GAP – practical anonymous networking*

Krista Bennett and Christian Grothoff

S^3 lab and CERIAS,
Department of Computer Sciences, Purdue University
kbennett@cerias.purdue.edu, grothoff@cs.purdue.edu
http://www.gnu.org/software/GNUnet/

Abstract. This paper describes how anonymity is achieved in GNUnet, a framework for anonymous distributed and secure networking.
The main focus of this work is GAP, a simple protocol for anonymous transfer of data which can achieve better anonymity guarantees than many traditional indirection schemes and is additionally more efficient. GAP is based on a new perspective on how to achieve anonymity. Based on this new perspective it is possible to relax the requirements stated in traditional indirection schemes, allowing individual nodes to balance anonymity with efficiency according to their specific needs.

1 Introduction

In this paper, we present the anonymity aspect of GNUnet, a framework for secure peer-to-peer networking. The GNUnet framework provides peer discovery, link encryption and message-batching. At present, GNUnet's primary application is anonymous file-sharing. The anonymous file-sharing application uses a content encoding scheme that breaks files into 1k blocks as described in [1]. The 1k blocks are transmitted using GNUnet's anonymity protocol, GAP. This paper describes GAP and how it attempts to achieve privacy and scalability in an environment with malicious peers and actively participating adversaries.

The GNUnet core API offers node discovery, authentication and encryption services. All communication between nodes in the network is confidential; no host outside the network can observe the actual contents of the data that flows through the network. Even the type of the data cannot be observed, as all packets are padded to have identical size. Availability is guarded by an accounting scheme that is based upon link authentication and which does not require end-to-end knowledge about transactions [10].

The goals of the GNUnet project are to explore the possibilities and limitations of secure peer-to-peer networking. Achieving privacy is a significant problem for peer-to-peer technology, even when no spyware is bundled with applications. Additional security features are needed before peer-to-peer networks can be trusted to store more sensitive data, such as medical records. GNUnet is a strict peer-to-peer network in which there are no nodes exercising control over the network.

* Portions of this work were supported by sponsors of CERIAS

Any kind of central server would open the network to attacks, whether by attackers trying to control these entities, legal challenges, or other threats which might force operators of such critical and exposed nodes out of business. The best way to guard against such attacks is not to have any centralized services.

GAP strives to achieve initiator and responder anonymity in relation to all other entities, including GNUnet routers, active and passive adversaries, and the responder or initiator respectively. The actions involved in publishing content are indistinguishable from those involved in responding to requests; thus, responder anonymity covers publisher anonymity in GAP. It is not possible for peers to retrieve content from publishers that do not use GAP. Also, content migrates over the network. Because of this, even if responder anonymity is broken, there will be no certainty that the original publisher has been identified. While anonymity for intermediaries would be desirable, participation in a protocol on the Internet is generally visible to any powerful adversary. Thus, GAP does not strive to hide participation in the protocol. For GAP, it is only important that no adversary can correlate an action with the *initiating* participant.

The most significant difference between GAP and prior mix-based protocols is that traditional mix protocols always perform source rewriting at each hop. GAP mixes can specify a return-to address other than their own, thereby allowing the network to route replies more efficiently. GAP does not attempt to avoid a direct network connection between initiator and the responder. In order to achieve anonymity, it is only important to decouple the relationship between the initiator and the action. Thus, anonymity is achieved if an adversary cannot determine the initiator of an action. This can be achieved by making the initiator look like an intermediary: a participant that is merely routing data. This realization allows GAP to bypass a typical restriction on most indirection-based anonymous routing protocols which require that either the reply takes exactly the same path as the request [5, 13] or the path is statically predetermined and cannot be optimized en route [3, 12, 21]. Some protocols, like [19], use multicasts for the reply, but these consume large amounts of bandwidth.

In order to understand how GAP works it is important to realize that given a powerful global passive adversary, the operation of proxy services like the website `anonymizer.com`, which are generally perceived to be anonymizing requests from their customers, degenerates to a situation where the customers merely provide cover traffic for the proxy service. The only entity which can then proceed with reasonable anonymity by using the proxy service is the actual operator of the proxy service (if they use the service from within). Similar problems arise for many other centralized anonymity systems (even the ones that use distributed trust), since they do not provide cover traffic for senders and receivers. A global passive adversary can attack these schemes by treating the whole set of mixes as a black box and simply looking at the messages going in and coming out at the initial entry point and final exit point. This type of network-edge analysis is made impossible in a peer-to-peer network where receivers and senders are part of the mix. Tarzan [9] is an anonymizing peer-to-peer infrastructure where the initiators are part of the mix network. Tarzan cannot anonymize responders since they are not part of the mix network. Since censors typically prefer to attack the small set of publishers instead of the large group of readers, GAP's approach of anonymizing both senders and receivers has potentially broader applicability and greater usefulness in censorship-resistant networking.

Anonymity and the Adversarial Model

A communication is defined to be anonymous with a probability p if the adversary cannot prove with probability greater than p that a node was the initiator or the responder in that communication. For the discussion in this paper, the adversary is assumed to have no means other than the interactions of the nodes via the protocol to determine if a node was the initiator. The paper will show that given certain assertions about the adversary, a node using GAP can determine its degree of anonymity and trade anonymity for efficiency. Note that the question of whether or not a system is *sufficiently* anonymous for a specific application depends entirely on that application's purpose and, thus, the choice of p may vary. This allows nodes to operate more efficiently whenever applicable.

