Proyecto de investigación básica o aplicada PEC3 — Tercera Prueba de evaluación continua

Apellidos: *FERNANDEZ FRANCO*
Nombre: *LETICIA*

- Para dudas y aclaraciones sobre el enunciado, debéis dirigiros al consultor responsable de vuestra aula.
- Hay que entregar la solución en un fichero OpenDocument o PDF utilizando una de las plantillas entregadas conjuntamente con este enunciado. Adjuntad el fichero a un mensaje dirigido al **espacio de evaluación del aula virtual**.
- El fichero debe tener la extensión .odt (OpenDocument) o .pdf (PDF) según el formato en que hagáis la entrega.
- La fecha límite de entrega es el **19 de junio** (a las 24 horas).

Respuestas

A survey and comparison of anonymous communication systems: anonymity and security.

Author: Leticia Fernández Franco
Tutors: Joan Manuel Marquès Puig and Helena Rifà Pous
Universitat Oberta de Catalunya (UOC)
Date: June 2012

ABSTRACT

Anonymous systems have received considerable attention since their starts. First were the low-latency anonymous communication systems which provide a decent degree of anonymity, most of them robust against attacks, but they have a scalability issue, they are not capable of keeping the same strength and performance once the network grows considerably. This is the reason why distributed systems have become so popular in the last ten years, because they are able to provide scalability. They appear to solve the scalability problem but unfortunately, distributed systems have shown by many researchers to be difficult to protect against security attacks, endangering the communication anonymity. In this survey an overview of the notorious communication systems has been presented and their security and anonymity studies. These systems have been categorized in four categories: low-latency anonymous communication, systems for unobservability, censorship-resistant systems and peer-to-peer communications.

1. INTRODUCTION

In recent years, investigators have been searching for the perfect secure communication system capable of providing the user with anonymity, that allows them to exercise their right to freedom of speech, and being strong enough to resist attacks from adversaries. In order to find it, different schemes have been created, analysed and improved. The most important ones have been studied and will be explained in this document.

Some of the robust anonymous systems like Tor have the problem of scalability due to their client-server architecture. That is the reason why distributed systems have been studied thoroughly in the last years. They are more scalable but, on the other hand, they are more likely to suffer from attacks, especially in the lookup process due to its importance to find nodes. Because of these issues, several investigations have focused in finding solutions to each one of the known attacks, some with more fortune than others.

The rest of the paper is organized as follows: In Section 2, the terminology in anonymous systems will be briefly described. In section 3, the most notorious attacks against anonymous communications will be defined and categorized. Section 4 is the main category, where the different anonymous communications have been analysed and classified. This category separates the schemes in four subsections which are: low-latency anonymous system with client-server architecture, systems for unobservability, censorship-resistant systems and peer-to-peer anonymous communications. The P2P schemes will be subcategorize in structured peer-to-peer topologies, distributed scalable lookups, secure lookups and peer-to-peer anonymity systems. The aspects that will be studied are: scalability, anonymity, security and censorship-resistant. Also, a security analysis of the literature will be exposed. Finally in Section 5, a comparison of the systems will be presented.

2. TERMINOLOGY IN ANONYMOUS SYSTEMS

This section explains some preliminary concepts of anonymity that will be used in the following sections to measure the degree of anonymity of the systems and schemes.

Anonymity and Pseudonymity

Anonymity must allow a subject to use a service or application without revealing its identity. Edman and Yener [24] state that the goal of an anonymous system is to provide unlinkability between received messages and their senders, and sent messages and their recipients so that when adversaries are observing the network, they can solely see the senders and recipients but cannot see who is communicating with whom.

Mutual anonymity is achieved when both subjects who exchange messages in an anonymous system are capable or remain anonymous.

Pseudonymity is the process where pseudonyms are used as a subject identifier. Its main goal would be to allow users to run services or access resources without having to reveal their own identity.

Unobservability and Unlinkability

While *unobservability* is the property whereby a subject can use a resource without an adversary having the opportunity to determine what it is being used; *unlinkability* hides the relationship between the sender and the receiver in a communication, so that the attacker is not capable of relating one with the other or with the information transmitted.

Censorship Resistance

Censorship resistance networks are those whose task it is to prevent third parties from denying the users access to a particular resource or file [75]. This is an important aspect due to anonymous communication systems being able to provide the groundwork for censorship resistance, for people who live in countries whose oppressive regimes control their citizens' movements on the Internet.

3. ATTACKS ON P2P NETWORKS

This section introduces a brief summary of the most common attacks in peer-to-peer network. A brief summary of them is needed in order to know the type of adversaries that these communications have to be prepared to deal with and their behaviour.

3.1 Denial of service

The Denial of Service or DoS attack, previously studied in [28][76][41], occurs when malicious nodes send excessive amounts of requests or duplicate packets intended for their peers to exhaust the resources of a target host. This host therefore will not be able to provide any service. Unfortunately it is a common attack but very difficult to prevent in P2P systems due to the large number of anonymous peers that form the system. Because of that, a malicious node can easily launch a DoS or Distributed DoS attack to the target server or client. A Distributed DoS or DDoS attack follows the concepts of a normal DoS attack but in this case, the adversary exploits a large number of distributed hosts to launch attacks to the target [76].
Wang [76] identifies two of the main DoS (DDoS) that peer-to-peer networks are vulnerable to:

TCP Syn Flooding attack

The attacker uses a forged IP address to send a SYN request to the target host. When the victim receives this request, it then replies with a SYN-ACK message and waits for the ACK message to finish the handshake. This message will not arrive because the attacker's IP is false; instead, he/she sends an extremely large amount of SYN messages to the target, exhausting the victim's resources and not allowing it to perform correctly.

Query Flooding Attack

This attack occurs in the application layer and it uses the fact that the queried node must broadcast the queries to all its neighbours in order to obtain the desired files. The malicious node then will create as many queries as possible to flood the network.

3.2.2 Sybil attack

The Sybil attack must be one of the most studied attacks in the literature [8][18][23][24][28][40][76]. In P2P networks, nodes join and quit the systems all the time, so it is not complicated for an adversary to introduce a large number of corrupted participants (Sybils) into the system [8]. These Sybils are controlled by the adversary, which can use them to gain control over the target objects. The attacker can choose the closest nodeID to all replica keys for a particular target object, hence controlling all replica roots. Then the attacker could delete, corrupt or deny access to the object.

Levine et al. [40] determined that half of the published papers either suggest certification as the problem solver or simply state the problem without giving a solution. They also placed all the approaches to prevent or deal with the Sybil attack into eleven groups: Trusted Certification, Resource Testing, No solution, Recurring cost and fees, trusted devices, Reputation Systems, Auditing and Cash economies.

3.2.3 Poisoning the network

This attack [41][76] consists of the adversary injecting false information into the system to break its integrity. The poisoning attacks can be divided into:

Index poisoning attack.

Here the adversary inserts fake records into the index server pointing to a target IP and port number. When other peers search for a resource, they will get false location information from a poisoned index server. Then those peers establish a TCP connection to the target that implies that these peers cannot get services from the victim node because the fooled nodes have occupied the allowed connections, or even worse, a DDoS attack can be launched.

Routing table poisoning attack

It targets DHT systems. The malicious attackers add the victim's IP addresses into the routing tables of a set of peers as their neighbours. Then the attackers send messages to the different peers with the victim IP address making the peers believe that the victim node is their neighbour. Therefore these peers could forward packages to the victim node. If we were talking about one peer, it would not affect the victim peer too much but this attack could reach thousands or millions of peers sending packages to the victim, which would cause the victim peer to crash.

3.2.4 Eclipse

The Eclipse attack [67][76] is a general attack in overlay networks. In this attack, an adversary controls a large amount of the neighbours of a trusted node, due to provoking malicious behaviour where adversaries attempt to inject fake nodes into other's routing tables. Since a node maintains just a handful of contacts in its routing table, if a significant fraction of these contacts is corrupted, such a node becomes isolated from the overlay. Consequently, incoming and/or outgoing requests related with the eclipsed node can be freely manipulated by the fraction of its malicious neighbours. Eclipse attack is still very effective because the adversary can easily support neighbours to fit in the top rows [67].

