nodes offered which next hops, and attempts to divine whether collusion is happening based on the distribution of node popularity in those lists. The authors claim that having a few very popular nodes and a great many less popular nodes reflects benign circumstances: mostly modest users with a few powerful, bandwidth-rich machines. Conversely, in a situation where colluding nodes are restricting a’s choice of hops, there will not be a broad selection of nodes, and the nodes will be repeated, thus producing a much different distribution than in the supposedly benign case.

The authors attempt to justify this assumption by running experiments and measuring the accuracy with which colluding nodes are detected, given an increasing percentage of them in the network, and assuming they offer to witnesses various numbers of honest nodes among malicious nodes. Their results indicate that, for their worst case scenario of 3,000 malicious nodes in a 10,000-node network, collusion goes undetected about 10% of the time with circuits of 4 hops or less. This seems rather high, especially given the proliferation of botnets, which provide a potential attacker with tens of thousands of malicious nodes.
5.6 Tarzan (2002)

Instead of using witnesses to enforce trustworthy paths, Tarzan [13] attempts to select neighbors with significantly different IP addresses, with the intent of spreading trust across different administrative domains. Tarzan is also the first system surveyed to operate at the IP-layer, thus requiring minimal alterations at the application level.

Each node maintains a list of all other nodes it knows about, dubbed its neighbors; a node initializes its neighbor set through external means, as in MorphMix. Initially all of these nodes are unvalidated; the process of validation involves making a direct connection to that node and requesting its public key and neighbor set. Not only does this ensure that the particular node under investigation is responsive, it gives a more potential nodes through which to route. Validation of the entire network requires $O(n)$ connections; this is a daunting task in the face of a large network, especially considering that, as a peer-to-peer network, Tarzan depends upon a large number of participating nodes to effectively provide anonymity.

Each node also maintains a set of mimics, a validated subset of neighbor nodes, to which connections are maintained; these connections form the paths for both anonymous communication and cover traffic, the latter of which is intended to foil traffic analysis. To minimize the chance of selecting colluding nodes as mimics, a attempts to choose nodes from a broad cross-section of administrative domains. To do so, a first chooses a /16 subnet at random from among its neighbors, and then a /24 subnet from within that selection, and finally an individual node from within that. This process is repeated until a has a reasonable number of mimics, ideally choosing nodes that are far apart in the global IP space. The assumption is that it is highly unlikely that nodes across such a broad swath of the IP space are colluding.

To construct a circuit, a selects one of its mimics, $b$, as the first hop; $b$ sends to a its list of mimics, from which a chooses the subsequent hop according to the algorithm described previously. This process is repeated until the penultimate node is added to the circuit, at which point a selects a node from its verified list to act as egress point. The reasoning behind not requiring the egress point to be in the mimic chain is because, should an intervening node become inaccessible, the circuit can be reconstructed and use the same egress point, thus maintaining the connection with the external machine.

Each mimic-to-mimic link carries a constant amount of traffic: valid, anonymized data padded with bogus bytes to foil traffic analysis. This does, however, present a potential attack: a malicious mimic could slow the rate at which it sends data to a target node and potentially learn something about which outgoing traffic is then originated by that node (as opposed to forwarded). To defend against this, a node's outgoing data rate to any particular mimic is kept relative to its total incoming data rate. This allows a node to maintain outgoing data in the face of some of its mimics attempting to starve it of sending capability, maliciously or otherwise.

5.7 Tor (2003)

Tor [11] builds upon the original Onion Routing system [29] with enhancements specifically intended to improve its practicality for a real-world deployment. The designers of Tor deemed Onion Routing to be a “fragile proof-of-concept” (section 1), though the short-lived prototype was able to handle fifty thousand connections per day.

The Tor network is very similar to that of Onion Routing, in that there is a cloud of onion routers to which a user connects by way of an onion proxy. One significant difference is that all onion routers maintain TLS connections to every other onion router, rather than a subset as in the original system. Tor explicitly avoids application-level anonymization (e.g., re-
moving cookies from HTTP requests), preferring to leave that responsibility to an external application\(^4\). Additionally, Tor uses the SOCKS interface to ease integration with existing applications.