It is important to note that the **burden of proof** is put on the adversary. The adversary must identify a node and prove that a communication originated from there. If the legal system were to require the *initiator* to disprove being the origin, anonymity becomes essentially illegal. Aside from the restriction of only using evidence obtained from protocol-related interactions to prove that a node performed a particular action, the adversarial model for GAP allows the adversary to do almost anything but break cryptographic primitives. The adversary is assumed to see all encrypted and unencrypted traffic between all nodes at all times, but cannot decrypt encrypted traffic between two nodes where neither node is controlled by the adversary. The adversary controls an arbitrary number of nodes in the network and the nodes are free to collaborate out-of-band. Adversarial nodes can violate the protocol and/or be well-behaved participants in the network. The adversary can also interrupt communications between arbitrary nodes in the network. Since GNUnet is a peer-to-peer network in which every peer has willingly joined and where no messages are sent to machines that are not part of the overlay network, problems with exit-nodes [6] which communicate with the final recipient (who often is not part of the network) do not arise.

While the adversary described above is extremely powerful, its power must be limited slightly. The reason for this is that if the adversary is able to control (or at least decrypt) all traffic that a node sends or receives, the node cannot engage in any anonymous communication. This is true for all anonymous networks, however. Thus, in order to provide the user with any degree $p > 0$ of anonymity, any protocol requires that a node must be able to communicate with at least one other node that is not controlled by a malicious adversary.

GAP cannot prove that the adversary is bound by this constraint in practice. In fact, even a powerful adversary that obeys this rule can break anonymity in GAP with a certain probability p. The degree of anonymity p provided by GAP depends on the strength of the adversary and the internal trade-offs that the individual node chooses to take. Note that the trade-off is not negotiated with other peers and does not impact their operation. The determination of the degree of anonymity that can be achieved is based on an estimate of the power of the adversary, which must essentially be guessed. If the guess is that the adversary controls all communications, the result would be that in order to achieve any degree $p > 0$ of anonymity, no communication may take place.

2 Anonymity in GNUnet

This section describes how anonymity is achieved in GNUnet using GAP. In order to be able to evaluate the anonymity guarantees that GAP provides, a new perspective on how indirecting communications results in anonymity is first described. Then the scheme employed in GAP is laid out and its guarantees and efficiency are discussed.

2.1 Hiding the Initiator of Activity

Consider the scenario illustrated in figure 1. In this scenario, a node receives two queries and sends three. In this picture, the two nodes that sent their queries

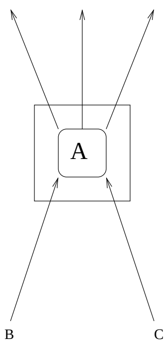

Fig. 1. Hiding

are exposed; node A can correlate these nodes with their queries, as a traffic analysis reveals that both B and C sent a query. Also an external adversary can tell that B and C started some communication. If A is allowed to send a query twice, traffic analysis alone cannot reveal if A sent a new query or was merely indirecting queries from other nodes.

In this sense, indirections do not hide the original senders B and C from powerful adversaries; instead, indirections obfuscate what the node that is indirecting is doing. No scheme that tries to achieve anonymity on an observable, open network can hide the fact that a node is *participating*. The best a scheme can do is guarantee that no adversary can distinguish activity that a node initiates from mere participation in the protocol. The example above demonstrates that a node can hide its own activities by handling traffic for other nodes.

2.2 Protocol Overview

The GAP protocol consists of two types of messages: queries and replies. A query consists of a resource identifier (in GNUnet, RIPE160 hash codes are used) and a node identifier that describes where the reply should be sent. This reply field is the primary difference of the wire-format compared to protocols such as Freenet or Crowds where the reply always goes to the sender of the query. A time-to-live field is also included for routing purposes. The time-to-live has a pseudo-random initial value and is decremented by routers with additional pseudo-random components in the expression. A reply is merely the data that was requested. Communication between nodes uses link encryption; each node is linked to as many nodes as possible.

The resource identifier of a query is passed to GAP by the application layer. If a reply is not received after a certain (randomized and exponentially increasing) amount of time, the query is retransmitted. Queries are searches; success is not guaranteed. For example, the resource may simply be unavailable at the time of the query. The application layer is responsible for deciding when to give up.

After a node receives a query, it processes the query by taking the following steps:

1. Determine if the node is too busy to process the query. This check includes CPU and bandwidth availablility and free space in the routing table. If the node is too busy, drop the query and exit.
2. Determine if the desired resource is available locally; if so, enqueue the reply into the sender queue of the receiver that was specified by the query.
3. Decide how many nodes n the query should be sent to (the query's time-to-live, information about the load, accounting information, and a random factor influence the decision); if $n > 0$, enqueue for sending to n other nodes by doing the following:
   (a) Decide whether to replace the identifier of the previous requester with the local identifier based on current anonymity goals and load for the local node.
   (b) If the node replaces the identifier of the previous requester with its own identifier, associate the previous requester with the query in the "routing table".
   (c) Choose n target nodes and enqueue the n queries.

Peers also always:

– Flush individual queues (containing a mix of queries and replies) after a random (but bounded) amount of time.

- When receiving a reply, look in routing table for matching query and the identity of the next receiver. Enqueue the reply, or hand the content to the application layer for requests from local clients. Copy content into local storage (migration) if content is considered valuable and space is available.
- Discard infrequently accessed content from local storage.
- Send random content out into the network to provide background noise when network is idle (and to boost content migration)

2.3 Anonymity Guarantees

This section presents an analysis of how much anonymity peers can achieve using GAP. For the analysis, it is assumed that peers achieve perfect mixing of messages (queries and replies) in a given time interval. Perfect mixing in this sense means that an adversary cannot use timing analysis to correlate messages that the peer receives with messages that the peer sends. The time-interval for which this is achieved depends on the delay strategy that the peer is using, but optimal delay strategies for mixes are out of the scope of this work.