3.2.5 Traffic Analysis attack

These attacks focus on trying to obtain as much network traffic information as possible, such as message lengths and packet arrival. Some of the most important attacks will be explained in the next subsections:

Website fingerprinting

This attack exploits the structure of websites. The adversary wants to learn URLs of websites that are requested over an encrypted tunnel by the victim. The attacker gets this information by observing the network and when the victim visits a website, he/she receives packets in response to their query. Then the attacker analyses this metadata (packages length and quantity) and uses it to build a fingerprint of what the website's response looks like when it is fetched via an encrypted connection [24]

Timing attacks

These types of attacks are based on the fact that during the exchange of data (traffic), the time and duration of the communication can be registered. This information is examined to determine detailed data of the data flow, both identities of the communicating parts and location. An attacker with minimum knowledge would be able to follow the typical communication patterns. The adversary will then use this information to link inputs and outputs based on their patterns of packet inter-arrival times [13].

Predecessor attacks

The predecessor attacks are a real threat to the systems' anonymity. They look at repeated connections suspected to be to (from) the same correspondent and look at intersections of predecessor nodes to see which occurs most often. The attacker then tracks an identifiable stream of communications over a number of rounds (path reformation). In each round, the attacker simply logs any node that sends a message that is part of the tracked stream. The attack does not always require analysis of the timing or size of packets (although that can speed up the attack), but instead exploits the process of path initialization [81]. It has been proved to be effective to learn the communication's initiator in systems like Crowds [61] and onion routing [72].

Disclosure attacks

These attacks allow an observer to learn the correspondents of each user and, in the long run, de-anonymize their messages. A user, *Alice*, repeatedly sends messages to one of *m* different communication partners in each mix round. A passive adversary observes the messages entering and exiting the mix and wants to identify with whom the user is corresponding. When the attacker has observed *m* mutually disjoint a set of recipients containing Alice's *m* communication partners, he/she starts refining the data by observing new recipients and intersects with the previously observed and so on until the recipient is reduced to a single element.

3.2.6 Range Estimation Attack

It is an attack where a passive adversary, even though a lookup key is hidden, is still able to narrow the range of possibilities of a lookup target down to a small number of nodes, by analysing the locations of observed queries [78].

4. ANONYMOUS COMMUNICATION SYSTEMS

In this section, many systems and mechanisms are analysed. The main criteria for the selection of these particular schemes have been because either they introduce novel mechanisms that pursuit the anonymity or security of some kind; they are an important background in the design of other systems and/or their important because of its highly use. The anonymous communication systems can be divided in two main groups: low-latency and peer-to-peer communications. The schemes within these two categories have been carefully arranged in different groups depending on their main characteristics and in chronological order within each one of them.

The first system that needs to be described is the Chaum's mixes network [9], because although it is a high-latency anonymous communication system, and these systems are not part of this research, it was considered essential to briefly summarize it due to many systems have been influenced by it and some of its techniques are still used as a building block in newer systems like Onion Routing [72] or Tarzan [27].

Chaum's mix network

Chaum's mix network [9] was introduced in 1981 to enable unobservable communication between users of the Internet and it has become the basic building block of the nowadays high-latency anonymous communication systems. The communication is kept anonymous by using *public key cryptography*. The message between a sender and a receiver gets encrypted with the recipient address and the message itself and gets sent through a series of mix nodes. Each mix node is a processor that accepts a number of messages as input, changes their appearance and timing using some cryptographic transformation, and outputs a randomly permuted list of function evaluations of the input items, without revealing the relationship between input and output elements.

In order to relay a message through this chain of mixes, a client needs to learn their network addresses and public keys. This information is managed by centralized servers that are responsible to provide it for all the mixes. One of the issues of using centralized servers is that they become a main focus for adversaries. Hence, the security relies on this lists not being manipulated by an adversary that could attempt to exclude all honest nodes from the network or even launch a fingerprinting attack, substituting their public key and therefore being able to impersonate them [12].

Even though, mixes can be used to prevent traffic analysis – a user encrypts the message with a key for a mix nodeID, encrypts the result with the key from mix nodeID − 1 and so on with the remaining keys, the next mixes receives a certain

number of these messages, which they decrypt, randomly reorder and send to the next mix node in the routes-; Having to encrypt and decrypt the whole message becomes difficult for the mixes on the path to determine the encrypted message, on the other hand if they knew the content of the message, the anonymity would be compromised.

4.1 LOW-LATENCY ANONYMOUS COMMUNICATION

Low-latency anonymous network systems were considered secure against timing attacks when the threat model does not include a global adversary. Unfortunately, these systems, as mentioned above, have the scalability problem due to their client-server topology.
The systems that are studied in this sub-section are: Pipenet [15], Onion Routing [30], Crowds [61], Oceanstore [], Tor and PIR-Tor [52].

PIPENET

Pipenet [15] was presented by Dai as an anonymity system for low-latency traffic. Its design was based on Chaum's mix network [9] and its main goal was security up to the point of paranoia, because if the system detected any oddity that could be interpreted as an attacker, the entire network would shut itself down. Pipenet's theoretical architecture would provide protection against traffic analysis based on a distributed system of anonymizing packet forwarders.
PipeNet would build anonymous channels for low-latency, bi-directional communication, using layered encryption similar to Chaum's design. The process starts with the user selecting a random sequence of servers in the network. Then, as in the Chaum's mix network, the client set up a multiply encrypted tunnel by establishing a symmetric key with the first hop, tunnelling through that encrypted connection and establishing another key with the second hop, and so on. The design expects one packet sent over each link between network nodes during a time unit. If a packet is not received by a node over any one of its links, it does not forward any packets for that time unit, forcing all communications in the entire network to cease by simply not sending a message for one or more time units [24]. Unfortunally, if a node fails it would be misinterpreted and treated as an anonymity threat. This is the reason why Pipenet was never publicly deployed; it is impractical for real networks like the internet.
Its connections carry constant traffic, making it resistant to timing signature attacks, and disruptions to any connections are propagated throughout the network.

Onion Routing

Onion Routing [30] is a distributed overlay network design to anonymize TCP-based communications. It was the first Pipenet-like system to be widely deployed. This scheme provides an anonymous socket connection through the mixes [62], which applications such as Web browsing or instant messaging use to preserve their anonymity.
Clients choose a path in the network and build a circuit, in which each node or Onion Routing (OR) in the path only knows about its predecessor and successor node, but no other node in the circuit. This favours the unlinkability between initiator and final node [72]. The messages in onion routing are multiply encrypted with symmetric keys.
The initiator generates two symmetric keys: a forward key and a backward key for each OR on its path; and forward and backward cryptographic functions which correspond with these keys. These two pairs of function-key will be in charge of encrypting and decrypting the message along the path. When a node receives the message, it will decrypt the outer encryption layer with their own symmetric key, obtaining the pair function-key and the next node in the path. Then the node will encrypt the message using the new key and will forward the message to the next node. This process is repeated until the message arrives to its destination. Once the circuit is completed, the reply traffic will be sent encrypted in the opposite manner: each router encrypts and forwards the result of its predecessor onion router.
Onion Routing can tolerate the inclusion of attackers in the anonymous path. This is because the information that they could detect would be limited, since the next hops are encrypted, to the previous and next nodes identities. It also preserves the unlinkability in the communication. Nevertheless, this system does not offer a mechanism to prevent timing attacks in case of a dishonest node owns the first and last nodes of a chain [12] and also, in absence of large amounts of cover traffic, patterns of traffic are present, which would allow the adversary to follow the stream in the network. Neither has been proved to be able to keep the privacy in cases of a local adversary that is observing a target node's activities. The privacy degree depends on the number of compromised routers vs. total number of participants in the network [30].

Crowds

This system was designed by Reiter and Rubin [61] for anonymous Web browsing. The goal was to hide the information about the user or the information they retrieved from web servers. The idea is to hide one's action within the action of many others. The users in the system are represented by processes called *jondos* (nodes). An administrative process, *blender*, is responsible for assigning the *jondos* to a crowd of other *jondos* and for informing them of other members of the crowd [24]. Their task is to randomize the path from the initiator to the Web Server.
To execute web transactions in this model, a user first joins a crowd of other users. In order to do that, a node contacts a central server and receives a list of participants. The user's initial request to a web server is first passed to a random node of the crowd. That node tosses a biased coin and decides to either submit the request directly to the end server or forward it to another randomly chosen member. In the latter case the next member independently chooses to forward or submit the request. The messages are forwarded to the final destination with probability $p_f = ½$. Finally, the request is submitted to the server by a random member, thus preventing the end server from identifying its true initiator.
The reply to the request is sent using reverse path using the route established as the request was being forwarded through the crowd. Reiter and Rubin first analysed the probability that the initiator was correctly identified. They proposed the

notion of probable innocence as happening whenever the true initiator is identified with a probability less than ½.
In other words, this system operates by placing users into large groups (crowds) that collectively issue requests on behalf of their members. This way the web server knows to which crowd belongs the request but is not able to learn which member from the crowd it has originated from. Not even other crowds involved in the request know it. This way the anonymity is preserved.