A user wishing to create a circuit does so hop-by-hop as in MorphMix, though the onion proxy is free to choose any node because it has complete knowledge of the onion router network. The circuit extension message is recursively encrypted such that nodes coming before the last node of the circuit do not know whether the message passing through is data or control. To amortize the cost of circuit creation, a single circuit can be used to transport multiple TCP streams. This circuit reuse might allow for easier traffic analysis; therefore, an onion proxy may cause a particular TCP stream to use an internal node in the circuit as the egress point, rather than the final node.

One of Tor’s less pedestrian contributions is that of rendezvous points and hidden servers, the idea of which is to allow both sender and receiver to stay anonymous to each other, as well as to potential adversaries. The server to be hidden chooses a set of onion routers to act as introduction points and creates an anonymized circuit to each; the list of introduction points for a specific hidden service is made publically available. A user wishing to access the service chooses an onion router to act as rendezvous point, connects to one of the service’s introduction points, and sends it a request to access the hidden service along with the chosen rendezvous point. The hidden server then creates a circuit to the rendezvous point, and the rendezvous point connects the user circuit and the service circuit, thus creating a link in which neither end knows the identity of the other.

With the exception of rendezvous points and hidden servers—which we will revisit in the conclusion—Tor incrementally improves upon the original Onion Routing system. These improvements are not to be underestimated, however, as they appear to be well on their way to their goal of practical deployment: the Tor network boasts 1202 nodes \([1]\) and “several hundred thousand active users” \([3]\) as of 03 January, 2007.

5.8 Salsa (2006)

While not an anonymous routing system per se, Salsa \([21]\) provides a framework for organizing nodes and selecting circuits in both cloud and peer-to-peer systems. The goal is to reduce the probability of selecting multiple malicious nodes in a circuit by choosing nodes using a random and unbiased method. In this sense, Salsa is an alternative to MorphMix’s witnesses.

All nodes in Salsa are identified by applying a consistent hash \([15]\) to their IP address; each node remembers a subset of other nodes in the system, dubbed its contacts. The ID space is divided into equal-sized groups; consistent hashing should ensure a relatively uniform distribution between groups. A node exists within a group if its ID falls within the group’s boundaries; a node “owns” the portion of its ID space between it and the previous ID; every other node in the group is part of that node’s local contacts.

To choose a routing hop, a node selects an ID at random and performs a lookup to find the node controlling the ID space within which the selected ID falls. In order to facilitate this lookup, groups are organized such that the set of groups forms the set of leaf nodes of a binary tree. Therefore, the tree may be traversed to find a certain ID space. However, to minimize the probability of malicious nodes providing an incorrect result for the lookup, each node keeps a set of global contacts in different groups. The groups from which the global contacts are drawn is determined by the tree structure: in each level of the tree, one global contact is drawn from both children of every node. Furthermore, the individual node in each group is selected by hashing the selecting node’s IP address with the height of the tree.

Salsa combines circuit creation with multiple redundant lookups to minimize the chance that a malicious node or nodes will be able to de-
duce the final node in the circuit and hence link
the sender and the receiver. A node wishing to
create a circuit does \( r \) independent lookups and
uses each result as the first hop of a separate cir-
cuit. Then it sends another \( r \) lookups through
the first set of nodes and extends each original
circuit to a random node in the second set. This
process is repeated until the initiating node has
\( r \) circuits of the desired length, the key being
that no nodes in a particular circuit were used
to look up any other nodes in the same circuit.

The benefit of Salsa is that nodes are able
to select circuits with limited knowledge of the
network. Consistent hashing ensures that malic-
ious nodes may not place themselves in specific
areas of the ID space to lure in victims. Addi-
tionally, redundant lookups are used to mini-
mize the chance that attackers can link a cir-
cuit’s final node to its initiator.

6 Applications

An extension of anonymous routing is anony-
mous publishing: the ability to make informa-
tion available without anyone knowing the
source, or where the information resides. This
is similar to Tor’s hidden services, though there
has been much parallel research in this area
worth mentioning.

