Peeriod: An Anonymous Approach for Decentralized Overlay Networks

Jonathan Pirnay[1], Jörn Röder[2]

Department New Media, School of Art and Design Kassel, Germany.

[1] mail [at] johnnycrab.com, [2] kontakt [at] joernroeder.de

Abstract. Peer-to-peer networks have become increasingly popular for transferring files over the Internet. In many popular cases, however, transmitted data is neither encrypted nor is the user's anonymity protected. Onion Routing has become today's typical solution for low-latency anonymous communication, but requires a public-private-key schema to authenticate relay nodes. In the absence of trusted third parties within a fully decentralized peer-to-peer network though, key management and distribution are problems very hard to solve.

We propose an approach for Onion Routing on a Distributed Hash Table topology without a public key infrastructure by using an additive sharing scheme. This approach prevents network participants from being able to spoof paths, but does not protect against an active man-in-the-middle who can observe and modify outgoing and incoming traffic. Users are able to share files with mutual sender-receiver anonymity. Furthermore, we show an approach for an efficient anonymous flooding-based search mechanism.

1 Introduction

The strongly centralized Internet of today has made peer-to-peer systems more and more important. Applications range from file sharing over digital currencies to decentralized video streaming solutions. But many popular systems do neither encrypt transmitted data nor do they securely provide anonymity for the user. This is a problematic situation in a society with a growing concern for privacy and censorship.

However, anonymous routing in overlay networks as a means for untraceable communication has been studied extensively over the past decades (for example [1], [2], [3], [4]). An important example is formed by *Dining cryptographers networks* (DC-Nets), originally introduced by David Chaum [2], for anonymously publicating messages. DC-Nets base their anonymity on secure multi-party sum computation and thus provide provable complete anonymity in the absence of trusted participants. The original system, however, was susceptible to transmission collision attacks. Later, Waidner [5] proposed a system of traps and commitments for DC-Nets in order to improve robustness and adversary-detection, but this is at the cost of multiple broadcast rounds and a significant communication overhead. At best, variations of DC-Nets require $\mathrm{O}(n^2)$ messages per anonymous message in a network of n participants. This high message complexity renders them poorly scalable and infeasible for many practical applications.

Mix-Nets [1], also introduced by David Chaum, are based on the notion that a trusted node, a 'Mix', batches, shuffles and routes messages from other nodes, thus complicating traffic analysis. Chaining Mixes together forms the basis for *Onion Routing* (OR) [4], a form of source routing where through layered encryption a node within an OR circuit can only tell its successor and predecessor: it has no knowledge of the sender, the receiver, the path or the content of the message. OR provides provable anonymity in the face of a passive adversary, but Wright et al. have shown that an active dishonest participant can theoretically degrade the efficiency of Mix-Net based systems through selective non-participation. [6]

Nevertheless, Onion Routing has become a common paradigm when it comes to anonymous routing in overlay networks, such as TOR [7] or Tarzan [8], as it is practical for near real-time communication. In order to provide sender and receiver anonymity, potential relay nodes in an OR circuit need to authenticate themselves so that the first node cannot spoof the rest of the path. This is ensured by public-private keys verified by a trusted certificate authority, i.e. it requires a public key infrastructure (PKI). The PKI also protects from an active man-in-the-middle who is potentially able to corrupt a Diffie-Hellman key exchange. Katti et al. have lined out the problematics of a PKI in a peer-to-peer network, especially its inapplicability in truly anonymous peer-to-peer systems without trusted third parties. [9] Their proposed method is based on *information slicing*: A message is divided into a number of blocks. Each block is multiplied with a random invertible matrix. The resulting blocks and the rows of the matrix are delivered to the intended node along disjoint paths which meet only at the intended receiver. Our proposed method uses a variation of *information slicing*, but utilizes an additive sharing scheme and - as opposed to [9] - does not make the assumption of a node being able to send from multiple IP addresses. Not being able to send from different addresses, however, makes our approach susceptible to an active man-in-the-middle attack where said attacker is able to observe, modify and reroute outgoing and incoming traffic.

We want to show an approach to a fully decentralized anonymous peer-to-peer network based on the topology of the Kademlia Distributed Hash Table (DHT) [10], without using a PKI, and a focus on file sharing. Each node maintains a number of Onion Routing circuits which it uses to store information about data locations, to retrieve them and to exchange data. At last a method is shown for the implementation of a flooding-based anonymous search with little message redundancy.