The key feature that enables the anonymity in crowds is that upon receiving a message from a crowd member, we do not know whether this is the initiator of the message, or an intermediary who is just forwarding it. We can, however, compute the probability that each member in the crowd is the initiator of the message and quantify anonymity [17] as the entropy of this probability distribution. Crowds can also prevent a webserver from learning any potentially identifying information about the user including its IP address or domain name.

In [17], Danezis et al. show that the passing algorithm in crowds is optimal and thus all attempts to improve upon crowds are bound to fail. It shows that Crowds' paths lengths, and associated latency, is also optimal in providing anonymity within its system constraints. To provide better guarantees, more robust source routing is required to limit the adversary from learning the remaining time-to-live of intercepted messages. This advantage would be provided through cryptography, which would turn crowds closer to a mix-network scheme [9]. They also introduced D-Crowds, a TTL based scheme that can be adapted to accommodate any path length distribution, while learning the minimal amount of info. It supports any path length distribution, while leaking the least possible info, and quantifying the optimal attacks against it.

Oceanstore

Oceanstore [39] is a global-scale, highly available storage utility system that allows users to access nomadic data in a uniform global scenario after they have paid the providers certain fees that guarantee access to persistent storage. The service providers in turn use utility model to form agreement and resource sharing. Oceanstore finds the closest cached replica that satisfies the closest distance metric. Its servers use Tapestry to store and locate objects, facilitating clients to locate quickly and retrieve nearby file blocks by their ID, despite server and network failures.

The data stored mechanisms use primarily untrusted servers with encryption to offer high availability and prevention of DoS type of attack. Persistent objects are uniquely identified by a Global ID (GUID), which is the secure hash of the owner's key and a human readable name. This scheme allows servers to verify and object's owner efficiently and facilitates access checks and resource accounting; GUIDs are located by either a non-deterministic but fast algorithm called *Attenuated Bloom Filters* or a slower deterministic algorithm (modified *plaxton trees* [57]). Each message sent through tapestry is addressed with a GUID rather than an IP address; tapestry routes the message to a physical host containing a resource with that GUID. Tapestry is a locality aware if there are several resources with the same GUID, it locates (with high probability) one that is among the closest to the message source.

Oceanstore uses ACL for restricting write access to data, while read access is available with the keys. Updates are achieved using the Byzantine agreement protocol between the primary replica and the secondary. For high performance, Oceanstore also provides self-monitoring introspection mechanisms for data migration based on access platform. This is also used to detect cluster and improve routing performance.

Two important aspects differentiate its design goal: first, the ability to be constructed from unstructured infrastructure and, second, the aggressive promiscuous catching. While in an unstructured infrastructure any server can crash without warning, information can be leaked to third parties or be compromised; aggressive promiscuous catching provide a faster access and robustness to network partitions. Although it complicates data coherence and location, it provides greater flexibility to optimize locality and trades off consistency for availability. It also helps to reduce network congestion by localizing access traffic. Promiscuous catching requires redundancy and cryptography techniques to ensure the integrity and authenticity of the data.

Oceanstore API employs a byzantine-fault tolerant commit protocol to provide strong consistency across replicas. A version-based archival storage system provides durability. Oceanstore store each version of a data object in a permanent read-only form, which is encoded with an erasure code and spread over hundreds or thousands of servers. A small subset of the encoded fragments are sufficient to reconstruct the achieved object; only global-scale disaster could disable enough machines to destroy the archive object. The oceanstore instrospection layer adapts the systems to improve performance and fault-tolerance. Internal event monitors collect and analyse information such as usage patterns, network activity and resource availability. Oceanstore can then adapt to regional outages and DoS attacks, pro-actively migrate data towards areas of use and maintain sufficiently high levels of data redundancy.

Tor

Dingledine et al. [22] introduced Tor, a circuit-based low-latency anonymous communication system, as the second-generation onion routing. Its design's goals have focused on preventing attackers from linking communication partners, or from linking multiple communications to or from a single user.

Tor relies on onion routing [72] and on a distributed overlay network to anonymize TCP-based applications. Its network's architecture is formed by a list of relays (*network consensus*) that clients can download from directory servers. Each relay runs as a normal user-level process without any special privileges and clients can also download detailed information about them. The *network consensus* is signed by trusted directory authorities to prevent these from manipulating its content. Tor has improved and modified the Onion Routing design in terms of efficiency, deployability and security. Some of the advancement in security includes circuit creation, changes in the proxy and data integrity verification.

Tor uses an incremental path building design to create the circuit, where the initiator negotiates session keys with the next hop. This makes it more reliable and, in case of a node failure, a new node can be added to the path. To improve efficiency and anonymity, Tor multiplexes multiple TCP streams from the same source on a single circuit; limit the linkability by not multiplexing new streams in circuit that hold older –than-10-minutes streams; and avoid delays by having created circuits pre-emptively in the background [44]. In order to preserve the integrity of the user's paths, Tor must prevent adversaries from adding too many servers to the network. To control/manage the size of the network, Tor relies on a small set of well-known directory servers to decide which nodes can join. This brings a downside and it is that building anonymous paths turns into a bottleneck performance due to the relationship between the large number of users operating the system and a much lower number of servers.

Tor also presents a data integrity verification system before the node leaves the network to ensure that the message remains intact and is pointing to the right direction. It also integrates, a mechanism for responder anonymity via a location-protected server, allowing clients to negotiate rendezvous points to connect with them. Tor also proposes to use location-hidden services via rendezvous points. One of the major vulnerabilities for a hidden service in Tor is the server's selection of the first and last node in the communication path [62]

Despite these improvements, some issues with Tor architecture have been found [49]: Attackers find the central directory authorities an achievable target, scalability problems appear when the number of user increases - Adding more servers is an expensive option and churn will cause a bandwidth overhead-. Tor does not offer security against passive global observers which makes it fragile against possible routing and connection recovery. However, it offers security services through being highly usable and cheap to operate.

In addition, the centralized nature of Tor's design and its low relay/client ratio might allow an attacker to take over an important fraction of the relays in the system, especially since it only needs to maintain control over much fewer nodes than it would if it was a peer-to-peer infrastructure. Also, the reported relay/client ratio of 1/500 can severely limit the bandwidth available for tunnels [46].

In [6][22], Tor is exposed to DoS attacks by situating an adversary at the beginning or end of the tunnel, observing its traffic for a while and matching with wherever a colluding node is the first or last router. If there is a match, the tunnel is compromised; If not, the attacker will kill the tunnel by ceasing to forward all the traffic addresses to that tunnel. Tor is also weak against selective DoS attacks.

Bauer et al. [3] introduced a low resource end-to-end traffic analysis that exploits the fact that the system uses a preferential routing algorithm that attempts to optimize the performance. Therefore, if in addition to this, false resource claims are reported to the Centralized Authorities and this bogus information is propagated through the network, the attack is able to compromise Tor due to it having no mechanisms to verify fake resource claims.

PIR-Tor

This client-server architecture was introduced by Mittal et al. [52] to address the scalability problem in anonymous communication systems, Tor in particular. They proposed that the clients use Private Information Retrieval (PIR) techniques to obtain only a few relays instead of the entire database, as most of the peer-to-peer systems approaches, to overcome the scalability issue and to prevent the leak of information about the clients' choice of relays in suspicious directory servers, and therefore being able to defend itself against passive attacks and preserve the clients' anonymity. It follows the same circuit building process as in Tor.

The authors suggested two different solutions based on PIR. These architectures are *Computational PIR* –CPIR- and *Information-Theoretic PIR* –ITPIR-. The CPIR is a single-server scheme that uses the current directory servers for the distribution of the network information. Due to PIR, when Tor clients download a small block of descriptors from an untrusted directory server, this directory server does not learn which block has been downloaded by the Tor client. To avoid high overhead, it is advised to the clients to send fewer queries and reuse descriptors in subsequent time intervals. The second solution is ITPIR, which is multi-server and this solution relies on the client's guards being the trusted entry points to the Tor network, to fetch the descriptors for a circuit. It shows that is acceptable to follow Tor's rhythm that is a query per client every 10 minutes [52].

