An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks

Dennis Kügler

Federal Office for Information Security
Godesberger Allee 185-189
53133 Bonn, Germany
Dennis.Kuegler@bsi.bund.de

Abstract. Peer-to-peer networks are a popular platform for file sharing, but only few of them offer strong anonymity to their users. GNUnet is a new peer-to-peer network that claims to provide *practical* anonymous and censorship-resistant file sharing. In this paper we show that GNUnet's performance-enhancing features can be exploited to determine the initiator of a download. We also present an efficient filter mechanism for GNUnet. Assuming that content filtering is legally enforced, GNUnet can be censored at a large scale.

1 Introduction

Peer-to-peer networks are widely used to share all kinds of digital content with other users. It was first demonstrated by Gnutella [FP00] that such peer-to-peer networks can be organized in a decentralized manner, so that there is no practical way to shut down the network. Therefore, Gnutella and related networks have become a popular platform for sharing legal and illegal content (i.e. copyrighted or subject to censorship). While searching for content on Gnutella is relatively anonymous, responses to search queries are non-anonymous as the IP address of the offerer is always exposed. Thus, it is possible to prosecute users who break the law. As this clearly discourages users to share illegal content, censorship is indirectly applied.

The idea of constructing a censorship-resistant network goes back to the idea of the Eternity Service [And96], an attack-resistant storage medium. Freenet [CSWH01,CMH+02] was the first approach that tries to combine both types of networks: an anonymous, decentralized peer-to-peer network and a censorship-resistant network. Freenet can perhaps be described best as "distributed file system storing replicated content in an obfuscated form". One obvious drawback of Freenet is that it "forgets" infrequently requested content, which not only contradicts censorship-resistance but also makes the retrieval of content inconvenient as downloads often fail.

Recently, GNUnet [GPB+02,BGHP02,BG03,Gro03] has been proposed, an alternative approach that claims to be much more practical and efficient. Therefore, we analyze GNUnet in this paper and point out some weaknesses. We start

in Section 2 with a short introduction to GNUnet. Then we show how anonymity can be degraded: We introduce our shortcut attack in Section 3 and consider the efficiency of this attack in Section 4. Afterwards, we discuss in Section 5 how censorship can be applied to GNUnet. Finally, we conclude our paper in Section 6.

2 A Description of GNUnet

GNUnet consists of nodes that communicate with each other. Every node chooses a key pair, that is used for identification, authentication, and to encrypt the communication between the nodes. To hide which nodes are indeed communicating with each other, GNUnet uses a MIX-like [Cha81] approach: nodes are intermediaries that send messages to other nodes.

In the following, we present the encoding scheme and the routing mechanism of GNUnet, which will both be exploited for our attacks.

2.1 Encoding[1]

GNUnet transfers blocks of fixed size (1Kb). There are three types of blocks: *Data Blocks* (DBlocks), *Indirection Blocks* (IBlocks), and *Root Blocks* (RBlocks).

Files of arbitrary size are split into DBlocks. For every DBlock D_i of a file the 160 Bit RIPE-MD hash value $H(D_i)$ is calculated. Then a tree of IBlocks is calculated recursively, where every (leaf) IBlock contains up to 25 query-hashes (see below), a superhash and a CRC32 checksum. The superhash of an IBlock is calculated as the hash of the concatenation of all query-hashes in the blocks below this IBlock. Every block B_i is encrypted with the hash value of its own content to $E_{H(B_i)}(B_i)$ and is stored under $H(E_{H(B_i)}(B_i))$. Thus, the query-hashes included in the IBlocks are pairs of the form $(H(B_i), H(E_{H(B_i)}(B_i)))$ that are necessary to retrieve the remaining blocks.

Finally, the RBlock is generated, containing a description of the file and the query-hash to retrieve the root of the tree of IBlocks. One or more keywords are used to encrypt and to store the RBlock. For every keyword K_j the RBlock R is encrypted to $E_{H(K_j)}(R)$, then both the encrypted RBlock and $H(H(K_j))$ are stored under $H(H(H(K_j)))$.