The rest of the paper is organized as follows. Section 2 outlines our goals and the model assumption. Section 3 takes a short glance at the topology and its extension of the Kademlia protocol. Section 4 presents the construction of the OR circuits in detail. Section 5 describes how a node uses its OR circuits to store and retrieve file locations and obviously how the file transfer is executed. Broadcast messaging is usually not used by applications built on peer-to-peer communication with many participants, as it generates large network traffic. However, query flooding does have benefits, so we show a flooding-based search algorithm which can be used to retrieve files in section 6. Section 7 concludes the paper and points out future work.

2 Goal and Model Assumption

We aim towards a fully decentralized and anonymous overlay network for file sharing. We focus on file sharing as it generally strives for anonymity but is not sensitive to low probable information leakage or unsuccessful file transfers.

We assume a computationally bound adversary unable to break cryptographic algorithms in polynomial time. We assume an adversary can adaptively operate/compromise a fraction of nodes himself. An adversary may be able to observe the links of a minority of nodes. However, as stated in [7], like all practical low-latency anonymizing systems, protection against a global eavesdropper being able to monitor all links in the network is not provided.

Our approach prevents network participants from corrupting the anonymity of other nodes, but does not protect against an active man-in-the-middle situation in which an attacker is able to modify the complete incoming and outgoing traffic of a node. Thus, this situation is excluded from our model assumption.

We extend our model by assuming an adversary may be able to send messages from spoofed IP addresses belonging to other nodes in the network, but is unable to receive messages on the same address. That is in our case, that the adversary can send UDP datagrams with arbitrary spoofed IPs to other nodes of his choice, however can not perform a TCP 3-way-handshake let alone carry on a TCP conversation with a spoofed IP address, as this would require predicting initial sequence numbers.

We assume a node in the network has an open port for both TCP and UDP. For Diffie-Hellman key exchanges, arithmetic operations will be performed over a primitive residue class modulo p, so at last we assume all participants in the network have agreed upon p, a large prime, and g, a primitive root mod p. E.g. these numbers can be provided by the client software.

3 Topology

One must provide a decentralized distributed system in which a node can efficiently retrieve a value associated with a given key. We base our network on the DHT system of Kademlia, as proposed by Maymounkov and Mazières in 2002. [10] The original paper describes the protocol in detail, nevertheless we provide a brief overview of the key concepts and their advantages:

Participating computers in the network have a node ID in a 160-bit key space which they share with keys (e.g. SHA-1 hash of some data) for ⟨key, value⟩ pairs "stored on nodes with IDs 'close' to the key for some notion of closeness". [10] Every node in the network maintains edges to the k-nearest nodes respecting Kademlia's XOR metric for distance.

The distance between two identifiers a and b is defined as:

$$distance(a,\ b) := a \oplus b$$

For each distance interval $d \in [2^i,\ 2^{i+1} - 1]$ with $i \in \{0,\ \ldots\ ,\ 159\}$, every peer maintains k neighbors who are chosen using a *Least-recently-seen* concept. k

neighbors linked to a specific distance interval are called a 'k-bucket'. At a closer look, Kademlia uses exactly the routing schema proposed by Plaxton et al. in 1997. [11] Due to the symmetry of the XOR metric, nodes can gain useful routing information through received queries.

Thus, the degree of Kademlia is a maximum of $\sum_{i=0}^{159} \min(k,\ 2^i)$. The commonly used $k = 20$ returns a maximum degree of 3131. Compared to a system like Chord [12], where a peer merely maintains at most 128 neighbors, the high degree gives a node ample possibilites of choosing relay nodes for potential Onion Routing circuits.

When searching for a specific ID, theoretically one bit is adjusted with every routing hop, succeeding after $\mathrm{O}\big(\log n\big)$ hops. Consequently a Kademlia network has an expected diameter of $\mathrm{O}\big(\log n\big)$.

Contact information of a Kademlia node is constituted by ⟨IP address, UDP port, Node ID⟩ triples. The original paper describes Kademlia's lookup algorithm as recursive, but it truly is an iterative one and thus can cope with UDP's unreliability. Our design on top of Kademlia, however, is based on message forwarding and accordingly reliable transport, so we extend a node's contact information to a ⟨IP address, UDP port, TCP port, Node ID⟩ quadruple.