Both solutions have in common that a Tor client only downloads a small set of descriptors. Thereby, PIR ensures that only the client knows which descriptor has been downloaded. The authors prove how both solutions scale sufficiently to overcome the Tor's scalability problem [49][50]. However, PIR-Tor maintains the same security level as Tor.

PIR-Tor is scalable and robust against attacks like route fingerprint. Even though the directory queries are identifiable by an adversary, PIR protects the system against them and preserving the unlinkability between the relays retrieved from the database and the client. However, the leaked information could be used to relate connections from the same user and building *behavioural profiles* [56]. Also, if the first and last nodes are dishonest, an attacker would be able to discover the client by launching a *traffic confirmation attack*. Additionally, PIR-Tor has not been able yet to mitigate the typical issues in a centralized scheme like basic trust or DoS attacks.

4.2 SYSTEMS FOR UNOBSERVABILITY

As it was explained in section 2, when a system provides unobservability, it is capable of hiding all the parts involved in the communication – sender, receiver and message- from adversaries that are monitoring the system. The systems in this subsection are: p^5 [69] and Herbivore [29].

***p^5*: Peer-to-Peer Personal Privacy Protocol**

p^5 protocol was introduced by Sherwood et al. [69] for anonymous communications over the Internet. Its logical broadcast hierarchy is a binary tree constructed using the

public keys where different levels of hierarchy provide different levels of anonymity, at the cost of communication bandwidth and reliability. Even though this structure favours the scalability of the network, it reduces its efficiency. The nodes communicate by broadcasting their messages over the overlay, providing both client and server anonymity.

p^5 provides the individual participants a trade-off between degree of anonymity and communication efficiency, which can be used to scalably implement large anonymous groups, by letting them choose whether to join groups higher or lower in the broadcast hierarchy depending on if they desire more anonymity or better performance, respectively [24]. Because of this, users must select a level of anonymity and communication efficiency based on their expected performance. Each node of the tree is represented by a bit string of a specific length to denote its level and group. Users are then mapped to a node and a group.

The channel of the root of the binary tree is composed of the entire overlay. The channel of the left successor of the root is the left sub-tree of the overlay and the root, and so forth. So when a user wants to join the system, he/she needs to locate the root of the channel that they want to join and then, descends the rest of the tree structure uniformly at random and finally join as a leaf of the tree.

To send a message, a user first encrypts the message with the receiver's public key and then broadcasts the *ciphertext* to one of the broadcast groups the sender has joined. The node will send the message to the root of the channel it belongs to and then routed it up to the root of the entire overlay. Next, it will get distributed downwards the tree structure to the root of the channel and then broadcast through the transitive closure of its successors. If the receiver is not in one of the sender's broadcast groups, the message can be anonymously broadcasted across other groups in the binary tree.

p^5 relies on public key cryptography, which is slow and costly. It also assumes an out-of-band method for obtaining public keys, such as a trusted third party, a directory server, or even an anonymous public key server in the p^5 system. This method would allow a user to get the public key of some entity while preserving their anonymity. In addition, p^5 uses cover traffic to make statistical analysis by a passive adversary infeasible.

p^5 also utilizes a noise mechanism, which enables peers within a group to send packets at a fixed rate for concealing the initiator's ID. As a result, an adversary observing the network is unable to discern when a user is sending an actual message. The noise traffic is sent to a group chosen uniformly at random. Due to the constant rate of traffic from each user and broadcast nature of the system, p^5 users may have to drop traffic they do not have the resources to handle. Since groups higher up in the broadcast hierarchy receive the most traffic, those channels may experience the highest loss rates [24].

Herbivore

Herbivore [29] is a peer-to-peer scalable anonymous communication system formed by a *round protocol*, which works at the lower level that controls how bits are sent among the participant nodes, and a *global topology control algorithm* that divides the network into smaller anonymizing *cliques*. Nodes within each clique are logically arranged in a star topology so nodes will communicate using a central node. Each user in the clique has a shared key with every other member of the clique. At a higher level, cliques are arranged in a ring topology, allowing inter-clique communication. A structured overlay is used to route messages between cliques, where an eavesdropper can observe communication between them, but in unable to tell which member of each clique is communicating. When a new node wants to join the network, it is assigned to one clique. Herbivore guarantees that each clique will have at least k nodes, being k a predetermined constant that describes the degree of anonymity offered by the system [62]. Herbivore controls the size of each clique, in a way that if it becomes too large, it gets divided into smaller cliques and if it is too small, it will get merged with another small clique.

Herbivore uses computational puzzles in order to prevent nodes from joining arbitrary cliques. Unfortunately, due to that a clique can have up to 128 nodes; an attacker controlling a small fraction of the nodes in the network has a reasonable chance of having a dishonest node in any given clique. Therefore, he/she will be in a position of launching a Sybil or a DoS attack (or both) stopping the clique from transmitting or receiving packets. Nodes will move to new cliques if they cannot transmit, but then they will be exposed to intersection attacks [24][62].

Herbivore presents an *anonymous slot-reservation protocol* for collision avoidance. As detailed in [24], when a node wants to transmit data, first it will have to reserve bandwidth on the channel for one round. In order to reserve a slot, it needs to pick a random number $i \in [1, m]$ and then generate an m-bit reservation vector r with $r_i = 1$ and the rest filled with zeros. The rest of the nodes that do not want to transmit will set an all-zero reservation vector. Then, all nodes will broadcast anonymously and at the same time their reservation vector to the group. The reservation vectors will be filtered by a XOR function – only letting pass the resulting m-bit vector having a 1 in each position in which some node wants to transmit. Then the node will send the message during that phase of the round. During each transmission slot in a given round, each node locally computes the XOR of the message it wants to send and the pairwise keys it shares with all other clique members. The central node then computes the XOR of its own local value and those sent by the other clique members, and then broadcasts the result. This star topology approach requires $2(k - 1)$ bits to transmit one anonymous message bit.

4.3 CENSORSHIP-RESISTANCE SYSTEMS

Censorship-resistance can be defined as the combination of privacy, unlinkability, and robustness. Privacy means that nobody intercepting a message should be able to learn the contents of the message; unlinkability means that nobody should be able to determine whether two people are communicating to each other beyond the fact that they are each communicating with someone; and robustness, nobody should prevent two people from communicating with one another if they both wish to do so.

The systems described in this section fulfil these characteristics. They are: Freenet [13], Free Haven [21], Endsuleit and Mie's system [26] and Achord [33].

Freenet

Freenet [13] is an adaptive loosely structured decentralized peer-to-peer file system based on anonymity, without a Central Authority (CA), where users can publish, replicate and retrieve data anonymously. In order to preserve anonymity, it uses probabilistic routing, a variant of mixes in a redundant tree-based structure, where each peer maintains a local data store and a dynamic routing table, containing addresses of other nodes and data keys that they are holding. These keys might be key Keyword Singed Key (KSK), Signed Subspace Key (SSK) or Content Hash Key (CHK), and they are obtained through a hash function based on a 160-bit SHA-1 cryptographic function.

When a user sends a query, this query may be locally processed, or in case of failure, routed to the lexicographically closest matching node in its routing table. Because Freenet's nodes are encrypted and routed through other nodes, it is really difficult to determine who is requesting the information and what its content is.

To join the network, new nodes need to discover the address of at least one existing node in the system and then send *data insert* messages. To insert a new file in the network, the node first must calculate a binary key for the file, and then send a data insert message to itself. Then any node that receives the insert message checks its own storage to see whether the key is already taken. In case of collusions, the user tries again using a different key. If the key is not found, the node lookups the nearest key in its routing table and forwards the insert message to the corresponding node that propagates through the nodes until the hops-to-live limit is reached. If there is no key collision, a success message is propagated back to the original sender.

Free Haven

Free Haven [21] is an anonymous publishing system made up of a number of servers –servnets-, which agree to share and provide documents for anyone. The identities of these servnets are publicly known. All communications are made over an external Mix-based communication layer. When a publisher wants to publish a file, it breaks it into a number of parts using Rabin's information dispersal algorithm [59] and sends each part to a different servnet. When a reader wants to download a file, it must first find the hash of the file it is searching for and send this to a servnet. The servnet broadcast the request to the other servnets, which then sends the pieces of the file to the reader.

In this system, every node is equal to each other and transactions are carried out in a symmetric and balanced manner. Free Haven provides anonymous communications by using onion routing (OR) and sending queries through it to prevent attackers from tracing routes. Also, users and servers have pseudonyms to hide and protect them from adversaries. Reputations are assigned to each pseudonym and are tracked automatically. Nodes communicate by forwarding messages randomly amongst each other using different pseudonyms at each hop making it difficult for adversaries to determine a message's origin or destination.