In order to answer the question of how strong the anonymity guarantees are that indirection can provide, some additional constraints must be considered. The first observation is that the more traffic a node creates, the more foreign traffic it must route in order to obscure its own actions. Suppose a node A injects n queries into the system and routes m queries from other users. An adversary that does not participate in the network but monitors the encrypted traffic can see the amount m of data that the node received and the amount $n+m$ of data that A sent. Thus, this simple adversary would determine that any of the queries originated from A with a probability of $\frac{n}{n+m}$.

If the adversary uses timing analysis, it is possible for the adversary to exclude certain amounts of traffic that were routed a long time ago. The interpretation of "long" here depends on the potential delay that a query may typically face in a node. Nodes can delay queries for random amounts of time in order to make this timing analysis harder. Excessively long delays make the query useless, and indirecting the query then becomes equivalent to producing noise (with the exception that other nodes will not perceive it as such).

Now, suppose that the adversary is actually actively participating in the network. In this case, traffic that originated from this active adversary cannot be included in m. The adversary knows that this traffic did not originate from the node under scrutiny. At this point, it should be clear why the assumption was made that every node always interacts with at least one node which does not collaborate with the adversary who is trying to break anonymity. Otherwise, m could be zero and the probability of the node being identified as the originator would be $\frac{n}{n+m} = 1$ for any $n > 0$.

The degree of anonymity that can be achieved depends upon the power of the adversary and the amount of traffic that a node routes compared to the amount of traffic that a node generates. As long as the node is routing foreign traffic, the adversary can never be absolutely certain that this node is not the originator. The level of certainty at which the adversary will consider the identity of the originating node sufficiently established depends entirely on the specifics of the application at hand. The next section discusses how participants can individually trade anonymity for efficiency in order to make the degree of anonymity provided fit the needs of their individual applications.

2.4 Trading Anonymity for Efficiency

Suppose a node A in GAP sends a query to node B. Now assume that B forwards the query to a third node C. Furthermore, suppose that B "maliciously" uses A's address as the return address instead of its own. In this case, C can determine that B forwarded the query, and C can eventually send the reply directly to A (see figure 2). The term *indirect* is used if the node performs source identity rewriting. *forward* is used to indicate that the original sender identity is preserved.

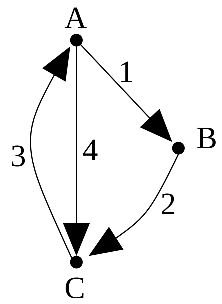

Fig. 2. Indirecting Replies

Notice that while A and C now know each other, C cannot be certain that the query originated from A, and A cannot be certain that the reply originated from C. Both A and C could simply be nodes that obey the protocol and have indirected the query (or the reply). The anonymity of A and C depends upon how many packets they indirect for others; this amount of traffic is not changed by B's "malicious" action. B has not damaged the anonymity of A or C. On the other hand, B *has* potentially damaged its own anonymity. C is now able to tell that this particular query did not really originate from B; all messages that originate from B will have B as the sender, as otherwise B would never receive the reply. C can discern that the message from A is not in the set of messages that originate from B. By excluding this message from that set, C can increase the probability that B is the originator of any of the other messages that B is currently sending. Thus, C now has a higher chance of guessing which traffic actually *did* originate from B.

Since C can also tell that A is closer to the intended recipient of the reply than B, C will send the reply directly to A. Because the reply takes the shorter path $C \to A$ instead of $C \to B \to A$, the total amount of traffic that was produced has been reduced. This "malicious" behavior of B has improved the efficiency of the network. Note that B's trade of anonymity for efficiency does not have any effect on A or C, either in terms of anonymity or of efficiency.

While this technique improves bandwidth utilization and latency for the reply by saving one indirection, the performance gain could be even higher. Because A and C needed to communicate, A may decide to send the next query directly to C. If A is likely to send many related queries (related in the sense that the responses are likely to be located on the same node), it is reasonable to assume that C will often be closer to the location of the document than B is.[1] This way, the number of hops between A and the content is decreased, speeding up the download process even further.

Let us suppose B is indirecting m queries and sending n new queries for its own user. As stated above, this would yield a probability of $\frac{n}{m+n}$ that any given query originates from B. If m is sufficiently large compared to n, this security may not be required by B. Indirecting m queries and m replies causes a great deal of work for B. If B chooses not to indirect k queries, and, instead forwards those queries preserving the original sender address, the probability that an adversary can assign to B to be the originator of a query is increased to $\frac{n}{n+m-k}$.

3 Implementation

An implementation of GNUnet with GAP is available on our website at
`http://www.gnu.org/software/GNUnet/`.

3.1 Joining the Network

A node that wants to join the network must obtain a list of peers that is large enough to contain at least one non-adversarial node. The other node must be *non-adversarial* in the sense that it is not an attacker that will only advertise nodes that are entirely under an adversary's control (in that case, the adversary could keep track of what the new node is doing by making sure that it communicates exclusively with adversarial nodes). If the new node has several initial public keys of other nodes, it is sufficient if one of these does not collaborate with an adversary. For convenience, GNUnet can automatically download addresses of several network entry points from multiple http servers.