An implementation can, of course, respect the fact that some nodes may be able to send messages from multiple IP addresses or receive them on multiple ports, but for the sake of simplicity we assume a node is associated to one IP, one UDP port and one TCP port.

At last we want to stress that our design does not provide anonymity for the maintenance of the DHT, i.e. messages of Kademlia RPCs, but rather tries to build anonymity on top of it.

4 Constructing Onion Routing Circuits

The general goal of a node in the network is to always maintain a fixed number α of disjoint OR circuits on top of this topology. The relay nodes of one OR circuit are determined by randomly choosing β ⟨IP address, TCP port⟩ pairs from the node's routing table. In order to guarantee a notion of randomness, nodes should not establish OR circuits as long as the overall count of contact information is less than some self-imposed limit.

These circuits must be regularly changed. As the only information an initiating node has is the IP/Port combination of another requested node and no knowledge of its (dis)honesty, we demand that OR circuits are torn down on any protocol non-compliance or unresponsiveness.

The main difficulty in this design is the absence of trusted third parties and thus impracticality of public-private key schemes. In traditional OR as in [7], a node A initiating a circuit needs to negotiate an ephemeral symmetric key with a potential relay node B. A does so by choosing a circuit ID cid_{AB} and the first half of a Diffie-Hellman handshake g^a. This information is encrypted by a public key which is bound to the identity of B. B answers with his half of the handshake g^b and a hash of the negotiated symmetric secret S_{AB} derived from g^{ab}. When further extending the circuit with a node C, A encrypts g^{a_2} with the public key of C and with S_{AB}. The resulting message is relayed via B (who peels off the first layer of

encryption and adds a circuit ID cid_{BC}) to C, who himself sends g^c and a hash of S_{AC} (derived from $g^{a_2 c}$) back to A via B.

A public-private-key scheme is needed here in order to ensure the authenticity of C so that B cannot impersonate C and spoof all remaining paths. But, as stated earlier, managing a PKI in fully decentralized peer-to-peer systems is difficult. Our proposed approach is based on an additive sharing scheme.

To recapitulate, A needs to send $g^a \circ cid_{AB}$, where $\circ$ denotes the concatenation, to B *in confidence of truly reaching* B.

In general, additive sharing of a secret s over a finite field $\mathbb{F}$ with $s \in \mathbb{F}$ consists of h random shares $s_1,\ s_2,\ \ldots,\ s_h$ distributed among h players such that

$$\sum_{i=1}^{h} s_i = s$$

Such a scheme naturally leads to a threshold $\tau = h - 1$ which means that *all* players must reveal their shares to reconstruct the secret s.

$Enc_S(t)$ denotes the encryption of a message t with a symmetric key S.

Let $\boldsymbol{m}$ be the message vector of $m := g^a$ which A wants to send to B, with $\boldsymbol{m} \in \mathbb{F}_{2^8}^n$ where n equals the number of bytes of m.

A generates h random vectors $\boldsymbol{r_1},\ \boldsymbol{r_2},\ \ldots,\ \boldsymbol{r_h} \in \mathbb{F}_{2^8}^n$ and calculates a ciphertext $\boldsymbol{c_{AB}}$:

$$\boldsymbol{c_{AB}} := \boldsymbol{m} - \sum_{i=1}^{h} \boldsymbol{r_i} \in \mathbb{F}_{2^8}^n$$

Now A chooses from his routing table a set of h random ⟨IP address, TCP port⟩ pairs disjoint with A's choice of relay nodes. A sends to each of these random messengers one share $\in \{\boldsymbol{r_1},\ \boldsymbol{r_2},\ \ldots,\ \boldsymbol{r_h}\}$ and the address of B. The nodes obtaining such a message send their received share to B. A sends $\boldsymbol{c_{AB}} \circ cid_{AB}$ to B himself. As soon as B has received all shares, he can compute $\boldsymbol{m}$, because

$$\boldsymbol{m} = \boldsymbol{c_{AB}} + \sum_{i=1}^{h} \boldsymbol{r_i}$$

B can now perform his part of the handshake g^b and send it directly back to A (because he received cid_{AB} from A) with the hash digest of g^{ab}.