2.2 Routing

Retrieving content from GNUnet is a two step process:

1. **Discover RBlocks:** RBlocks are discovered using search queries containing a list of triple hashed keywords $H(H(H(K_j)))$. A node that receives a search query and has a RBlock stored under one of those values returns the

[1] The encoding scheme used by GNUnet to store and retrieve files has changed recently. The original scheme can be found in [BGHP02].

encrypted RBlock $E_{H(K_j)}(R)$ together with $H(H(K_j))$ to prove that it indeed possesses the requested content stored under this keyword. The node that has issued the query is able to decrypt the RBlock using $H(K_j)$.
2. **Download Content:** After an RBlock is discovered, the node can download the corresponding file by issuing several download queries. The node first uses the query hash included in the RBlock to retrieve and decrypt the root IBlock. Then the node uses the query-hashes included in the root IBlock to retrieve and decrypt all additional IBlocks and finally, the query hashes included in the leaf IBlocks are used to retrieve and decrypt the DBlocks of the file. The node may also use multi-queries to retrieve several I- or DBlocks at once. A multi-query consists of several download hashes plus the corresponding superhash to speed up lookups.

Thus, there are two types of queries: search queries and download queries. Both types of queries contain three additional values that are used to process the query: a return address, a priority, and a time-to-live (TTL).

Processing Queries. Each node sets up a message queue for every neighbor node. Before a query or a response is sent to a neighbor node, it is temporarily stored in the queue representing this node. Each queue is flushed at random intervals, then all messages waiting in this queue are sent to the corresponding node. A node that receives a query proceeds as follows:

1. If the requested content (RBlock, IBlock or DBlock) is found locally, a response is enqueued in the message queue corresponding to the return address given in the query. Otherwise (but always in the case of a search query), the query is sent to some other nodes.
2. Depending on the priority and the available bandwidth the node randomly selects n neighbors and enqueues the query in their message queues. The query is adjusted as follows:
   - The new priority is calculated as $\frac{p}{n+1}$, where p is the old priority.
   - The node either keeps the return address of the predecessor node or replaces it with its own address. In the latter case, the node adds the query to its local routing table.

It is important to mention that intermediary nodes are not aware which queries belong together (with the exception of multiqueries). Therefore, queries are routed individually, i.e. there is no static path that is used to route related queries. However, the selection process is biased towards neighbors that have responded recently. This is discussed in more detail in Section 4.

Credit. GNUnet is based on an economic system, where every node associates any other known node with a local value called *credit*. The goal of this economic system is to prevent flooding attacks by limiting the resources available to an attacker:

Let A and B be two nodes. A associates B with credit c_B and vice versa. After B received a query with priority p from A, it first checks whether A has enough credit.

- If $c_A = 0$ the query is dropped, otherwise set the new priority p' to p.
- If $p' > c_A$ the priority of the query is reduced to $p' = c_A$.

Then B processes the query and charges A by reducing A's credit to $c_A = c_A - p'$. However, if B has excess bandwidth, it may decide not to charge A. The decision whether and how much B decreases A's credit is invisible to A.

If B returns a valid response, B expects A to increase his credit to $c_B = c_B + p$. Again, A may decide not to pay B for returning a valid response. The decision whether and how much A increases B's credit is invisible to B.

Zero Priority Queries. If a node indirects a query to another node without charging the preceding node, it may assign zero priority to the forwarded query, because it does not want to be charged by the following node.

Thus, a node that receives a zero priority query cannot gain any credit when responding to this query. But if the load on the node is low, responding to a zero priority query does not hurt this node.

3 Deanonymization in GNUnet

An informal definition of anonymity can be found in [PK01]:

> *"Anonymity is the state of being not identifiable within a set of subjects, the anonymity set".*

We consider a network of several nodes, where some nodes may be malicious as shown in Figure 1. A malicious node is behaving correctly, but it tries to acquire as much information about the network as possible. Furthermore, we assume that all malicious nodes are controlled by the same attacker.

3.1 Identifying Sessions

To successfully apply our attacks, we first have to identify sessions. This can be done very easily: GNUnet splits larger files into small DBlocks. If we know the corresponding IBlocks, we are able to identify DBlocks that belong together. Thus, the attacker prepares a dictionary of interesting keywords and their corresponding hash values, queries the network with those keywords and receives a number of encrypted RBlocks $E_{H(K)}(R)$ which he is able to decrypt. Then the attacker can also retrieve the corresponding IBlocks. As the IBlocks contain the query-hashes of all DBlocks, the attacker is able to use his malicious nodes to observe the following:

- Queries containing one of the prepared keywords.