Each node in GNUnet has an RSA key pair. The nodes use these keys to exchange 128-bit session keys that are used to establish a link-encryption infrastructure between the nodes. Blowfish is used for the symmetric cipher. Nodes periodically sign their current Internet address (together with a time stamp for expiration) and propagate this information together with their public key. Except for the initial exchange of public keys that occurs when a node joins, this exchange of public keys can also use the encrypted channels.

[1] The encoding of content in GNUnet [1] requires many related queries before a download of a single file can be completed. In other systems, queries may be related less often.

3.2 Queries and Replies

Nodes indirect queries and can thereby hide the queries they originate since source rewriting makes all queries sent by the node look uniform. Every node uses a combination of network load and other factors that are internal to the node to determine how often to indirect queries. If the general network load is high, then the node indirects fewer queries, assuming that its own traffic is already well hidden. If the network load is low, more queries are indirected.

Several queries are usually sent out in a group, potentially mixed with other messages such as content replies or peer advertisements. Grouping several messages to form a larger packet introduces delays and decreases the per-message overhead. Encrypted packets containing queries are indistinguishable from packets containing other data because grouping and padding with noise makes them equivalent in size.

Each query contains the identity of the node where the reply is to be sent. While this was originally the address of the initiator of the query, nodes that indirect the query must change this sender identity to match their own. This is because these indirected packets could otherwise be distinguished (by the receiver) from packets that originate from the node itself, which have a different return address. The node must keep track of the queries that it has indirected so that it can send the reply to the node where the query originally came from. This statefulness of routing is probably the biggest scalability issue in GAP. Mix networks avoid this issue by keeping the state encrypted in the messages itself. The problem with this approach is that it requires slow public-key cryptography to encrypt the reply blocks. Also, query messages and replies have to accomodate encrypted reply blocks. The reply blocks would either have to be signed (increasing the number of public key operations and the message sizes even further) or would be subject to manipulations by malicious hosts, which could prevent nodes from properly routing the replies. Onion routing also addresses these issues with mixes by adding state (symmetric keys). Note that the number of anonymizing tunnels used in onion routing is typically smaller than the number of messages in GAP; thus, the problem with large routing tables is significantly smaller in the case of onion routing. For GAP, we decided that bandwidth and public key operations will presumably be the bottleneck rather than memory for routing information. The implementation of GAP contains various heuristics for estimating how long to keep entries in the routing table (based on time-to-live and importance of the query). Also note that when downloading a large file, not all queries for all blocks have to be parallelized.

3.3 Source rewriting is optional in GAP

In GAP, the "malicious" behavior described in the section 2.4 is allowed. Nodes usually increase k if they receive more traffic than they are willing or able to handle. Thus, if nodes receive a great deal of traffic, they can improve their performance by reducing the number of packets they indirect. Because the replies are significantly bigger than the queries, this behavior can improve the situation (particularly for bottlenecks in the network).

It is possible that this behavior could be exploited in an attack that uses flooding of a node, A, with traffic from a malicious node M which tries to break A's anonymity. As seen before, indirecting queries that originate from the attacker M do not count toward m in the formulas given above because the adversary knows that they do not come from A. If A decides that the amount of traffic it gets is too high and then starts to preserve the sender addresses of most queries from other nodes, m may decrease so far that A can no longer protect its own n queries from being discernible by M.

The same attack also applies to *mixes* [6] where adversaries that dominate the traffic at a mix can deduce the operation of the mix. GAP attempts to guard against this type of attack by dropping queries from nodes that are generating excessive amounts of traffic.

3.4 Choosing the Next Node

Whenever a GNUnet node receives a query, it decides how many nodes it will send the query to based upon its load (CPU, network), the local credit rating of the sender [10] and a random factor. The number of nodes that will be chosen to receive the query could be zero. The nodes that will receive the query are then chosen from the list of nodes that the node has established connections with (using a biased random selection process). The selection process is biased toward nodes where the hash of the hostkey is close to the query using some metric. This is a variant of the algorithm used by Pastry [15, 2]. The selection process also takes into account recent network activity, with preference given to hot paths. Furthermore, queries are used to pad messages to uniform size, making use of bandwidth that would otherwise be wasted to transmit random noise.

The query is not sent immediately to the next group of nodes. Instead, it is put in a buffer that queues data that is to be sent to each selected node. The buffer is sent whenever it is full, or when a randomized timer goes off; it can also be entirely discarded if the node decides that it is too busy (note that the protocol does *not* guarantee reliable delivery).

This behavior does not directly leak any information to an attacker, as it is independent of the original sender; in fact, the originator has the same chance to receive the indirected query as everyone else has. Replies are sent back on the path that the query took originally, with the potential for shortcuts if intermediaries do not perform source rewriting and advertise another peer as the receiver of the reply.

Various attacks can be applied by a powerful adversary to almost any message routing scheme. One such attack would be a timing analysis that looks at the time that passes between query and reply. This time could be used to estimate the distance to the sender. GAP defends against this attack by introducing random amounts of delay for the query and the reply at every step. Furthermore, nodes choose the routes for a query at random, making the timing results hard to reproduce. Also, as with Freenet [5], content migration that can be caused by a request constantly changes the location of content, making it even harder to pinpoint its exact location.

Routing in distributed hash tables such as Chord [20] and Pastry [15] is subject to attacks where the adversary can predict the route of a query. The adversary can use the likelihood that a given peer will route a query to break anonymity. GAP makes this harder by adding significant amounts of variability to the routing process. One method GAP employs to add this variability is to modify the nature of cover noise sent out by peers. Instead of always generating and injecting noise into the network, peers also look at their lists of pending (unanswered) queries and choose queries to forward to additional hosts in the host list who have not yet been sent these queries by this peer. This suffices to make it plausible for any peer to receive and then route any query. The disadvantage is that GAP cannot guarantee $O(\log n)$ steps for finding a reply.