If A receives this message over the same connection he established with B and the hash digest equals the one A computes of his exponentiation, he defines B as a valid relay node of the OR circuit.

If A receives an invalid or no message from B, he randomly chooses substitutes for B and messenger nodes.

If A wants to extend the circuit further with C, he repeats the process, but sends $Enc_{S_{AB}}(\boldsymbol{c_{AC}})$ to node B, who himself chooses cid_{BC} and relays $\boldsymbol{c_{AC}} \circ cid_{BC}$ to C. The rest follows analogously. Figs. 1 and 2 depict the flow of the two stages.

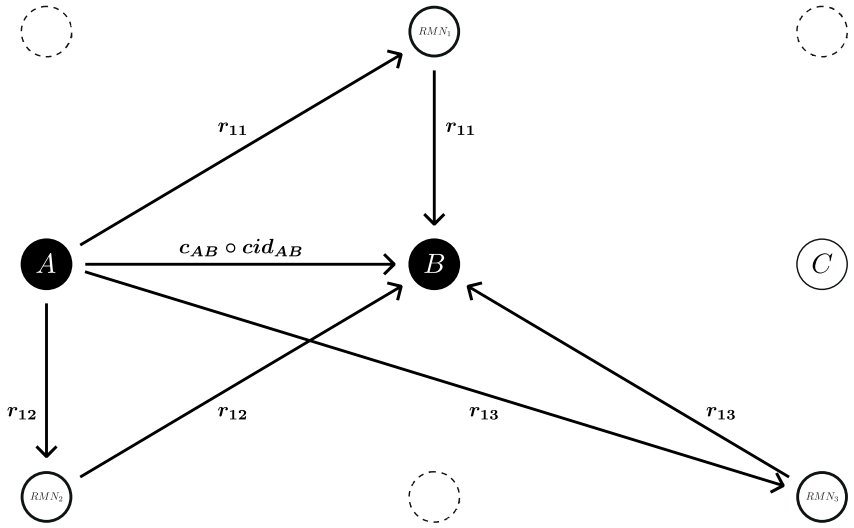

Fig. 1. A creates an OR circuit with B as the first relay node with $h = 3$. When B has received all shares, he can successfully calculate $\boldsymbol{m_{AB} = c_{AB} + r_{11} + r_{12} + r_{13}}$ and send his response back to A (RMN denotes 'Random Messenger Node').

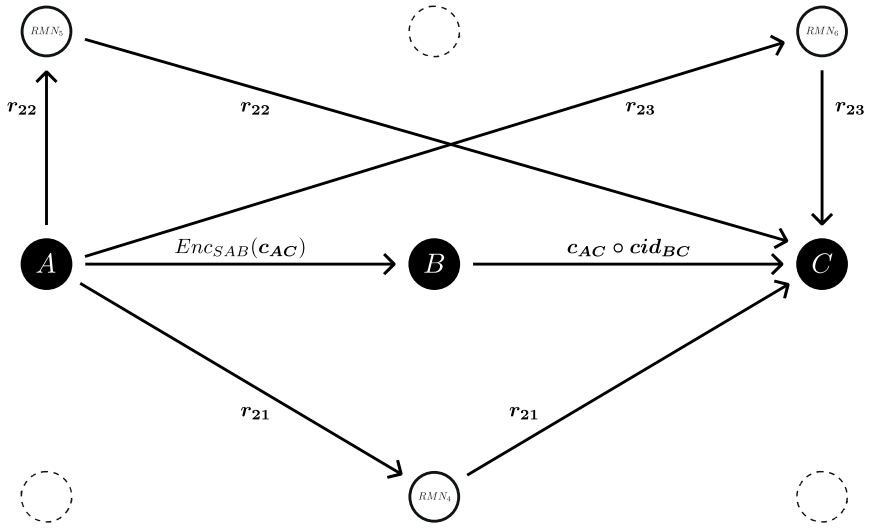

Fig. 2. A extends the circuit further to a node C. One share of the message is relayed to C via B, who can choose an arbitrary circuit identifier and append it to the share. As in traditional Onion Routing, C sends his response back to A via B.