Free Haven trades-off efficiency and convenience for anonymity, persistence, flexibility and accountability. The persistence of data published is based on duration instead of popularity, to prevent popular files from pushing out other files, restricting the damage that an attacker could create by censoring information. Free Haven was designed to resist censorship and to provide strong persistence; that is why the authors considered all possible attacks that could threat the system in its implementation.

Endsuleit and Mie's censorship-resistant system

Endsuleit and Mie [26] use the secure lookup scheme of Castro et al. [8] in a censorship-resistant publishing system. In this scheme, files are represented by sets of keywords.

A file is published by encrypting it with a key derived from its keywords and splitting it into two fragments, both of which are needed to reconstruct it. The fragments are signed with the publisher's private key, which allows the publisher to update or delete the file, and are stored in a distributed hash table at independent locations that are derived from the keywords. Readers only need to know the keywords to retrieve, reconstruct and decrypt the file; however, unless a reader has the publisher's public key there is no way to tell whether a corrupt node has modified the file.

The deterministic placement of data in distributed hash tables may be useful for censorship-resistance if it prevents corrupt nodes from making themselves responsible for storing particular files.

Endsuleit and Mie's results [26] show that the system offers protection of nodes, client and file's owners against legal prosecution, and therefore, it makes this system censorship-resistant. Also, reduces the chances of an adversary to launch a DoS attack because the authentication mechanism used on the file fragments does not allow generating several identities on one computer.

Moreover, to prevent the certificate authority from linking users to their files or nodes, signatures of publisher and node identifiers are blinded. Unfortunately this violates the requirement of Castro et al. that nodes should not be able to choose their own identifiers.

There has not been any recent work or improvements to this scheme.

Achord

Hazel and Wiley [33] introduced Achord, an anonymous improved version of the Chord lookup to use in censorship resistant peer-to-peer publishing systems. In order to provide censorship, each node has a limited knowledge of the system. The authors explained that in order to make this lookup mechanism suitable in this environment it must be capable of locating the data in a scalable manner while maintaining the following set of properties in the system [33]: The first two properties state that it must be possible to insert/retrieve information into/from the system without revealing the identity of the inserter/recipient, so an attacker will not be able to censor that information. The third

property says that "*it must be difficult to introduce a new node to the network*". Since nodes contain files, deleting them is not viable because it would be a way of censoring the contained documents. The last property states that "*It must be difficult to identify the node which is responsible for storing a given document*", for the same reason as before, deleting a node is not a viable option.

Achord's lookup mechanism is equal to Chord in performance and correctness, but it has been modified so that the node identification information is suppressed as the successor nodes are located [1]. Also, Achord defines some constraints for finger requests on Chord. A node cannot send a finger request to every node.

Achord's key lookup use an operation called *connect_to_successor*. When the successor of a node is contacted during the request, the value –*unlike Chord that is the nodeID*- is send back along the recursive search path to the originator of the request. Also, to insert a value into the system, a node performs a *connect_to_successor* operation to establish a tunnelled connection to the node which would be responsible for the key, and send the value along the connection path.

Tunnelling offers some degree of anonymity to the requesters and inserters, due to a node which receives a given request cannot determine if the immediate requester is sending the request from a node even further away from the key or not. In a similar way, the identity of the node responsible for storing a given key is protected. Achords attempt to disguise the identities of inserters and requesters might not be as effective as Freenet's [33]. This is due to that in Achord, a node receiving a request will have an idea of what the distance between the key and the requester's node ID is. Clearly, this raises an issue because this knowledge could be used to estimate the probability that the requester in fact originated the request. Even though, this scheme offers anonymity, the authors state that it could be vulnerable to correlation attacks. Achord cannot protect anonymity against a global passive adversary either.

4.4 PEER-TO-PEER COMMUNICATION

Peer-to-peer networks are composed of connections between anonymous nodes creating an Internet overlay. These networks can be unstructured, they use central servers to coordinate the communication between nodes, or structured, where nodes can communicate between each other and scalability and performance are controlled by the topology.

This section is divided into four categories: structured peer-to-peer topologies, scalable lookups, secure lookups and anonymity systems.

DISTRIBUTED SYSTEMS

In order to understand these systems behaviour, first an overview a distributed hash table is due:

A Distributed Hash Table (DHT) is a distributed system that supports a distributed lookup protocol that powerfully locates the node that stores a particular data item. Each node in a DHT has a unique identifier *nodeID* and the data identifiers from the same *spaceID* are called *keys*. The overlay assigns the ownership of a set of keys to a single unique live node. If a node owns a key then the node is said to be the root of that key. Data location is based on associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Every peer has the IP address of *logN* peers in its finger table. DHT provides two basic operations, each incurring *O(Log N)* messages:

- *put (k, data):* stores a key k and its associated data in the DHT.
- *get (k):* retrieves the data associated with k in the DHT.

To route queries efficiently, every node maintains a routing table with multiple entries of the form nodeID, IP address and port. Queries can proceed recursively or iteratively. In a recursive lookup, the lookup initiator contacts one of its *fingers* (routing tables), and that finger recursively passes the query to another one of its fingers. In an iterative lookup, the initiator contacts each intermediate node directly. Different overlays networks operate in different ways.

DHT routing has two main security problems [73]:

- Security in the query: When a node starts the lookup, there is a high probability that that query will pass through a malicious node. Besides, once the result of a query has been returned, there is no way to determine whether it is correct or not.
- Attacks on routing: They occur when adversaries try to claim some portions of the spaceID.

These issues make the probability of success, when routing between two honest nodes, become approximately *O((1-f) logN)* only, where *f* represents the fraction of malicious nodes in the system.

4.4.1 STRUCTURED PEER-TO-PEER TOPOLOGIES - SCALABLE AND FAULT TOLERANCE SYSTEMS

These systems are the base for most of the newer systems and/or mechanisms. They count with different fault mechanisms to detect errors and attacks but unfortunately they do not preserve anonymity.

Chord

Chord [70] is a scalable lookup in distributed peer-to-peer systems protocol responsible for assigning keys to the active nodes in the system. This system is formed by 2^n nodes, where they can be active or not. Chord follows a circular figure with an ID space of size *N* that is where the nodes are situated. Both the identifiers and the keys are situated in the same ring. The messages are forwarded only in a clockwise direction in the ID space.

Chord uses a distributed algorithm based on the hash function SHA-1 called *consistent hashing* to assign the keys to the peers. This algorithm is designed to allow nodes to join and leave the network with the minimal disruption and the hash function produces an m-bit identifier for the nodes and keys. Each node *n* keeps a routing table up to *m* entries, called a Finger Table (FT). A FT holds Chord and peer identifiers (plus port number). The successor nodes contain finger tables, where in their i_{th} entry of node n will contain the address of successor $((n+2^{i-1}) \bmod 2^m)$ in the clockwise direction. The node identifier *(NodeID)* is chosen by hashing the node's IP address and the *keyID* is produced by hashing

the data key. The *keyID* ***k*** is assigned to the first node whose *nodeID* is the same or follows ***k*** in the ID space. This peer is known as successor node. When ***k*** connects to the network, its successor node will transfer the keys that were for it. When a node abandons the network, it transfers the keys that its successor was in charge of. This hashing protects the system against adversaries.

The lookup process comes as a natural result of how the spaceID is partitioned. Both the insertion and querying of items depend on finding the successor of an ID. Any node looking for the key k will not need more that M hops[2], a node will discover the node at which k is stored. In general, under normal conditions a lookup takes $O(\log_2(N))$ hops.

If a node fails in Chord, it will lead to the loss of items and will cause the ring to break and therefore some IDs will not be found. Also, if after a short period of time a node does not respond, the nodes update both their successor's and predecessor's pointers and transfer the responsibility of the *keyID* of the failed node to the rest of the nodes [71]. To deal with these problems, if a node detects that its neighbour has disappeared, the node will replace it with the next node in the list and all the items stored in the failed node will be replicated in the node that follows it in the list or successor node. The only way to lose an item is when both the node which contain it and its successor node fail simultaneously.

This decentralized scheme tends to balance the system load, because each peer receives more or less the same number of keys, and there would be minimal movements when peers join or leave the system/network. The expected number of routing hops in Chord is $\frac{1}{2} \log_2 N$.