3.5 Looping Queries: Hops-to-Live and Time-to-Live

So far, one field that is contained in each query message, the time-to-live, has not been discussed. This field is used to bound the extent of the network that the query traverses. In particular, it is needed to prevent queries from looping indefinitely. More deterministic routing algorithms, such as Pastry [15] or Chord [20], do not have this problem since by design loops cannot occur. The more random nature of routing in GAP can cause queries to loop back to a peer that has already forwarded them. Thus, loop detection is a requirement to prevent queries from staying in the network for too long. Detecting loops by simply remembering which queries have been routed in the routing table is not feasible since the routing tables are typically unable to keep enough state for all queries in the network. Therefore, each query contains a field, the time-to-live, which bounds how long a query will be routed.

Freenet [5] uses a very similar scheme. Their routing algorithm deploys a hops-to-live field in every query. Every node on the path decrements the hops-to-live value by one until it reaches 1. Then, the query is forwarded only with a certain probability. This is needed to prevent an adversary from sending queries with the lowest hops-to-live value in order to see which nodes send replies; those that reply can be determined not to have forwarded the queries, since that would exceed the hops-to-live value, and the conclusion could then be drawn that nodes which reply are in fact the nodes storing the requested content. The Freenet scheme is problematic in that the adversary can use many probes to test whether a peer only replies with a certain probability or with certainty. While content migration is likely to actually always *make* the peer under investigation *a* node storing the content, this attack may still work if the adversary knows a set of correlated queries to probe multiple times with fresh content. In GNUnet, the block-encoding of the files would give any adversary an easy way to produce multiple independent probes to run this probabilistic attack.

For this reason, GAP does not use a hops-to-live field. The semantics of GAP's time-to-live field are slightly different. The time-to-live field is a relative time that specifies how long peers should route replies for this query. When a peer receives a query, it adds the time-to-live to its local time. This new absolute time is then used to produce a total ordering for all the queries that the peer receives. The fixed number of routing slots are assigned to the latest queries according to that total order. Note that the relative time-to-live field in a query can be negative, and that a peer may still route these queries if the replaced routing table entry is sufficiently old. The total order of the queries guarantees that a query can not loop in the network.

The implementation needs to be careful in routing replies to received requests; a node that is replying to a query A should only send replies after the routing table slot for query A has been allocated for long enough to make it plausible that this node has received a response for A from another peer. Otherwise, the adversary could mount an attack where a series of queries M_i is used to overwrite a routing table entry for A (and its subsequent reply, of course). In this situation, the victim can only claim a plausible delay for the short time that A was in the routing table. Any additional delay can no longer deceive the adversary because responses to query A from other peers could obviously not be routed back to the adversary after the routing table entry was replaced by one of the M_is. GAP defends against this attack by sending replies only after the query has stayed in the routing table for an amount of time that makes it plausible for the peer to have received a reply from elsewhere. Thus, even if a node has the requested content, if A disappears from the routing table, the content will not be sent. It should be noted that this solution may appear to leave the system open to denial of service attacks which would keep the node from being able to route any content at all. However, GNUnet's economic model [10] ensures that the traffic generated by flooding a node eventually causes the flooded node to drop traffic from the malicious node; thus, the only routing table entries which will be overwritten in such an attack will be those from other abusive nodes.

3.6 Measuring Anonymity

The discussion in this section assumes that the routing in GAP is sufficiently random to prevent the adversary from establishing that a node has a low probability of being chosen to route a specific query. Furthermore, the adversary is not supposed to be able to correlate queries (e.g. by the fact that they belong to the same file). This assumption is often reasonable since only by having the plaintext of the file it is possible to make that correlation. If the adversary does not know the exact details of the file that is being accessed, queries cannot be correlated.

From the explanation of how efficiency can be traded against anonymity in section 2.4, it should be clear that the degree of anonymity that GAP offers is configurable. If a node injects a query from the user into the network in only 1 of 1000 indirected transactions, it will surely be more anonymous than if it does so in 1 out of 10. Note that there are two parameters here. First, a node determines how much bandwidth it has available. Once that bandwidth quota is exceeded, is ceases to perform source rewriting on the additional traffic. Next, it chooses how anonymous a specific download or search needs to be. Given the currently available traffic (which may be anywhere between the minimal amount of background noise and the available bandwidth), it can then inject the queries at a certain frequency into the network.

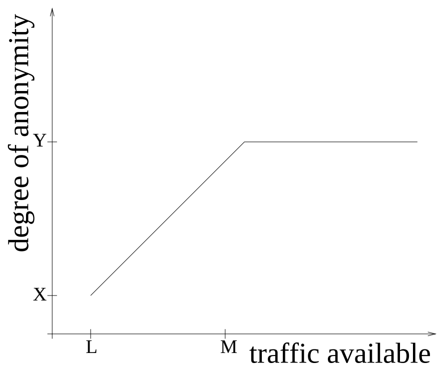

Fig. 3. Anonymity and Traffic

Figure 3 illustrates this relationship. Suppose that the noise in the network that is routed through the node is at least L kbps and that there are no active adversaries contributing to the traffic in the network. If a node on the network sends requests at a constant rate r, the probability p that a packet originates from the node is $\frac{r}{L} = p = \frac{1}{X}$. If the available foreign traffic on the network increases above the basic noise level, the anonymity of the node also increases. At some point, the network traffic may reach the capacity M of the node; the node stops routing the additional traffic, and the maximum degree of anonymity Y that the node can achieve (while sustaining a data rate of r) is $\frac{r}{M} = \frac{1}{Y}$. If a node needs more anonymity than the current volume on the network can guarantee, it must slow down the rate r at which it requests data from the network.