We can assert the following properties of our approach:

1. As already stated, the main drawback of this approach is that it is susceptible to an active man-in-the-middle who is able to route all outgoing and incoming traffic of A through himself. Such an attacker can easily impersonate B, spoof the remaining OR circuit and thus render A's anonymity defective.
2. Every random messenger node knows that B is part of an OR circuit initiated by A, however cannot know the intended position of B within the circuit.
3. An involved node can jam a handshake by non-participation. In this case, as stated earlier, A substitutes all nodes with another set of random nodes.
4. A passive eavesdropper spying on all incoming and outgoing links of A may be able to reconstruct all paths of the circuit, however has no knowledge of any symmetric keys. Assuming the eavesdropper also monitors all links of the last relay node, he may be able to assign outgoing unencrypted packets to A. Assuming further that a node can be part of many OR circuits impedes such traffic analysis attacks significantly. Again, this would be even more complicated in later stages of the circuit extension, as the adversary can only view the contents of $Enc_{S_{AB}}(\boldsymbol{c_{AC}})$ if B's outgoing links are being spied on, too.
5. B being controlled by an adversary has no effect, as long as B acts compliantly to the protocol.
6. B is able to spoof the rest of the OR circuit, though, if
   (a) B is controlled by an adversary who can spy on all outgoing links of A.
   (b) B is controlled by an adversary also controlling all chosen random messenger nodes in every stage of the circuit extensions.
7. A Sybil attack aimed towards compromising the routing table of A by trying to fill it with as much hostile nodes as possible (comparable to [13]) seems difficult. The distance interval $[2^{159}, 2^{160} - 1]$ alone includes theoretically half of the nodes in the whole network.
8. The security of the scheme can be improved if A maintains existing OR circuits, thus already sharing symmetric keys, and can relay one or more random shares via these existing entry nodes. This would guarantee encrypted transmission of at least one share even in the first stage of the circuit construction.

The problem of property number two - random messenger nodes gaining knowledge about who is constructing circuits with whom - can be solved by using a *multi-hop* approach of the additive sharing scheme. Taking the example from above, if A were to initiate a circuit with B, A would again generate $\boldsymbol{r_1}$, $\boldsymbol{r_2}$, ... , $\boldsymbol{r_h}$ from g^a. Let's assume A would use an intermediary hop amount of 1. Instead of sending the shares with the address of B to random messenger nodes RMN_1, ..., RMN_h, A would *split up each share again*: A generates for each share a new message which is the concatenation of B's address and the share, padded to a fixed length, e.g. $\boldsymbol{s_1} := Padded(address_B \circ \boldsymbol{r_1})$.

Each one of the resulting messages $\boldsymbol{s_1}$, ..., $\boldsymbol{s_h}$ is then again divided into $h+1$ random shares of an additive sharing scheme. The appropriate parts are transferred to RMN_1, ..., RMN_h via $h+1$ different relay nodes each. Thus, if for example RMN_1 receives all shares of $\boldsymbol{s_1}$ from different nodes, he can successfully reconstruct $\boldsymbol{s_1}$, notices the existence of $address_B$ at the beginning of the message, removes $address_B$ and sends the remaining message to B.

Finally B gains knowledge of $\boldsymbol{r_1}$, ..., $\boldsymbol{r_h}$ and receives $\boldsymbol{c_{AB}}$ from A himself. B can now compute the secret.

In this multi-hop scheme, the shares must be padded to a fixed length in order to conceal how many intermediary hops are still left. However, using this concept greatly increases the number of messenger nodes needed until B receives the initial cleartext message. For each of the original h shares, A needs another $h+1$ messenger nodes, until all RMNs have received their share $\boldsymbol{s}$, which they can send to B. For only one intermediary hop this adds up to a total of $2h^2 + 3h$ sent messages. Using for example $h = 4$ (i.e. each message consists of 5 shares) would amount to 44 messages.

Splitting up each share *again* for $h = 4$, i.e. using a multi-hop level of 2, results in 100 shares and a total message amount of 224 until B can generate the secret. Generally a multi-hop level n leads to a total message count of $(2h+1)(h+1)^n - 1$ until B has received all shares. Obviously each additional message increases the probability of coming across a malicious node. The advantage however is that it becomes very hard for a collaborating group of malicious nodes to keep track of the fact that A is extending/initiating a circuit with B, as they would need to be present on every level of the whole scheme.