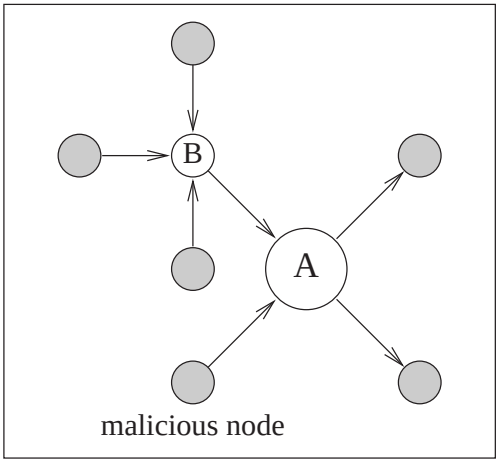

Fig. 1. Malicious nodes in a network.

- Queries for known I- or DBlocks.
- Responses containing known I- or DBlocks.

Note: The attacker can also observe responses to queries with non-trivial keywords if the file also has been inserted (probably by a different user) under trivial keywords. Inserting content under a non-trivial keyword renders the content useless as only few people are able to retrieve the content again. This is a contradiction to the idea of a censorship-resistant network, where everybody should be able to access any possible content. Therefore, we expect that most content is available to the attacker.

An attacker can exploit the linkability of related I- and DBlocks to execute two types of attacks on the anonymity:

1. **Intersection Attacks:** The intersection attack [BPS01] exploits the fact that not all users of a MIX network participate in every batch. Thus, all users that have not contributed to a batch containing linkable traffic, can be subsequently removed from the anonymity set.
2. **Predecessor Attacks:** The predecessor attack [WALS02] extracts information from the setup of dynamically chosen, linkable paths, where each node randomly selects the following node out of the set of all nodes. To determine the initiator, the attacker logs the preceding node. Besides the initiator, every node should be logged with equal probability. Thus, the expected number of times the initiator is logged is greater than the expected number of times any other node is logged.

The predecessor attack has been used in [WALS02,Shm02] to successfully attack Crowds [RR98]. In GNUnet however, the assumption that every node is logged

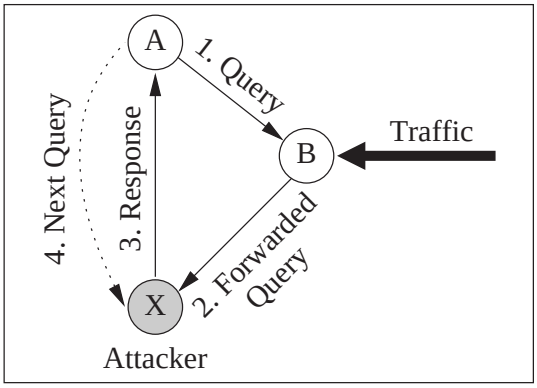

Fig. 2. The shortcut attack.

with equal probability is not satisfied, as each node only sends queries to its direct neighbors, not to all nodes. Therefore, we will not discuss this attack. However, if the attacker should be able to establish a connection to all nodes in the network, then he will receive the same query from many nodes. In this case, the predecessor attack should be reconsidered, as the attacker will receive the query more often from the initiator node than from any other node.

3.2 The Shortcut Attack

Our *shortcut attack* is a special intersection attack that exploits GNUnet's shortcut feature. Depending on the current load, a node uses this feature to decrease both bandwidth utilization and latency:

- If the node is idle, queries are *indirected*. The node replaces the return address with its own address.
- If the node is busy, queries are *forwarded*. The node does not change the return address of the preceding node.
- If the node is very busy, queries are *dropped*. The node does not process the query.

Each node monitors its own outbound traffic to determine how busy it is. A node that has too much outbound traffic simply optimizes the routing of responses by removing itself from the path.

It is claimed by the authors of GNUnet that shortcuts do not hurt anonymity: A node that stops indirecting some traffic can only hurt itself, as hiding its own activities is worse with lower (outbound) traffic. We exploit this optimization to discover the initiator of a query as shown in Figure 2, where node X is an attacker who receives queries from node B that have been issued by node A.

Basic Shortcut Attack. The attacker tries to entice an attacked node to forward traffic instead of indirecting it by increasing the node's outbound traffic:

1. The attacker floods B with high priority queries. Due to the promised credit B will try to respond to those queries. The attacker may hide his attack by using several malicious nodes to flood B.
2. The load of B increases and the node decides not to indirect but to forward (or to drop) some queries. When B forwards queries, it does not overwrite A's return address, so that the attacker will learn that A is the node preceding B.
3. The attacker directly responds to A. As the attacker is providing valid content, A will send further (perhaps unrelated) queries to the attacker. The more the attacker responds to A's queries, the higher is the probability that A sends more queries to the attacker.