The main emphasis in Chord's design is robustness and correctness, achieved by using simple algorithms with provable properties even under concurrent joins and failures.

In [55] O'Donnell and Vaikuntanathan analysed how much information Chord leaks. They proved that a passive observer can only view those requests that are routed through itself, being incapable of eavesdropping on other links. They also conclude that, even though anonymity is not its design goal, Chord is capable of providing a high degree of anonymity against passive attackers in its recursive version, unlikely in its iterative version that it offers no anonymity at all. Larger successor lists also increase the anonymity set size of nodes close to data keys, an important consideration.

Pastry

Pastry [64] is a self-organising decentralized, structured overlay network that uses Plaxton-like prefix routing [57] and in where each peer routes clients and interacts with local instances of one or more applications. Its main goal is to route objects efficiently on the network. The nodes and the data items have unique 128-bit IDs, which go from 0 to 2^{128}-1, and they are distributed in a circular space. The *nodesID* are obtained using hashing in the IP numbers and Public Keys. Pastry organizes the ID space based on numeric closeness of IDs, and therefore, the nodes with adjacent ID numbers could be further in the physical space [25].

Each node maintains three tables [64]: *Routing table, leaf set* and *neighbourhood set*. The *routing table* contains $\log_2{}^b N$ rows with 2^b columns, N being the total number of nodes in the system. The entries in row *i* refer to a node whose ID shares the present node ID only in the first *i* digits. In that way, in the first row, the node's routing table contain ID nodes which have a different first digit (base 2^b). The second row of an ID node *n* contains 2^b-1 nodes that share the first of their digits but differ in the second one. The third one contains nodes that share the first and second digit but differ in the third one and so on. Hence, the node in the routing table with the smallest network delay will be included in the routing table. The *leaf set* contains a list with L/2 predecessors and L/2 successors. The node keeps a register of the m closest nodes following a different metric from the ID space, for example, network delay [25]. Finally, the *neighbourhood set* maintains information about nodes that are close together in terms of network locality.

The routing in Pastry follows the next steps: when a message is given with its key, the node first checks its leaf set. If there is a node whose ID is closest to the key, the message is forwarded directly to the node. Otherwise, the node checks the routing table and the message is forwarded to a node that shares a common prefix with the key by at least one more digit. If no appropriate node exists in either the routing table or neighbourhood set, then the current node or its immediate neighbour is the message's final destination. The number of routing steps in Pastry is at the order of *O(logN)*.

When a node wants to join the network, it needs to know another node that is already on the system. It generates an ID and sends it to this known node. The request will be routed to the node with the node whose ID is numerically closest to the new node ID. All the nodes found on the route to the destination have to send their state tables to the new node. It will initialize its own tables and inform the nodes of its presence.

As a fault detection mechanism, Pastry uses keep-alive messages between neighbours' nodes to check that they are alive. If a node has not been in touch with its neighbours for a while, it is supposed failed and all the members in its leaf set will be notified and their leaf sets will be updated. To replace a failed peer in the leaf set of its neighbour, its neighbours in the nodeID space contact the live peer with the largest index on the side of the failed peer, and request its leaf table.

Tapestry

Tapestry [82] is a structured peer-to-peer overlay routing infrastructure, in which nodes are assigned unique IDs and messages are routed incrementally through the overlay according to each node ID. Tapestry uses two secondary pointers in each neighbour map entry to allow routing in the presence of node failures. Objects to be stored in Tapestry are mapped to a node and to look them up request messages are routed on the system to the node, where the object is mapped.

The lookup and routing mechanisms of Tapestry is similar to Plaxton [57], which are based on matching the suffix in NodeID. Routing maps are organized into levels where each of them contains entries that point to a set of peers closest in distance that match the suffix for that level. Also, each peer holds a list of pointers to peers referred to as neighbours. In Tapestry, neighbouring nodes in the namespace are not aware of each other. Tapestry stores the

location of all data object replicas to increase semantic flexibility and allowing the application level to choose from a set of data object replicas based on some selection criteria, such as date. Each data object includes an optional application-specific metric in addition to a distance metric [42].

When a node's routing table does not have an entry for a node that matches a key's n_{th} digit, the message is forwarded to the node with the next higher value in the n_{th} digit, modulo 2^b, found in the routing table. This procedure, called *surrogate routing*, maps keys to a unique live node if the node routing tables are consistent.

The goal of tapestry is the ability to detect and recover from failures like neighbour map corruption or link failure. It can detect failures quickly and recover *routing state* when they are repaired. To detect server and link failures, Tapestry uses TCP timeouts. Also, to ensure the reachability of the message source, nodes send to their neighbours UDP periodic heartbeat packets. By checking the ID of each delivered message, faulty and corrupted neighbour tables can be quickly detected. In order to operate under failure, every entry in the neighbour map maintains two backup neighbours beside the primary one. As well, Tapestry's replica function produces a set of random keys, yielding a set of replica roots at random points in the ID space. The expected number of routing hops in Tapestry is $\log_2{}^b N$.

When a node detects that a node is unreachable, it marks the neighbourhood as unreachable instead of removing the pointer. It maintains a reasonable second change period, during which the message is still routed to the failed node. If the failure cannot be repaired in this period, the failed neighbour is removed from the map.

CAN: Content Addressable Network

CAN [60] is a scalable, fault-tolerant and completely self-organizing distributed infrastructure that provides hash table-like functionality on Internet-like scale. The key space of the CAN is a d-dimensional Cartesian coordinate space, where each node has an identifier that is mapped to a point P in the key space and is responsible for its zone -*rectangular portion of the key space that contains P*-.

CAN routes messages in a d-dimensional space, where each node maintains a routing table with O(d) entries and any node can be reached in $(d/4)(N^{1/d})$ routing hops on average. The entries in a node's routing table refer to its neighbours in the d-dimensional space. Nodes can route messages using only information about neighbouring nodes and their zones.Basic requests for keys like *insert, lookup* and *delete* are routed by the intermediate nodes that contain the key to zones using a *greedy routing algorithm*. Hereby, CAN nodes only have to maintain a small amount of states that are independent of the numbers of nodes in the system, which make CAN scalable. CAN's replica function produces random keys for storing replicas at diverse locations. CAN's routing table does not grow regardless of size changes in the network.

In case of failure, CAN's intermediate nodes will still be able to build a path for the request to be routed. The lookup is achieved by using the straight line path through the Cartesian space from source to destination [25].

When a node wants to leave the system, it hands over its zone and the associated key-value pair database to one of its neighbours. Then depending on whether the zone of one of its neighbours can be merged with the leaving node's zone or not, the new single zone will be created, or the zone will be handed to the neighbour with the smallest zone, making this node handle the two zones. When a node fails, a "take over algorithm" is put into practice but the pair key-value is lost.

One of the problems in CAN is that neighbour nodes may be geographically distant.

In CAN nodes are able to join/leave the system dynamically and without restriction, changing their managed zones each time. This makes the system unstable and weak against attacks like DDoS, due to an adversary being able to control different zones and corrupt the target node isolating it or bringing it down.

Viceroy

Viceroy [43], DHT-based overlay peer-to-peer network designed to handle the discovery and location of data and resources in a dynamic butterfly technique. It also allows nodes to contact any server in the network to locate any stored resource by name. Like Chord [70], Viceroy uses consistent hashing to distribute data and keep it balanced across the servers, making the network strong against churn. Viceroy organizes ID space as a circle of length one - and forms a butterfly-shaped neighbouring overlay nodes into $\log_2(N)$ levels numbered from 1 to $\log_2(N)$, where N is the total number of nodes in the system-. Level l node's two edges are connected to peers at level l+1. A down-right edge is added to a long-ranged contact at level l+1 at a distance about $1/(2^l)$ away, and a down-left edge is added at a close distance on the ring to the level l+1. The up edge to a nearby peer at level l–1 is included if $l > 1$. Then, level-ring links are added to the next and previous peers of the same level l [43].

Each node (apart from nodes at level one) has an "up" pointer and every node apart from the nodes at the last level two "down" pointers (one short and one long). Those three pointers are called the butterfly pointers. All nodes also have pointers to successor and predecessor pointers on the same level for short distances. This makes a total of 7 outgoing pointers per each node.

To lookup an item x in Viceroy, first a node n follows its *up* pointer until it reaches level 1. Then, it starts moving down the levels of the tree using the down links. In each hop, the node should traverse a pointer that does not exceed the target x. In the worst case, all the levels can be traversed up and down. This recursively continues until a node is reached with no down links, and it is in the vicinity of the target node. The expected lookup path length is O(logN) [25].