6.1 The Eternity Service (1996)

Ross Anderson’s 1996 paper [4] lays out a laun-
dry list of requirements for such a system, and
forms the basis for research in the area. While
he doesn’t present an implementation, let alone
a design, his work is important in that it pro-
vides an excellent start on enumerating threats
and target features. His goal, in a nutshell, is
a filestore that is resilient to errors, accidents,
and denial of service attacks—where the latter
includes direct attacks on the system as well as
indirect attacks on publishers.

The author sketches a system in which a user
pays a fee to store a file of a certain size for
a certain length of time. The file will be dis-
tributed across a redundant array of servers
around the world, more servers providing more
assurance that a single adversary cannot hunt
down and destroy all copies. Anonymous re-
mailers (i.e., the systems discussed in section 5)
are used to transport files to the service, thus
preventing an adversary from tracking uploads.
The cryptography used will need to be easily
upgradeable: given the projected longevity of
the system, it is imprudent to assume that cur-
rent cryptographic methods will be useful for
the duration. The system will need to provide
an index of stored files, orchestrated in such a
way that users can find what they want but an
adversary cannot deny service by preventing ac-
cess to the index. The various servers involved
in storing the files will need a secure method
of synchronizing their clocks to correctly expire
files.

One point that goes unmentioned is quality
control on the data stored in such a system. It
is a wonderful idea to be able to make informa-
tion available forever and to everyone, but who
makes sure that information is correct? If verifi-
ably false information is disseminated, how can
its falsity be made known? The next system
addresses this problem.

6.2 Publius (2000)

In addition to replication, Publius [31] supports
mechanisms to verify the integrity of published
files, to delete and update files, and to publish
files which contain hyperlinks to other Publius-
published files.

A user wishing to publish a file encrypts it us-
ing a symmetric key. That key is then divided
into \( n \) parts using Shamir’s secret-sharing algo-
rithm [27], a property of which is that only \( k \)
parts (where \( k < n \)) are required to reconstitute
the key, and hence decrypt the file. Each key
share is then distributed to a different server,
along with a copy of the encrypted file. A malic-
ious server can corrupt either its replica of
the file or its key share, both of which will be
discovered when file decryption fails.

Published files may be deleted or updated by
also distributing a hashed password to servers
containing the key shares, and then providing
the password when issuing a delete or update
request. Delete is straightforward: upon receiving the correct credentials (i.e., the password), a server removes all files related to the published file. Updates are more complicated: instead of changing the file in place, a file called update is associated with the published file on each server, containing the Publius URL of the new content. Thus when a user wishes to retrieve the file, their request is redirected to the update URL, much like a UNIX symbolic link.

The authors point out that some files to be published (e.g., HTML files) may contain references (e.g., links) to other Publius-published content; furthermore, such links may create circular dependencies because Publius URLs are dependent upon a hash of the message. This is alleviated by using the update procedure described above: publish both files, then update either or both with mutual links corrected.

One of Publius’ strengths—the ability to divine the specific servers hosting a file from its URL—is also a weakness. Given a URL, an adversary knows exactly which machines to attack. This calls into question exactly how censorship-resistant Publius really is: it would almost certainly be more so if an adversary could not trivially figure out the locations storing a particular document. Later systems address this shortcoming.

An additional side-effect of the method chosen to encode the server list into the URL is that the machine mapping depends upon a static list of servers, effectively preventing expansion of the network. Admittedly, machines may be replaced with more capable hardware, but machines may not be added. It is entirely conceivable that a Publius deployment could outgrow such constraints.

6.3 Freenet (2000)

Freenet [10] is a peer-to-peer anonymous publishing system in which the physical locations of the files themselves are anonymized, thus impeding the ability of an adversary to directly affect them. This is achieved by anonymously routing the publication and retrieval of files, and by randomly replicating files as part of the retrieval process itself. A further side-effect is that, because nodes evict files based on an LRU scheme, files may expire from the system and become forever unavailable.

Three different kinds of keys are used to identify files within Freenet. A keyword-signed key identifies an individual file by taking a hash of a descriptive string. A signed-subspace key establishes a namespace under which the owner of the key can publish documents, which are individually identified by keyword-signed keys. Lastly, a content-hash key is used in conjunction with a signed-subspace key to facilitate an update-by-indirection mechanism similar to that used by Publius.