The basic shortcut attack is very inefficient as the attacker is not able to control which query the attacked node forwards. Given that the attacked node may be connected to hundreds of other nodes, it is very unlikely that just the query the attacker is interested in is forwarded to him. Furthermore, credit is required at the attacked node to increase its load. As credit is not global, the attacker has to gain credit at each attacked node before the attacker can proceed. To gain credit at a node, the attacker has to respond to several queries.

Thus, the basic shortcut attack is only applicable, if paths are very static. As this is not the case, we have to remove the requirement to flood the attacked node.

Improved Shortcut Attack. A precondition for the improved shortcut attack is that the attacker is somehow aware of the topology of the network. Then the attacker can simply guess that A is closer to the initiator than B. The attacker therefore connects to A and returns responses to queries received from B directly to A. If the attacker has guessed correctly, then he provides valid content to A and attracts more queries from this node. Furthermore, B is perhaps unable to respond to A's queries without the help of the attacker, which again will increase the probability that A indirects more queries to the attacker.

4 Applying the Shortcut Attack

To apply the improved shortcut attack, the attacker uses the following strategy get subsequently closer to the initiator:

1. The attacker connects to all neighbors of the current preceding node and waits for linkable queries.
2. The attacker uses a statistical test to determine the neighbor that is closer to the initiator.
   (a) If a closer node exists, it is selected as new current node.

(b) Otherwise the current node must be the initiator.

In the following, we show how the attacker can decide which of the tested nodes is closer to the initiator. As the success of the test depends on how GNUnet routes queries, we first discuss the efficiency of the attack when only random routing is used, which is GNUnet's basic routing mechanism. Then we show that the attack becomes much easier when GNUnet tries to improve performance and additionally uses hot path routing.

4.1 Testing Neighbor Nodes

Assuming that the attacker receives linkable queries with a higher probability from the node that is closer to the initiator than from the other tested nodes, the attacker can use a distribution-independent statistical test (e.g. Wilcoxon-Mann-Whitney or Run-Test) based on the sequence of received linkable queries to determine the node that is closer to the initiator.

Exact Test. For each pair of nodes (A, B) we have to decide, whether both nodes are equal likely to receive linkable queries (null hypothesis) or not (counter hypothesis). Thus, we have the following hypotheses:

$$H_0\text{: } P_A = P_B$$
$$H_1\text{: } P_A \neq P_B$$

To be able to accept or reject H_0 with significance α, several linkable queries have first to be received from those nodes. If the attacker rejects H_0 he is only able to state that both nodes are not equal likely to receive linkable queries. But as we assume that the closer node receives more linkable queries, the attacker chooses to continue the attack with the node that has delivered more linkable queries.

Simplified Test. Although the exact test should provide good results even if it is difficult to distinguish the tested nodes ($P_A \approx P_B$), the attacker may want to use a simplified test. The exact test is expensive for the attacker as several linkable queries first have to be received from the tested nodes before the attacker can accept or reject H_0 with significance α.

If we additionally assume that $P_A \gg P_B$, if A is closer to the initiator than B, we can simplify the test. The attacker guesses that the node that first delivers a linkable query is closer to the initiator. The attacker chooses this node to continue with, but has to verify his guess afterwards. Therefore, the attacker has to monitor the tested nodes until enough queries are received to either accept or reject H_0 with significance α. If it turns out that the attacker has to unexpectedly accept H_0, the attacker has to chose another node based on the results of the exact test, because his guess was wrong.

4.2 Random Routing

With random routing the nodes to receive a query are selected randomly. Let k be the number of neighbors of node A. For every query the node randomly chooses m of the k neighbors and sends the query to them. Thus, the probability that a certain neighbor is randomly selected is uniformly $P_{rnd}(A) = m/k$ for all neighbors of A. Note that the probability P_{rnd} is node-specific and unknown to the attacker.

Testing. To test which of the neighbor nodes B_i of the current preceding node A is closer to the initiator, we compare the number of queries received from those nodes. Let q_A be the number of linkable queries the attacker receives from A. Then the attacker can expect to receive the following number of linkable queries from a tested node B in the same time interval. If the tested node B is closer to the initiator, the attacker expects to receive

$$q_B = q_A / P_{rnd}(A)$$

linkable queries from this node. Otherwise, the tested node is more distant from the initiator and the attacker expects to receive only

$$q'_B = q_A \cdot P_{rnd}(B)$$

linkable queries from this node.

It is important that $P_{rnd}(A)^{-1} \gg P_{rnd}(B)$ to succeed with the simplified test. Therefore, the smaller P_{rnd} is, the better can the attacker use this test to distinguish which node is closer to the initiator.