With adversaries that have unknown power, the exact degree of anonymity and the probability with which the adversary can determine the originator cannot be computed in this manner in practice. The reason is that this would require knowledge about how much traffic is controlled by the adversary. In fact, the more bandwidth an adversary has, the less anonymity can be provided since the adversary can flood the network with messages. This attack also applies to other anonymizing networks [6]. As stated in section 1, if the adversary controls the traffic of the entire network, it can always determine with certainty from where an action originated.

Estimating the number of adversary-controlled hosts in an open network like GNUnet is obviously very difficult. Proving the authenticity of a remote microprocessor [11] and (ultimately) the trustworthiness of a remote machine are still open problems. GNUnet compensates for the impossibility of guaranteeing anonymity against very powerful attackers by providing deniability [1]. Even if a powerful adversary can determine who sent a message, the deniable encoding and searching mechanism for content (see [1]) ensures that the adversary may still be unable to determine what the message is about.

A situation similar to that in which an adversary floods the network with known traffic occurs when there is only the minimal amount of traffic L on the network, allowing for only a minimal degree X of anonymity. This situation is not as problematic as it may sound since peers can start with a very slow download rate r. This will increase the network load on the network, especially since idle nodes are likely to spread a query much further than busy nodes would. Thus the load on the network will quickly rise (at least in the proximity of the peer starting the download), allowing the peer to increase r. Since GNUnet's content encoding [1] has the inherent property that a downloading node can initially only send a small set of queries (due to the tree-structure of the encoding), the requirement that a node must start with a small r to achieve anonymity until the network load rises is in practice what the code must do anyway. The network always has some level of background noise for key exchange and node advertisement propagation that should be sufficient for a node to hide the origin of a single query.

4 Related Work

Indirection, delays and noise have been used to achieve anonymity long before GAP [3, 21, 12]. Interestingly, the traditional perception has focused on decoupling the identity of the receiver from the identity of the responder by placing an anonymizing service like `anonymizer.com` in the middle. While this approach works for weak adversaries like web-site operators that only see the intermediary's IP in their logs, it does not help against adversaries that can perform traffic analysis and even become a part of the anonymizing infrastructure.

Indirecting *all* communications is also very costly. For example, in Freenet [5], the number of indirections is determined by the length of the search path that the query takes until a node that has the content is found. Thus, if the search path has length l, there are l transfers of the content. The traffic overhead is then approximately $(l-1) \cdot s$ where s is the size of the content. Freenet attempts to counter this problem by using clever routing and content migration strategies to minimize l. The design does not allow the user to trade anonymity for efficiency.

Other systems, like Crowds [13], allow the user to set the number l of indirections that the system should aim for. While the traffic overhead is again $(l-1) \cdot s$, the l can be adjusted. The authors describe a network where n nodes indirect requests with a probability p_f. The degree of anonymity that Crowds offers is defined as the probability that a node collaborating with the adversary receives a communication from the node that actually sent the request. As with

GAP, the (unknown) strength of the adversary makes it impossible in practice to determine the exact degree of anonymity that is achieved.

The analysis of Crowds assumes that all nodes are equally active and thus equally suspicious. Even if the adversary has only c nodes under control, traffic analysis may give much better data about which node is responsible for the query – even under the assumption that traffic between the $n - c$ non-malicious nodes cannot be decrypted. Sending noise to make the traffic analysis harder is not discussed and would, of course, increase the network load beyond $(l - 1) \cdot s$.

In Crowds [13], a probabilistic attack is described that can be used to infer the identity of the initiator. The attack is based upon the idea that an adversary could attempt to correlate multiple transfers over the network and then infer the initiator who would have a higher-than-average chance of being the sender. Hordes [19] attempts to make this attack more difficult by using multicasts for replies, choosing a different path through the network from initiator to responder than the path back from responder to initiator. The probabilistic attack described requires not only that the adversary control nodes that are participating in the network, but also that the adversary is able to relate multiple transactions. In GNUnet, transactions cannot be correlated since neither the query nor the reply reveal any context unless the adversary knows exactly what content is transmitted. Thus, in this case, a probabilistic intersection attack over time will not help to reveal the identity of the user.

Another distributed network with anonymity goals is P5 [18]. P5 uses broadcasts (or multicasts) for data transfer. It achieves scalability by dividing the network into a tree-hierarchy of small broadcast groups. For sender-anonymity, P5 nodes constantly broadcast noise in order to disguise active phases. Receiver addresses in P5 are addresses of these broadcast groups. A peer in one of the broadcast groups can advertise any group that is located higher up in the tree as its address. Each group forwards all received messages to all child-groups. Messages are dropped if the network load gets to high. Messages that have been addressed to a less specific group will be dropped earlier. Thus by advertising a less specify broadcast group the anonymity of a peer can be increased since the number of groups that may receive the reply is larger. Advertising a more specific group on the other hand improves the latency of the peer since fewer messages will be dropped. The overall traffic in P5 is assumed to be always at the maximum of what the peers can sustain. Thus P5 allows peers to trade-off anonymity for latency but not for bandwidth.