To retrieve a file, a user must possess the file’s key (i.e., the descriptive string, which is then hashed, or the descriptive string and subspace key). She presents it to a member of the Freenet network—presumably an instance of the software running on her own machine—which checks to see if the request can be satisfied locally. If so, the file is returned directly; if not, the request is passed along to the known node most likely to have it, where “most likely” is determined by lexicographic distance between keys. The request is forwarded until either the file is found or the hops-to-live, assigned when the request is made and decremented at every hop, expires. If the file is found, it is transferred back along the path that was used to find it, and nodes along the way may cache the file as it passes through them.

This system has two interesting properties: firstly, it guarantees neither file retention nor that a file will be found if it exists in the network and an appropriate key is requested. While the authors’ experiments indicate that median performance scales fairly well (request path length scales logarithmically with network size), they do not address worst-case performance: namely, how likely is it for a file to be purged from the network, or for a search to fail when the file actually exists in the network? These metrics may be important factors in the feasibility of such a system; users may become justifiably unhappy if files disappear, or seem to if searches fail, even though they haven’t ac-
cessed them lately. At the very least, users need to be educated about these possibilities.

Secondly, the location of files within the network is effectively random because it may have been replicated by one node’s cache as a result of user activity and evicted from another’s out of disuse. As a result of the unknown and fluid nature of file locations, Freenet is a much more viable censorship-resistant system than Publius. Also in contrast to Publius, individual nodes may freely join and leave Freenet with minimal effect on the network (section 5.3 of [10]), potentially improving its attractiveness to casual users.

One of Freenet’s downsides is that performance can suffer because content traverses many nodes in a single request. For small documents, this is probably acceptable, but as file sizes increase, this becomes less tenable. Additionally, the paper actively avoids the topic of a file index to enable the finding of files; this is an important point, as the act of searching a publically-available (and not anonymously-accessible) index might be incriminating.

6.4 Tangler (2001)

Tangler [30] takes a different approach to spreading files across a network. Rather than duplicating the file in its entirety and distributing parts of a key, Tangler divides each file into a set of blocks, a subset of which can be used to reconstruct the file, and combines them—through a process called entanglement—with blocks from other files before distributing them about the network. This has the interesting side-effect of propagating existing files whenever any new file enters the system. This ties files together with unrelated files, thus encouraging the mutual prosperity of the system by unrelated users. It is not without its drawbacks, however: malicious servers may abuse the credit system and cause denial of service to legitimate users attempting to either publish files to that server or retrieve blocks from that server. Additionally, new servers are only used for replication until they have proven themselves reasonably trustworthy (a period of one month is suggested).

The Tangler system is particularly intriguing because it ties files together with unrelated files, thus encouraging the mutual prosperity of the system by unrelated users. It is not without its drawbacks, however: malicious servers may abuse the credit system and cause denial of service to legitimate users attempting to either publish files to that server or retrieve blocks from that server. Also, the requirement that files must be manually refreshed could be problematic. Additionally, the fact that an adversary can directly divine the location of blocks for a specific file presents an attack vector for invading or shutting down individual servers.
and hence censoring content or denying service.

7 Attacks

Anonymity systems present new avenues of attack. This section summarizes some of the more high-profile papers describing attacks against such networks.

7.1 Timing Attacks

In the simplest case, a global passive adversary could attempt to correlate traffic entering and exiting a cloud-based anonymity network to link a sender and a receiver. Whether or not this particular adversary is a practical concern is a matter of some debate; Tor assumes not (section 3.1 of [11]) and systems such as Tarzan [13] enforce a circuit-creation policy that purposely attempts to spread a circuit among administrative domains, lessening the possibility of such an adversary.

Shortly after Tor was deployed, Murdoch and Danezis [20] outlined how traffic analysis could be used to degrade the anonymity of Tor’s users. Traditional traffic analysis would require being able to sniff all traffic in and out of the Tor cloud. However, the authors point out that the very nature of Tor as a low-latency network makes it susceptible to a slight variation. Specifically, the overhead of a single anonymous connection running through a Tor node increases the latency of all other anonymous connections running through that Tor node.