- In general P_{rnd} is very small in GNUnet, as every node has many neighbor nodes but only few of them are randomly selected. In this case the node that is closer to the initiator can be determined efficiently.
- However, if the load of a node is low, the node begins to pad the empty message queues with received queries and P_{rnd} increases. If nodes are finally broadcasting queries to all neighbors, even the exact test fails, as $P_{rnd}(A)^{-1} = P_{rnd}(B) = 1$.

Thus, the attack becomes more difficult with lower load. If the load on the network is low, the attacker has to increase the load for the attack to be successful. Note that nodes with low load are accepting zero priority queries and no credit is required to increase the load.

Number of Queries Required. To estimate how many messages the initiator can send without being identified we assume P_{rnd} to be equal for all nodes in GNUnet. Let l be the distance of the attacker to the initiator.

1. The attacker will receive linkable queries from intermediary node i with probability P^i_{rnd}, where $1 \leq i \leq l$. The first query is therefore received after the issuer has sent $m_i \geq 1/P^i_{rnd}$ queries.

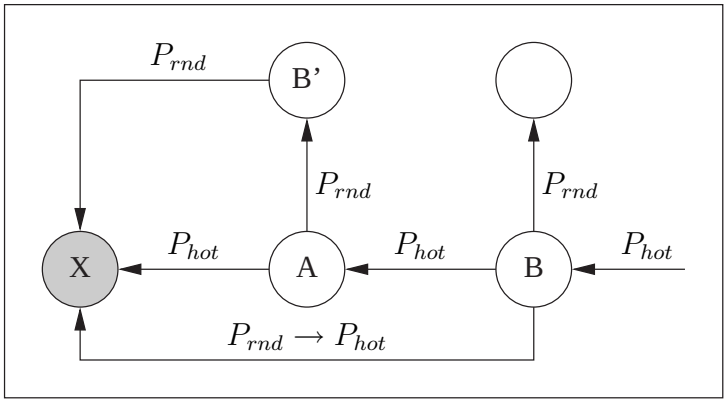

Fig. 3. Attacking Hot Paths.

2. The attacker will receive linkable queries from the neighbors of the initiator with probability P_{rnd}^2. The first query is therefore received after the issuer has sent $m_0 = 1/P_{rnd}^2$ queries.

Altogether, a lower bound on the total number of queries required to determine the initiator is

$$m \geq \frac{1}{P_{rnd}^2} + \sum_{i=1}^{l} \frac{1}{P_{rnd}^i}$$

4.3 Hot Path Routing

With hot path routing the nodes to receive a query are selected by the number of their valid responses to recent queries. Neighbors that respond more often are more likely to receive further queries. The probability that node A selects its neighbor I is $P_{hot}(A, I) = r/q$, where q is the number of queries A has recently (in a time interval) sent to I and r is the number of valid responses. Thus, if the attacker is a hot node, the shortcut attack becomes considerably easier:

1. More queries are routed to the attacker and thus fewer queries are required in total. If a node is on the hot path, then $P_{hot} > P_{rnd}$, otherwise $P_{hot} \approx P_{rnd}$.
2. Due to the random routing, the hot path still leaks queries to neighbor nodes that are not on the hot path.

Both properties help the attacker to determine the initiator node. Figure 3 shows a hot path, where the attacker is a hot node. In this case, the attacker can efficiently test, which neighbor of A is closer to the initiator.

Testing. To test which of the neighbor nodes B_i of the current preceding node A is closer to the initiator, we compare the number of queries received from those nodes. Let q_A be the number of linkable queries the attacker X receives from A. Then the attacker can expect to receive the following number of linkable queries from a tested node B in the same time interval. If the tested node B is closer to the initiator, the attacker expects to receive

$$q_B = q_A \cdot \frac{P_{rnd}(B)}{P_{hot}(A,X) \cdot P_{hot}(B,A)} \approx q_A \cdot P_{rnd}/P_{hot}^2$$

linkable queries[2] from this node. Otherwise, the tested node is more distant from the initiator and the attacker expects to receive only

$$q'_B = q_A \cdot \frac{P_{rnd}(A) \cdot P_{rnd}(B)}{P_{hot}(A,X)} \approx q_A \cdot P_{rnd}^2/P_{hot}$$

linkable queries from this node.

It is important that $P_{rnd}/P_{hot}^2 \gg P_{rnd}^2/P_{hot}$ to succeed with the simplified test. As $P_{hot} \geq P_{rnd}$ and P_{rnd} is small, the node that is closer to the initiator can be determined efficiently.