DC-net is an anonymous network based on the Dining Cryptographers protocol [4] that requires each node to communicate with each other node for every message. Onion Routing is based upon Chaum's mixes [3]. For each message, the user chooses a path though the mix-network and encrypts the message with the public keys of each of the mixes. The message is then sent along that path and as long as not all nodes on the path are compromised, the identity of the initiator is not directly exposed.

In most of the networks above, the anonymity of a node depends upon the behavior of the other nodes in the network. In GAP, each node is able to individually choose whether to exchange portions of its own anonymity for its own efficiency without impacting the security of other nodes.

Comparing Anonymous Protocols

We now compare GAP to related anonymity protocols, focusing on P5 [18], Freenet [5], DC-Net [4], mixes [3], Onion-Routing [21], Crowds [13], Hordes [19] and a simple proxy. Note that the designs compared here address very different applications, from interactive anonymous browsing to high-latency anonymous mail. The systems differ widely in their respective costs and benefits and it is thus difficult if not impossible to make a fair comparison.

P5, DC-Net and Hordes rely on broadcasts or multicasts for communications; all of the other protocols use unicast. GAP and Freenet use unicast, but nodes may choose to route the same message to multiple nodes (which could be seen as application-level multicast); however, these duplicates are actually *processed* by each recipient (as opposed to most anonymizing multicast schemes, in which every recipient but one just sees the traffic as noise).

Most anonymous protocols achieve at least some form of initiator (or sender) anonymity. In the case of a proxy, the sender is only anonymous in relation to the responder, not in relation to the proxy, as the sender's identity and data are directly exposed to the proxy. In mix networks, if the initiator is not part of the mix network, initiator anonymity is again only partial for the same reason. In these anonymizing networks, there is a single point of failure (typically the first hop) at which a node controlled by the adversary can be used to fully expose the initiator. One could argue that the message is encrypted and can only be decrypted by the final recipient and, thus, that anonymity is not violated. But even when the contents are not exposed, it is still possible to correlate the transaction with the sender. While our definition of anonymity allows that the adversary may see participation in the network, it does not allow the adversary to determine that a peer actually initiated a transaction.

In P5, Freenet and GAP responder anonymity is achieved in addition to sender anonymity. This is because the reply gets anonymized as it goes back through the network and the responder is not known to the initiator. The responder is also anonymous in DC-Nets, since the protocol only deals with one-way broadcast messages (symmetry). Mixes and Onion Routing can have anonymous responders when reply-blocks are published, but not all responders are anonymous by default. Crowds and Hordes require prior knowledge of the responder by the initiator, thus making responder anonymity impossible unless a public rendezvous point is used [16]. Mixminion [6] discusses attacks on mix networks.

Some protocols allow nodes to trade anonymity for efficiency in order to improve performance. With a simple proxy, DC-Net or Freenet, the degree of anonymity is fixed by the circumstances. In Crowds and Hordes, a node can choose to reduce the number of indirections on the communications path (or the size of the multicast group in the case of replies in Hordes), increasing the efficiency of the network. Note that a node that reduces its number of indirections reduces the load on other nodes in the network, but not the load on itself (thus reducing incentive to make this trade-off). Also, joining a larger multicast group in P5 or Hordes affects other nodes that will receive additional useless multicast traffic. With GAP, a node that indirects a query but tells the recipient to short-cut the response actually is able to reduce its own load (since it does not have to route the reply) without having an impact on the anonymity or load of any other node.

Attackers that actively participate in the protocol are hard to defend against. The best defense is a DC-Net where a single non-collaborating node can thwart an attack against any node except itself. In the case of a proxy, the only node in the network must be trusted; thus, a "collaborative" attack will always succeed. In P5, an adversary must control the respective multicast group (which can be small) in order to ascertain the identity of the recipient. Freenet, Onion Routing and Crowds (and to a lesser degree Hordes) are vulnerable to the probabilistic attack over time described in [13, 19]. In GAP, the adversary must control a significant portion of the bandwidth that a node is routing in order to be able to determine with reasonable probability which traffic was initiated by that node.

Freenet and GAP are anonymity protocols that have built-in content location capabilities. P5 requires knowledge about the address of a multicast group containing the responder. All other systems compared require prior knowledge by the initiator about the address of the responder.

P5 and Mixes require one or more public key operations *per request* on each node processing the message; the other systems require only symmetric operations for each request after an *initial* public key exchange that is used to establish link encrypted channels.

4.1 Other anonymizing systems

Tarzan [9] is a peer-to-peer system that allows IP-level anonymization of Internet traffic based on onion routing. Tarzan cannot offer responder anonymity. Worse for the user is probably the fact that applications that use Tarzan are typically not aware of the anonymization requirements. Users are likely to use Tarzan in combination with applications such as web-browsers or mail clients that will often allow the responder to identify the user due to information leaked in the higher-level protocol that is tunneled in the anonymizing IP layer infrastructure.

Morphmix [14] improves on mixes by using a witness to help select the path through the network. In this way, the initiator does not need to know a large set of peers in order to build an anonymous tunnel to the receiver. Like Tarzan, Morphmix is a peer-to-peer network where the set of mixes is large and dynamic, as opposed to static sets that were used in previous architectures.

Peer-to-peer networks like Tarzan, Morphmix and GNUnet are faced with the problem that there is no operator that is responsible for keeping the network running. This makes these open networks an easy target for adversaries that are powerful enough to disrupt Internet connections. Individual peers maybe isolated from all other peers on the IP level and be only allowed to connect to nodes controlled by the adversary. Peer-to-peer users would probably not even notice this type of attack since peers are always considered to be unreliable, thus not being able to reach a large fraction of the peers would be considered normal. GNUnet attempts to make this attack harder by providing a transport abstraction that can tunnel GAP messages in other protocols, such as SMTP [8], making it harder for the adversary to disrupt all communications without being noticed.