To join the network, a node looks up its successors, fixes the ring pointers and takes the required items from *s*. After that it selects a level based on the estimation of the number of nodes. It finds, by a combination of lookups and stepping on the ring, the rest of the pointers (successor, predecessor at the selected level, up and down pointers). When a peer leaves the overlay network, for reliability and fault resiliency, it hands over its key pairs to a successor from the

ring pointers and notifies other peers to find a replacement. Additionally, the stored items are transferred to the successor.

Koorde

Koorde [35] is a distributed system based on Chord [70] and De Bruijn graphs [20] which keeps a constant number of edges to maintain a low overhead. As in Chord, a Koorde node and a key have identifiers that are uniformly distributed in a 2^d identifier space. A key *k* is stored at its successor, the first node whose ID follows *k* in the identifier space. The last node is 2^d-1 and is followed by node 0. To insert a De Bruijn graph on the ring, each node needs to know about its successor on the ring and its first De Bruijn node. A De Bruijn graph maintains two pointers to each node in the graph, because of each nodeID is represented as a set of binary digits, each node is connected to nodes with identifiers 2*m* and 2*m* + 1, *m* being a decimal value of the node's ID. By succession shifting of bits, lookup time of logN can be maintained.

To look up a key *k*, Koorde's routing algorithm must find k's successor by going through De Bruijn graph. This graph of base n values and m bits of resolution will have a node identified by each possible combination of the n-m bits. Koorde exploits this ordered connectivity to reduce the state of each node in a network. Apart from the smaller routing tables, the rest of the protocol follows Chord.

One of the reasons why Koorde stands out is because it reaches a lookup performance $\log_k N$, with a low number of links, *k*=2; and it can also lookup with $\frac{\log N}{\log(\log N)}$ hops per request, when there are *logN* neighbours per node. However in order to be scalable, the maximum size of the network must be pre-determined so the key space can be configured. What it is more, Koorde requires a highly ordered and rigorously maintained organization, which has high maintenance costs.

Koorde uses Chord's stabilization protocol for maintenance and, in order to be fault tolerant, it is needed to keep an out-degree less than logN nodes so the nodes do not lose their contacts easily. The load balancing depends on the uniform distribution. However, the load of message passing on each node is an issue. In De Bruijn graph, some nodes will have more traffic than others by a factor of θlogN of the average traffic load.

Datta, Girdzijauskas et al. [19] argued that Koorde makes unrealistic, simplifying assumptions of uniform key distribution, showing that for an arbitrary key distribution, De Brujin graph fails to meet the dual goals of load balancing and search efficiency.

Kademlia

Kademlia [45] is a peer-to-peer information system that incorporates attributes from both Pastry and Chord and, seeks to improve efficiency and knowledge sharing. It uses the symmetric properties of bitwise XOR operations to determine the distance to a target node. Kademlia creates partitions of the ID space where the NodesID are leaves of a binary tree. In there, each node position is determined by the shortest unique prefix of their ID, and divides the binary tree in a series of minor binary trees that do not contain the nodeID and keep at least a contact in each of the subtrees. Per each subtree in the ID space *k* contacts are kept instead of just one. A group of no more than *k* contact in a subtree is called *k-bucket*, which is where Kademlia nodes keep lists of the known nodes sorted by time last seen-least-recently-seen node at the head, most-recently seen at the tail. The parameter k indicates the maximum number of entries one bucket can store. Kademlia algorithm [2] is based on distance of nodes and key-values pairs. Each node has a nodeID. A SHA-1 algorithm is used to build a 160-bit key from the node ID for each node. Keys follow the same process. Each node knows more about the nearest nodes than the further ones.

The lookup in Kademlia is concurrent and iterative. When a node is looking for an ID, it checks to which subtree belongs the ID and forwards the query to α nodes selected randomly from the subtree's k-bucket. Each node returns a minor subtree or closer to the ID k-bucket. From the return k-bucket the same process will be done and so on until the node ID is found. Lookup will end in *O(logN)* hops. To insert, a query is first sent to a few neighbours of the initiating node and then travels to the node which is the nearest node to the hash of the key-value pair. If a node goes down, another will take the responsibility of that region of the hash table space.

Kademlia uses parallel, asynchronous queries to avoid timeout delays from failed nodes. It also uses 128-bit routing table to speed up the search and maintains a separate list for each bit. Every list belongs to a specific distance from current node. As a result of this efficient method, each search iteration in a network with 2^n nodes will take at most n steps to complete [25][42].

When a node wants to join the system, it is inserted into its appropriate k-bucket and then a lookup for its own nodeID starts, while gaining information about other nodes that form the system. Finally, node n refreshes all its k-buckets and inserts itself into other nodes' k-buckets at the same time. If another node detects the new node and some of the keys in his database are closer to the new node than to itself, to provide consistency, the node replicates these keys to the new node. The more popular a key is, the more often it is stored at different nodes and the faster it can be found. If a node did not perform lookup in the range of one of its bucket within an hour, it picks a random ID in the bucket range and performs a search for that ID.

4.4.2 DISTRIBUTED SCALABLE LOOKUPS

When a system owns a scalable lookup, potentially it will be able to increase its size, allowing a higher number of users to join the network. With a larger network, and due to the peer-to-peer nature where nodes do not know each other, it will be common to find that more adversaries are trying to access the system if they are not in already. The problem of these adversaries is that, whether they are passive or active, they will try to create vulnerabilities in the system and, if they can, bring it down.

Skipnet

SkipNet [32] is a randomized structure based on SkipList, which is a sorted linked list where some nodes have pointers that skip over varying numbers of list elements in the increasing sort order, and therefore reducing the search time of a node in the list. Skipnet applies this idea to a ring structure, where data are node names – nameID- and nodes maintain supplementary pointers in the circle identifiers space. The nameID scheme allows the key to be stored locally or within a confined administrative domain – path locality-. To provide path locality, the linked list is changed to a doubly-linked ring and restricts the lookups in the DHT only to domains that contain the required key. All SkipNet nodes store *2logN* pointers where N denotes the number of nodes in the P2P. All pointers of a node constitute its routing table.

SkipNet supports constrained load balance. This is implemented by dividing the file name into two parts: a prefix and a suffix. While the prefix specifies the domain where load balance should occur, the suffix is hashed uniformly to the peers in that domain.

The routing efficiency is O(logN) with high probability where N is the number of peers in the P2P. SkipNet generates a random binary bit vector for each peer. These random bit vectors are used to determine the random ring memberships of peers. A ring at level *i* consists of all peers whose random vectors have the same *i*-bit prefix. Each level skips over 2^h nodes.

In Skipnet, a file is stored in the node whose nameID is closest to the file name. To provide content locality, the node name is used as the prefix of the file name.

Searching for a file in Skipnet can be done either by nameID or numericID. To search for a file by nameID, the query visits nodes whose nameIDs share a non-decreasing prefix of the target file name. On the other hand, in a search by NumericID, the querying node starts the search from the lowest level or Level 0. There, the lookup stops at the node whose numericID matches the first bit in the target NumericID. Then, it continues in the Level 1 ring until a node whose numericID matches the first two bits, and so on until the longest prefix is found at level H. Finally the process finishes when the numerically closest node to the numericID is found.

When a node leaves, Skipnet will continue to route correctly as long as the bottom level ring is maintained, then the upper-level rings will be repaired lazily. Each node maintains a leaf-set that points to additional nodes along the bottom ring which allows the partitions to remerge.

Bamboo

Bamboo [63] is a distributed algorithm based on the routing logic of Pastry [64], although the management of the overlay structure is different in order to be more scalable in dynamic environments. Its structure is formed by two sets of neighbour information at each node: *leafset* - comprised of successors and predecessors numerically closest in the key space - and *routing table*.

When doing a query, the predeccessor is forwarded until a node which has the key in its leafset to ensure correct lookup is reached. To improve the lookup performance, a routing table is used, which is populated with nodes that share a common prefix.

Bamboo performs lookups in *OlogN* hops, while the leaf set allows forward process in the case that the routing table is incomplete. Moreover, the leaf set adds a great deal of static resilience to the geometry; Gummadi et al show that with a leaf set of 16 nodes, even after a random 30% of the links are broken there are still connected paths between all node pairs in a network of 65536 nodes. This resilience is important in handling failures in general and churn in particular, and was the reason Pastry geometry was chosen for use in bamboo. Bamboo performs lookup recursively.