5 File Transfer Using the Onion Routing Circuits

In this section we outline how the OR circuits are used to store locations to files and how to retrieve them. We assume that a node A maintaining an OR circuit has negotiated individual identifiers fid_{A1}, fid_{A2}, ... with each of the circuit's relay nodes. This can be achieved by e.g. deriving an additional identifier from the secret exchanged through the Diffie-Hellman protocol. As opposed to *cid*, fid will be made public.

Let A maintain a variety of OR circuits and provide a file. A computes the SHA-1 digest of the file. He adds the ⟨IP address, TCP Port, fid_{Ax} ⟩ pairs of the exit nodes of his circuits to the hash and pipes an instruction to store the resulting information through one of his circuits. The exit node of the circuit locates k nodes close to the hash value and stores the same information on them using Kademlia's RPCs.

Let D be a node maintaining a variety of OR circuits who likes to retrieve the file A possesses. Assuming D has knowledge of the file hash, he sends the instruction to search for the hash value through one of his OR circuits. Again, with Kademlia's RPCs, the exit node locates the information and passes it back to D. D calculates his half of a Diffie-Hellman handshake, appends ⟨IP address, TCP Port, fid_{Dx} ⟩ pairs of his circuits' exit nodes and sends this information along with the hash of the desired file through a circuit to one of A's exit nodes, marking the message with fid_{Ax}. The message will finally reach A as each exit node of A's circuits can relate his individual fid_{Ax} value to the appropriate circuit.

The exit node pipes the message back to A, who himself computes the other half of the handshake as well as a hash of the derived key. The resulting information, marked with fid_{Dx} is now sent back to an exit node of D who pipes it back to D through the circuit he can relate fid_{Dx} to.

The rest should now be trivial: D and A share a secret which can be used to derive symmetric keys and encrypt further communication (in addition to the usual OR encryption schemes). They also have knowledge of each other's exit nodes and fids which they can append to their messages, so the exit nodes know how to deal with the received messages. D acknowledges the handshake by sending an encrypted version of the desired file's hash value back to A. The file can now be securely transmitted with mutal sender-receiver anonymity.

Of course this is just a general outline of a file transfer and open to propositions. Obviously any relay node included in the process could jam / delay the transfer by non-participation or impersonating A respectively D. In the worst theoretical case this can lead to downloading completely useless data, consuming bandwidth and computational power, noticeable only when finally comparing the hash values of the desired and the actually downloaded file. However, this is why a node maintains multiple OR circuits: e.g. in the initial stages, two handshaking nodes can alternate between their OR circuits, thus being able to earlier notice protocol non-compliance, jamming or impersonation. Moreover, maintaining multiple circuits gives the nodes ample possibilites to use a different circuit if one fails or must be torn down. Naturally these changes in the "onion topology" need to be reflected to the other side and to the information stored on the nodes close to a file's hash value.

Describing solutions to these problems goes beyond the scope of this paper though. Furthermore, issues like verifying the authenticity of a file during transmission have been extensively covered in scientific literature.

Still, assuming A and D have successfully set up valid OR circuits, any misbehavior should not compromise their anonymity.

6 Flooding-Based Search

Scalable peer-to-peer networks infrequently utilize broadcasting of messages. In fact structured overlay networks were designed to perform queries with a logarithmic number of hops and to reduce traffic generated by lookup requests, which were based on flooding in earlier systems like Gnutella. Although query flooding obviously scales poorly and demands defense efforts against additional denial of service attacks [14], it has appealing benefits, as it extends the exact-match search of DHT-based systems to the full world of search methodologies, e.g. enabling users to write their own algorithms to match against an incoming query. Furthermore, it avoids outsourcing the indexing of in-network data. Regarding file sharing, for example the guilty verdict of The Pirate Bay trial against four individuals maintaining a BitTorrent tracker has shown the problematics of unknowledgeably indexing (jurisdiction-specific) copyright infringing or generally illegal data.