4.2 Measuring Anonymity

GAP's model for anonymity is probability based. [7, 17] have described why an approach based on probabilities does not prevent individual peers from sticking out. A peer that has only a probability of 5% to be the originator of a query may still be an easy target if all other peers have a significantly lower probability, say 1%. [7, 17] proposed an entropy based anonymity metric that takes the probability distribution into account. In this information theoretic approach, the resulting metric expresses how many bits of information the adversary would need to expose a peer. It turns out to be difficult to apply this metric to GAP since computing the entropy of the network requires a global view of the network that typical peers do not have. Thus peers can only attempt to reduce their individual probabilities, but they can not do this based on knowledge about the global distribution.

The entropy based metric shows that all anonymizing protocols need to attempt to balance probabilities as much as possible. While this requirement was taken into account when GAP was designed, future work will be needed to formally determine the amount of information leaked by the protocol for a given type of adversary.

5 Future Work

The discussion of GAP in this paper has focused on a single query sent by the initiator. In practice, downloading a file in GNUnet will require multiple queries. While an active adversary as described in the introduction cannot correlate these queries, a content-guessing global active adversary that has obtained the file that the user is downloading (e.g. by downloading it for himself) will be able to correlate these queries. If the adversary is able to correlate queries, it may be possible for the active adversary to infer the identity of the initiator probabilistically. Better defense against this type of attack is an open issue.

In this paper we have also assumed that the mixing performed by the peer is *optimal*. While improving mix algorithms is basically an orthogonal issue to indirection-based anonymization protocols, the routing decisions made at each peer impact the mixing strategy; this makes this type of mixing slightly different in comparison to traditional mix networks in which the next hop is determined entirely by the message itself. In general, scalable anonymous routing strategies are still an open problem. We hope to evaluate the scalability of the routing strategy presented in this paper in the future.

Content migration is another issue. If a peer sends out local content that does not correspond to any query that it has recently received, the recipient knows that the sender stored the content. One solution to this is to randomly forward content on a frequent basis that the peer received but did not store locally.

Another approach may be to forward content only when the peer is about to remove it from its local store. The use of onion routing for content migration may be another solution which is more costly, but less limiting.

6 Conclusion

This paper presented an adaptive indirection scheme that allows each node in an anonymizing network to individually trade anonymity for efficiency without negatively impacting other nodes in the network. A new perspective on how to perceive anonymity from the perspective of strong adversaries that monitor all network traffic and potentially even participate in the anonymizing network was described. This perspective was then used to derive an anonymizing protocol which allows participants to choose not to perform source-rewriting (in order to increase efficiency) without violating the protocol. The implementation of GAP in GNUnet uses the current network load to exchange increasing anonymity for efficiency, allowing the improvement of scalability.

A final thought: while trading anonymity for efficiency may mean being less anonymous in the short term, increased network efficiency may mean greater usability. This may lead to more users and therefore eventually to *increased* anonymity.

Acknowledgements

We would like to thank the anonymous reviewers for constructive feedback and Roger Dingledine for helpful discussions. Additionally, we are grateful to Jan Vitek and Victor Raskin for support.

References

1. Krista Bennett, Christian Grothoff, Tzvetan Horozov, and Ioana Patrascu. Efficient Sharing of Encrypted Data. In *Proceedings of ACISP 2002*, 2002.
2. M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables. In *Proceedings of the International Workshop on Future Directions in Distributed Computing*, 2002.
3. David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.
4. David Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptography*, pages 65–75, 1988.
5. Ian Clarke, Oscar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009, 2000.
6. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE Symposium on Security and Privacy*, 2003.
7. Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Proceedings of the Workshop on Privacy Enhancing Technolgoies*, 2002.

8. Ronaldo A. Ferreira, Christian Grothoff, and Paul Ruth. A transport layer abstraction for peer-to-peer networks. In *Proceedings of GP2PC 2003*. IEEE Computer Society, 2003.
9. Michael J. Freedman and Robert Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, D.C.
10. Christian Grothoff. An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks. *Wirtschaftsinformatik*, June 2003.
11. Rick Kennell. Proving the Authenticity of a Remote Microprocessor, 2003.
12. Michael Reed, Paul Syverson, and David Goldschlag. Proxies for anonymous routing. In *12th Annual Computer Security Applications Conference*, pages 95–104, December 1995.
13. Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.
14. M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES), in association with 9th ACM Conference on Computer and Communications Security (CCS 2002), Washington, DC, USA*, November 2002.
15. Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218, 2001.
16. Vincent Scarlata, Brian Levine, and Clay Shields. Responder anonymity and anonymous peer-to-peer file sharing. In *Proceedings of IEEE International Conference on Network Protocols (ICNP) 2001.*, 2001.
17. Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *Proceedings of the Workshop on Privacy Enhancing Technolgoies*, 2002.
18. Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A Protocol for Scalable Anonymous Communication. In *IEEE Symposium on Security and Privacy*, 2002.
19. Clay Shields and Brian Neil Levine. A protocol for anonymous communication over the Internet. In *ACM Conference on Computer and Communications Security*, pages 33–42, 2000.
20. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
21. Paul Syverson, David Goldschlag, and Michael Reed. Anonymous Connections and Onion Routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, 4–7 1997.