Number of Queries Required. To estimate how many messages the initiator can send without being identified we assume P_{rnd} and P_{hot} to be equal for all nodes in GNUnet. Let l be the length of the hot path.

1. The attacker will receive linkable queries from the preceding node with probability P_{hot}^l. The first query is therefore received after the issuer has sent $m_l \geq 1/P_{hot}^l$ queries.
2. The attacker will receive linkable queries from intermediary node i on the hot path with probability $P_{hot}^{i-1}P_{rnd}$, where $1 \leq i \leq l-1$. The first query is therefore received after the issuer has sent $m_i \geq 1/P_{hot}^{i-1}P_{rnd}$ queries.
3. The attacker will receive linkable queries from the neighbors of the initiator with probability P_{rnd}^2. The first query is therefore received after the issuer has sent $m_0 = 1/P_{rnd}^2$ queries.

Altogether, a lower bound on the total number of queries required to determine the initiator is

$$m \geq \frac{1}{P_{hot}^l} + \frac{1}{P_{rnd}^2} + \frac{1}{P_{rnd}} \cdot \sum_{i=0}^{l-2} \frac{1}{P_{hot}^i}$$

[2] Moreover, as shown in Figure 3 the local receiving probability will increase over time to P_{hot}, as B learns that the attacker is a hot node, while A may be unable to respond to B's queries without the help of the attacker.

4.4 Discussion

In the previous sections we have presented lower bounds for the number of queries required to determine the initiator. Those lower bounds are only a rough estimation under several simplifications (e.g. no TTLs, no loops, etc.). Nevertheless, we now try to discuss the efficiency of the shortcut attack in practice. Therefore, we assume "realistic" parameters: the initial distance of the attacker to the initiator is $l = 6$ and every node in the network uses $P_{hot} = 0.9$ and $P_{rnd} = 0.1$.

If random routing is only used, GNUnet seems to be secure, as the attacker is relatively far away from the initiator and receives every query sent by the initiator only with probability 10^{-6}. Thus, it is very unlikely that the attacker even notices a download. The question is however whether the shortest path from the initiator to the attacker is indeed relatively long.

Let n be the total number of nodes, c be the number of attackers, and k be the number of neighbors. Assuming that each node chooses its neighbors randomly, then the probability that a node connects to i attackers is

$$p_i = \frac{\binom{c}{i}\binom{n-c}{k-i}}{\binom{n}{k}}$$

For $k \ll n$ the probability that a node is not connected to an attacker can be simplified to $p_0 = (1 - c/n)^k$. Thus, unless the fraction c/n is negligible, the number of neighbors has a great impact on the length of the shortest path to an attacker. Interestingly, having fewer neighbors not only increases the length of the path, but also increases P_{rnd}, which has two effects: On one side, it becoms more difficult for the attacker to test neighbor nodes, but on the other side more queries are routed to the attacker, which helps the attacker unless $P_{rnd} = 1$.

Therefore, it seems to be important that each node does not choose its neighbors randomly, but uses a trust-metric, e.g. [LA98], to minimize the risk connecting to an attacker. Furthermore, such a trust-metric makes shortcut attacks much more unlikely, as a node can reject connection requests from untrusted nodes.

While the security of random routing remains controversial, our shortcut attack is very successful, if the attacker is able to exploit a hot path. With the parameters of the example above only 82 multiqueries are required to determine the initiator and thus downloading a file of approximately 2 MB size is potentially dangerous.

Even if an implementation of the attack actually requires a multiple of the estimated number of queries, it shows that hot path routing is a potential weakness. The best strategy for an attacker is to provide GNUnet with several high bandwith nodes. As hot paths always leak some queries to the neighbor nodes, the attacker can very efficiently test which nodes are on the hot path, until the initiator is discovered.

To summarize, hot path routing not only improves GNUnet's performance, but it also increases the efficiency of our attack. As GNUnet is very inefficient without hot path routing, we suggest to make hot paths more secure.

- To prevent the improved shortcut attack, hot paths must not leak linkable queries to nodes not on the hot path. Therefore, hot paths have to be much more static, and all nodes on the hot path must have knowledge which queries are linkable.
- But even with very static paths, the basic shortcut attack can be used to discover the initiator. While the basic shortcut attack requires credit, it should be noted that the attacker earns this credit by responding to queries.

Therefore, hot paths should be very static, i.e. shortcuts should not be allowed on hot paths.