Our query flooding algorithm is based on the work by Czirkos et al. who proposed a cost-efficient broadcast by taking advantage of the fact that Kademlia nodes can be structured into binary trees. [15] A node intending to broadcast a message sends it to a freely chosen node in each of his subtrees, i.e. his k-buckets. Assuming for simplicity that the bit-length of node identifiers is 5, and there is a

node with identifier 01101 initiating a broacast. In each of his non-empty buckets he sends the message to one node, e.g. 10110, **0**0111, **01**010, **011**11 (the shared prefix is displayed in bold font for clarity). The nodes receiving the message are responsible for propagating it in their own subtrees, and so on. In our example: 1****, 00***, 010**, 0111*. Naturally a broadcast using this algorithm will be finished in logarithmic time, reaches all nodes and generates zero redundancy.

Although in theory one node per subtree is sufficient, for an environment suffering from constant packet loss, non-participation and node churn, the authors recommend sending the broadcast message to c random nodes from each bucket, where c is a predefined system-wide constant. In this case, a search query must be supplied with a unique identifier to prevent a message from infinitely traversing the network. For one fifth of the packets lost, a broadcasting reliability of 90% is evaluated for $c = 2$, 97% for $c = 3$.

Combining the broadcast algorithm with the results from section 5 is now almost self-explanatory. A node issuing a search generates a practically unique query identifier *qid* which he adds to the query message and sends to his circuits' exit nodes. The node also appends the exit nodes' address information and fids to the query message which is then flooded through the network.

A node being able to respond adds *qid*, exit node addresses and his fids to his file suggestion and sends it back (of course via one of his OR circuits) to the requesting node, who can now pick out the hash digest of the desired file. The remaining steps follow analogously to section 5.

Moreover, the security of this scheme can be improved by a requesting node generating a public-private key pair of an asymmetric encryption scheme for each query, demanding to encrypt responses to the query and their hashes with the public key, thus impeding malicious efforts of piggybacking invalid exit node information on perfectly sane file suggestions.

7 Conclusion and Future Work

We have shown how the combination of different key concepts can be used to create a fully decentralized, distributed peer-to-peer file sharing network which makes mass oberservation hard. We have outlined how creating Onion Routing circuits with the help of an additive sharing scheme can be seen as an alternative to relying on a public key infrastructure in a peer-to-peer network situation which strives for privacy, but does not rely on zero information leakage.

Countless open roads suggest themselves. A true evaluation of the system can only happen through real-world software, thus we are implementing the concepts in an easy-to-use client application. In such an implementation our proposed techniques obviously need to be refined, e.g. schemes for a stable and adaptive download, expiration of queries and storage of data locations and their required updates when circuits are altered or nodes change their IP addresses. Routing can get bandwidth-optimized. Exit policies for relay nodes can be added.

The list of possibilities is long and usability and public perception are security parameters as well.

Nevertheless we believe the implementation and deployment of an easy-to-use client software leveraging the proposed concepts is an important contribution to large scale anonymous communication.

References

[1] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM 24(2)*, pages 84-88, 1981.

[2] David Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology 1(1)*, pages 65-75, 1988.

[3] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security 1/1*, pages 66-92, 1998.

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

[5] Michael Waidner. Unconditional sender and recipient untraceability in spite of active attacks. *Advances in Cryptology: EUROCRYP'89*, pages 302-319, 1989.

[6] M. Wright, M. Adler, B. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. *Proceedings of ISOC Symposium on Network and Distributed System Security*, 2002.

[7] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. *Proceedings of 13th USENIX Security Symposium*, 2004.

[8] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. *Proceedings of ACM CCS*, 2002.

[9] S. Katti, D. Katabi, K. Puchala. Slicing the onion: Anonymous routing without PKI. *MIT CSAIL Technical report 1000*, 2005.

[10] P. Maymounkov, D. Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *Proceedings of IPTPS02*, 2002.

[11] C.G. Plaxton, R. Rajaraman, A. Richa: Accessing nearby copies of replicated objects in a distributed environment. *9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97)*, pages 311-320, 1997.

[12] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proceedings of the ACM SIGCOMM '01 Conference*, 2001.

[13] John R. Douceur. The Sybil Attack. *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[14] N. Daswani, H. Garcia-Molina. Query-Flood DoS Attacks in Gnutella. *Proceedings of the 9th ACM conference on Computer and communications security (CCS '02)*, 2002.

[15] Z. Czirkos, G. Bognár, G. Hosszú. Packet Loss and Overlay Size Aware Broadcast in the Kademlia P2P System. *ACEEE International Journal on Communication (IJComm)*, 2013.