A malicious web server, for instance, could infer a client’s entire circuit by obtaining a list of all Tor nodes and characterizing their performance by measuring the affects of circuits created through each node (recall that Tor is a free-route network in which the initiator defines the entire route). Then, when a client connects, the webserver can again measure latency through each Tor node and note which have increased, which would indicate an added circuit.

This seems to be a potential security hazard for all low-latency anonymizing networks, given their very nature, and is most certainly an interesting avenue for research. Peer-to-peer networks are a promising path, given that the initiator is part of the network, and may be obfuscated more effectively because of this. Unfortunately, peer-to-peer networks are especially vulnerable to the following attack.

7.2 Predecessor Attack

One of the most potentially damaging attacks on circuit-based networks, originally proposed in Reiter and Rubin’s Crowds paper [24] and further explored by Wright et al. [32], is the Predecessor Attack.

The authors assume that the initiator of an anonymous connection will maintain that connection across potential reformations of the intervening circuit. It is these reformations that are key to the attack. Malicious nodes will be chosen at random as part of the reformed circuit; the nature of IP is such that a node knows its predecessor in the circuit. Statistically, the initiator of the circuit will be the predecessor the plurality of the time; given enough circuit reformations, the attacker could divine with reasonable surety which node is the initiator.

Certain systems are less prone to this attack than others, though all eventually succumb. Crowds is particularly susceptible because data is sent in the clear and hence a single circuit is easy to track. Onion Routing is more secure because the layers of encryption stripped or added at each step help to obfuscate the circuit. Chaum’s mixes, which batch and re-order messages, provide a better defense, as well. Peer-to-peer systems in which a list of all participants is readily available (such as Tarzan [13] and Crowds [24]) are vulnerable because of the potentially high churn rate of peers. Conversely, peer-to-peer systems in which all participants are not easily known (e.g., MorphMix [25]) may be more resistant to the predecessor attack. In the case of Crowds specifically, the chance of a certain node being the initiator can be tuned by tweaking the probability of forwarding, $p_f$: a higher value makes it more likely that the preceding node is merely forwarding and is not the initiator.
Given the nature of this attack, research into peer-to-peer systems which do not require knowledge of all other nodes seems to be a fertile area.

### 7.3 Sybil Attack

John Douceur describes the Sybil Attack [12], in which a single malicious entity is able to masquerade with multiple identities in large-scale peer-to-peer systems. He shows that in systems lacking a central authority to verify identities, such an attack can allow malicious users to overwhelm honest users.

Some of the systems discussed in this survey (such as MorphMix [25]) are susceptible to the Sybil Attack. Most systems, however, enforce an indirect requirement that a single IP address may only be home to one logical node (Tarzan [13] and Salsa [21] behave this way). An interesting side-effect of this is that users behind firewalls performing network address translation (NAT) are effectively prevented from using the service. (Not to mention the fact that most firewalls do not allow incoming connections, which are necessary in the peer-to-peer systems described here, and if a user wishes to participate in the cloud.)

### 7.4 Hidden Server attack

Similar to the predecessor attack is a method used by Øverlier and Syverson [22] to expose hidden servers in the Tor network. Their attack is able to identify a machine running a hidden service within minutes, using only a single malicious node. The authors also propose a modification to the Tor network to mitigate this vulnerability.

A user wishing to locate a hidden server must first become a member of the Tor routing network. Once in place, this malicious node makes continual connections to the hidden service, each in an attempt to become the node closest to the hidden server, between it and the rendezvous point. The attacker can correlate traffic patterns as both circuit and attacker, and identify whether its routing node instance lies on the circuit between the hidden server and the rendezvous point. (Recall that in Tor’s hidden server scheme, the client constructs a circuit to the rendezvous point and thus all nodes therein.)

Once the attacker is able to identify whether they are on the correct circuit, their next goal is to become the first node in the circuit, taking advantage of the statistics presented in the Predecessor Attack. The IP address of the circuit initiator (i.e., the hidden server) is exposed to the first node in the circuit, and hence the hidden server is located.