5 Censoring GNUnet

Assuming that GNUnet turns out to be practical and is in wide use, we expect that there will be attempts to censor GNUnet. We present two methods to censor GNUnet: rubber-hose cryptanalysis and content filtering.

5.1 Rubber-Hose Cryptanalysis

One possibility for censorship is to apply "Rubber-Hose Cryptanalysis" by forcing the owner of a node to remove certain content. Rubber-hose cryptanalysis can only be used to censor infrequently requested content that is not widely distributed, as it is necessary to discover which nodes store the content to be censored. On the other side GNUnet also has built-in censoring capabilities as infrequently requested content is automatically removed from the node's caches and rubber-hose cryptanalysis becomes more or less unnecessary. To prevent infrequently requested content from vanishing, GNUnet provides an alternative method to add content.

- The default method to add content is inserting: All blocks are generated and stored in the local cache in encrypted form.
- Alternatively, indexing can be used to add content: Only the RBlock is generated and all other blocks are produced on the fly upon request.

While indexing saves space and prevents infrequently requested content from vanishing from GNUnet, this approach has a major drawback: A reverse shortcut attack can be used to discover nodes providing such infrequently requested indexed content.

Reverse Shortcut Attack. The shortcut attack can similarly be used to discover the responding node by exchanging the roles of A and X (see Figure 2). The attacker can successively get closer to the responding node by eliminating intermediary nodes one after the other.

Compared to the basic shortcut attack, where it is quite difficult for an attacker to force a node to reveal the preceding node, the reverse shortcut attack is much simpler.

1. The attacker floods B with high priority queries. Due to the promised credit B will try to respond to those queries. The attacker may hide his attack by using several malicious nodes to flood the attacked node.
2. The load on the attacked node B increases and the node decides not to indirect but to forward (or to drop) some traffic. When forwarding queries B does not overwrite A's return address, so that the X will send responses directly to the attacker.
3. After the attacker has received a valid response from X, the attacker will send all further queries to X.

Thus, the path is very static, which makes this attack practical. While the attack still requires credit to flood the node, it should be noted that the attacker is downloading the content, therefore, the attacker can postpone queries until he has acquired enough credit at the attacked node to proceed with the attack.

5.2 Content Filtering with Licenses

Rubber-hose cryptanalysis is neither suited for large scale censorship nor to censor frequently requested content that is stored in many locations. Therefore, we present a content filtering mechanism that can be legally enforced to make sharing illegal content nearly impossible.

Our filter mechanism makes use of the fact that I- and DBlocks are stored under the hash value of their encrypted content. Thus, indexing illegal content is possible and censorship can be applied very efficiently by allowing nodes to only deliver root IBlocks together with a valid license (in principle it is sufficient to filter root IBlocks, however, this requires that root IBlocks are distinguishable from other blocks). A censoring authority issues licenses by signing the query-hash $H(E_{H(I)}(I))$ of the root IBlock. There are two types of licenses, positive and negative licenses. A positive license allows, a negative license prohibits delivering the corresponding content.

Positive License: A positive license certifies that $H(E_{H(I)}(I))$ is currently not on the index. Positive licenses are always time restricted. Thus, to issue a positive license, the censoring authority need not check the actual content.

Negative License: A negative license certifies that $H(E_{H(I)}(I))$ is on the index. Negative licenses are not time restricted.

A node that wants to respond to a query, but does not possess a positive license for the IBlock (e.g. the content is new or the positive license is not valid anymore), first tries to retrieve a license from GNUnet. If no license is available, it

directly requests a license from the censoring authority. Thus, for frequently requested legal content the permission to deliver this content is widely distributed in GNUnet and can be retrieved efficiently.

Censoring. Searchability and censorship-resistance exclude each other to a certain extent. If content is easy to find for users, it is similarly easy for others to find illegal content:

1. Content owners can search for their copyrighted content. After finding such content, the corresponding root IBlock is sent to the censoring authority together with a proof of ownership.
2. The censorship authority itself searches for illegal content and issues negative licenses for all found IBlocks.

To check whether a node applies content filtering, the censoring authority only has to search for indexed content. The owner of a node that returns such content (without a valid positive license) may be prosecuted for e.g. copyright infringement. As every owner of a node can be prosecuted, GNUnet may discourage users to share illegal content even more than any non-anonymous approach.

Multiple Censoring Authorities. Instead of using only a single censoring authority, it is also possible to use multiple censoring authorities, depending on the jurisdiction of the node indirecting the content.