To combat this attack, the authors present a system of guard nodes, in which the hidden server uses a number of different—but fixed—entry points to the anonymity network. While they are convinced, this does not seem to be a particularly robust solution, as it merely introduces another level of nodes in which the attacker must insinuate themselves. Hidden services remain an open research area, and as discussed in the next section, will be an important component of anonymity networks as they develop into broadly usable systems.

### 7.5 Clock Skew attack

Some recently-published attacks include various methods of identifying machines by tracking their clock skew. Kohno et al. demonstrated that, in a 69-machine sample set run for 38 days, “clock skew estimates for any given machine are approximately constant over time, but that different machines have detectably different clock skews” [16].

Stephen Murdoch describes an attack [19] using this technique in which a malicious external user generates load on a hidden server with the intention of raising its processor temperature, hence altering the operating characteristics of its clock in a manner that is measurable in the network packets the machine emits. It is a very interesting covert-channel-like approach, though it does require that the attacker sniff outgoing traffic on all potential hidden servers, which may be an unrealistic requirement.
8 Looking Ahead

As evidenced by the volume of publication in the area, anonymity systems are an active research topic and fertile ground for new ideas. From the systems surveyed, some promising trends and interesting questions emerge.

Firstly, Tor explicitly states the need to develop a reputation as a legitimate service before venturing into such litigously gray areas as file sharing. We believe this to be an absolutely vital aspect to any anonymity project: without sufficient legal uses, any network of reasonable size will be shut down in short order (though not anonymous, Napster and Kazaa are prime examples, with Gnutella not far behind). If anonymity systems are to be widely adopted, they must establish themselves as providing legal services.

Another way to encourage wider adoption is to expand the multi-purpose nature of anonymous networks. As more and more applications are supported by a single network, more anonymity is gained because an adversary cannot tell the application(s) in which a user is participating. More applications will also very likely attract more users, which increases the anonymity of everyone involved. Some systems—such as Tarzan [13]—operate at the IP level, an approach which has great potential to enable current applications to be easily anonymized.

None of the systems surveyed hide which users are participating, though all attempt to hide how they are participating; that is, none of the systems provide unobservability. Sender/receiver anonymity, therefore, might not be enough protection in some situations: for example, an oppressive government might consider mere participation an objectionable offense, regardless of any active communication on an anonymous network. This is a further argument for expanding application support: mere participation is less likely to be an offense if redeemable applications run on such a network. Nonesuch [14] is a very recent system that attempts to reduce observability, though there is certainly room for further research.

An interesting point to note is that as all nodes become equally unlikely to originate traffic, they also all become equally likely to do so. This presents challenges when considering possible uses of anonymous networks: how does a node owner prevent someone else from using her machine for illegal purposes? Could a node owner take the opposite route and instead enforce a policy to actively support certain services? For example, a user might want to act as a storage mirror for a preferred charity and hence only participate passively in the network.

Not only might such a user want to help a particular group, but that user might not want to be identifiable as such. To that end, we believe that anonymity for servers is an important research direction. Tor has the concept of hidden servers and rendezvous points, but as discussed in section 7, compelling attacks have been presented against them.

Another probable vector for attacks is presented by the Salsa paper: the growing number of botnets pose a significant challenge to anonymity networks. They provide a potential attacker with tens of thousands of machines for use as colluding nodes or as a platform for launching denial of service attacks against participating nodes. This daunting adversary poses an enormous threat to systems that depend on the difficulty of a single malicious user gaining control over a variety of machines in a variety of networks. Salsa [21] presents an interesting angle on solving this problem, but we believe there is more work to be done in the area.

9 Conclusion

This paper has provided a summary of research into anonymous routing systems, beginning with Chaum’s initial proposal of mixes in 1981 and progressing through systems that are currently operational in 2006. Certain concepts—such as circuits—are fairly consistent throughout, whereas others vary wildly—peer-to-peer systems and cloud systems, free routing vs cascades, initiators choosing their
own circuits vs having the next hop chosen at each individual node, and so on.

Anonymity on the internet is an admirable goal and, while much work has been done already, there yet remains a large and exciting problem space, ripe for research.

References