5.3 Discussion

Constructing an anonymous, censorship-resistant network is very difficult. Besides Free Haven [DFM01] the only approach achieving a similar goal is sketched in [Ser02]. There, it is also discussed why most censorship-resistant networks (e.g. Publius [MWC00], Freenet [CSWH01], and Tangler [WM01]) fail to resist rubber-hose cryptanalysis. Unfortunately, due to indexing GNUnet is even more affected by rubber-hose cryptanalysis than those approaches. If indexing is used to add infrequently requested content to GNUnet, the node risks being identified with the reverse shortcut attack. Therefore, indexing should not be used for such content.

While the reverse shortcut attack can only be used to censor infrequently requested indexed content, our filter mechanism is well suited to censor popular content. Content filtering is a very strong, controversial attack, as it requires the nodes to apply the filter. Content filtering is also often compared to shutting down GNUnet. However, we think that is important to consider such attacks:

- History has shown that such attacks are used in practice, e.g. Napster was obliged to filter content before it was shut down.
- Content filtering can be applied very efficiently with only minor changes in GNUnet, resulting in an anonymous but “clean” network.

As every block in GNUnet is identified by an unique identifier, content filtering is always possible. Removing unique identifiers, also removes the ability to automatically replicate content. Therefore, preventing content filtering seems to be impossible.

6 Conclusion

We have pointed out some potential weaknesses of GNUnet. A problem that is inherent to the design of GNUnet is the decision to split larger files into many small linkable pieces. To download a file all those linkable pieces have to be retrieved separately. An attacker who can link those related queries is able to determine the initiator of the download. Depending on the distance of the attacker and on the load of the network a huge number of queries may be necessary to determine the initiator. However, to be more efficient GNUnet tries to automatically optimize routing with hot paths. We have shown that the attacker can benefit from hot paths, which makes our attack much more efficient.

Another problem that we have discussed is censorship-resistance. We have shown that GNUnet is vulnerable to rubber-hose cryptanalysis, but we additionally have presented an efficient content filter for GNUnet. If content filtering is legally enforced, censoring GNUnet is possible at a very large scale.

Acknowledgements

Many thanks to Christian Grothoff from GNUnet for discussing the attacks and to Andrei Serjantov for commenting on earlier versions of this paper.

References

[And96] Ross J. Anderson. The eternity service. In *Proceedings of Pragocrypt '96*, 1996.

[BG03] Krista Bennett and Christian Grothoff. GAP – practical anonymous networking. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2003*. Springer-Verlag, 2003.

[BGHP02] Krista Bennett, Christian Grothoff, Tzvetan Horozov, and Ioana Patrascu. Efficient sharing of encrypted data. In *Information Security and Privacy – 7th Autralasian Conference (ACISP 2002)*, Lecture Notes in Computer Science 2384, pages 107–120. Springer-Verlag, 2002.

[BPS01] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 30–45. Springer-Verlag, 2001.

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

[CMH+02] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, pages 40–49, 2002.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 46–66. Springer-Verlag, 2001.

[DFM01] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 67–95. Springer-Verlag, 2001.

[FP00] Justin Frankel and Tom Pepper. Gnutella v. 0.56. Nullsoft, 2000.

[GPB+02] Christian Grothoff, Ioana Patrascu, Krista Bennett, Tiberiu Stef, and Tzvetan Horozov. GNET. Whitepaper version 0.5.2, 2002.

[Gro03] Christian Grothoff. An excess-based economic model for resource allocation in peer-to-peer networks. *Wirtschaftsinformatik*, (3), 2003.

[LA98] Raph Levien and Alex Aiken. Attack-resistant trust metrics for public key certification. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[MWC00] Aviel D. Rubin Marc Waldman and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, 2000.

[PK01] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity – a proposal for terminology. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 1–9. Springer-Verlag, 2001.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[Ser02] Andrei Serjantov. Anonymizing censorship resistant systems. In *Peer-to-Peer Systems, First International Workshop – IPTPS 2002*, Lecture Notes in Computer Science 2429, pages 111–120. Springer-Verlag, 2002.

[Shm02] Vitaly Shmatikov. Probabilistic analysis of anonymity. In *15th IEEE Computer Security Foundations Workshop*, pages 119–128. IEEE Computer Society Press, 2002.

[WALS02] Matthew Wright, Micah Adler, Brian N. Levine, and Clay Shields. An analysis of the degradation of anonymous protocols. In *Network and Distributed System Security Symposium – NDSS 2002*. Internet Society, 2002.

[WM01] Marc Waldman and David David Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *ACM Conference on Computer and Communications Security*, pages 126–135. ACM Press, 2001.