Bamboo has two recovery methods to a node failure: *Reactive recovery* and *periodic recovery*. In the reactive recovery, when the neighbours of a node fail, it will broadcast its updated routing and leafsets to all of its *k - 1* neighbours. This action may cause network overload in situations when all the nodes detect the failure at the same time and send their tables to each other, or when a node has not really failed but the keep-alive messages were delayed. On the other hand, in the periodic recover, a node is sharing periodically its leafset with each of the members of that set, independently of whether the node detects changes in its leaf set or not.

It has been proved that under low churn, reactive recovery is very efficient, as messages are only sent in response to actual changes, while periodic recovery is wasteful. As churn rates increase, reactive recovery becomes more expensive due to an increase on its leaf set size. On the other hand, periodic recovery aggregates all changes in each period into a single message, resulting in periods of no churn; reactive recovery uses less than half of the bandwidth of periodic recovery. In contrast, under churn its bandwidth use jumps dramatically. As results for reactive recovery are poor, Bamboo focuses on periodic recovery.

As a simple local optimization [46], a source node contacts another node in its routing table at level l and asks it for its level l neighbours to find out whether these other nodes have lower latency than some of the source node's existing neighbours. The same process is followed for the search key prefix. The results are compared and the source node's tables are updated if any closer nodes are found. The *inverse neighbours protocol* begins with sampling the node's neighbours at level l, only keeping the *k* nearest nodes from that set. Then, level l is decremented by one and another sample is performed on the remaining *k* nodes. This process continues until $l < 0$, with consideration paid at each step to possible new neighbours [63].

Bamboo supports two types of timeout calculations: *TCP-style* and *virtual coordinates*. The *TCP-style* timeout calculation scheme allows nodes to have a rough idea of expected base timeouts for issuing searches to different portions of the network. On the other hand, the *virtual coordinate* timeouts assign to each node a coordinate in a virtual metric space such that the latency between two nodes is represented as a line between them in the virtual coordinate space. Bamboo uses the virtual coordinate system found in Chord, called Vivaldi [77], which maintains an exponentially weighted average of past round trip times between nodes, and uses that to create reasonable timeout values.

Westermann et al' lookup

Westermann et al. [80] presented a scalable node lookup system based on Kademlia [45] for anonymity networks. All the servers have a unique nodeID and only those who provide anonymization service are members in the DHT. The users use a small set of servers which they trust to perform node lookups. Additionally, the results are not immediately used to build a connection to prevent timing correlations. Clients are not registered as members and to execute a query, users maintain encrypted connections to a few semi-trusted servers it knows and then send the results to these servers. They then execute the queries and send the results back to the user. This procedure aims to harden fingerprinting attacks [24].

In this design, the nodeID is the SHA-1 hash of the DHT public key of the node. Using cryptography limits the attacker's ability to freely choose its position in the DHT, preventing him/her from placing the same descriptor various times under different IDs and hideding him from updating the DHT with erroneous descriptors for already existing honest nodes. The private key used for signing the descriptors is also used for signing the server certificate which is used during the establishment of an encrypted connection to the anonymizing node. This ensures a one-to-one mapping between a descriptor and the corresponding server certificate. By verifying this certificate with the public key of the descriptor, the client can check if the server is the one referenced within the descriptor.

Eclipse attack is a threat, so it is fingerprinting. It does not protect against DoS or Sybil. In [56], Panchenko et al. showed that this approach does not provide enough security in big networks as an attacker can significantly bias the node selection.

4.4.3 DISTRIBUTED SECURE LOOKUPS

Unlike the systems in the previous section, networks with a secure lookup are robust against attacks. They introduce mechanisms that prevent attacks by hiding information during the lookup process. Some of the most known mechanisms will be explained below:

Castro et al's secure lookup

Castro et al. [8] proposed a strong DHT system that relies on redundant lookups, which floods the message along multiple paths. Each key is replicated among several replica nodes. The initiator performs multiple redundant lookups towards all the replicas. The lookup result would be successful as long as there are no malicious nodes in at least one of the redundant lookup paths.

The authors acknowledged the Eclipse attack as a threat in overlay networks. As a defensive measure, they propose the use of Constrained Routing Tables (CRT), where they impose strong structural constraints on the neighbour set. Even though an idealized CRT ensures that the expected fraction of malicious nodes in the neighbour set of correct nodes would be equal to the total fraction of malicious nodes in the network, the cost would be astronomical. They also introduced a set of mechanisms to counter the attacks against lookups and proposed the following techniques: *Secure node identifier assignment* -A trusted authority issues a certificate to each node and is responsible for assigning the node identifier with a public key-, *Secure routing table maintenance* -It creates a parallel, constraint routing table where each slot can only have a single possible node- and finally, *Secure lookups* -This property is divided into two stages. First, a routing failure test is applied to the sent message. If it fails, redundant routing is used and all the messages are forwarded according to the CRT. Even though these three techniques used together permit the lookup to return the closest node to the randomly chosen identifier, many extra messages are generated by the secure lookup mechanism when the routing failure test in unsuccessful. Unfortunately, the increment of messages every time this happens can alert malicious nodes that a lookup is being performed, losing its privacy and therefore its anonymity. Another limitation of this scheme is that the redundant lookups tend to converge to a smaller number of nodes close to the target, and one malicious node in this set could infect many redundant lookups [74].

Mittal et al. [49] put Castro et al's lookup into test, where they detected that a small fraction of 5% compromised nodes can detect the lookup initiator more of the 60% of the time. Furthermore, when the fraction of compromised nodes is increased to 10%, the lookup initiator is revealed 90% of the time. This shows how, even though this scheme's mechanisms are secure against active attacks, the lookup process loses its anonymity and can be observed by malicious nodes, exposing not only the lookup process but also all the anonymous communications that are built on top of it.

This scheme's security is examined in [74] against attacks such as Eclipse or Sybil. To defend against a Sybil attack, Castro et al's lookup proposed the use of a Certification Authority (CA), assuming that this can be trusted by all the nodes in the system and that it is capable of detecting Sybil attackers accurately. A CA would always be a target for attackers, so using it offline should reduce the chances of being attacked. However, centralized management of identities carries with it the problem of certificate revocation when identities are no longer valid for any reason, including online detection of a Sybil attack. On the other hand, as an Eclipse defence, Castro et al proposed the use of the *Secure routing table maintenance* to exploit the potentially vulnerable information, where its entries can be verified by a routing failure test and they are not taking into account network proximity. The authors did not test these aspects so a few years later Condie et al. [14] showed that attacks will progressively poison the optimized routing table, because after a while most of the routing would have had used it, which will cause an increase in the overhead in the table.

S/Kademlia

S/Kademlia [4] is a secure key-based routing protocol based on Kademlia [45]. Baumgart and Mies [4] propose several improvements in order to make Kademlia more robust against attacks. This scheme introduces a novel proposal of using crypto-puzzles to avoid collusion of malicious peers by restricting the nodeID generation, making

computationally expensive to generate valid nodeIDs. However, the adversary is free to generate valid IDs offline without any time bounds before actually joining and subverting the network. S/Kademlia also has refined two of Kademlia's mechanisms that include extending the routing table with a sibling list –sibling broadcast- to secure the protocol against storage attacks; and the use of multiple disjoint paths for node lookups to defend against routing attacks. S/Kademlia included this latter as an extension for Kademlia's lookup algorithm. This should increase the lookup success ratio in a system with presence of dishonest nodes. The initiator will begin the lookup by taking the *k* closest nodes to the destination key from his local routing table and distributes them into *d* independent lookup buckets [4]. Then, the node continues with *d* parallel lookups similar to the traditional Kademlia's lookup. All these lookups are independent and to provide really disjoint paths, each node will be used only once during the lookup process.

Baumgart and Mies only tested S/Kademlia in a network of 10000 nodes, although the results were promising, where in a system with 20% of malicious nodes the lookups using disjoint paths were 99% successful. Also the network topology can its diameter modified in order to adjust to the level of security in different scenarios.

Halo

Kapadia and Triandopoulos proposed Halo [36], a secure lookup scheme which provides a method for performing redundant routing over a Chord-based DHT to locate a target. These redundant searches are performed towards *knuckles* – nodes that have fingers pointing to the target.

Halo provides robust lookups over an unmodified distributed hash table by looking up nodes that are likely to have direct links to the target node due to the structure of the overlay. Unlike simple parallel lookups, these lookups are unlikely to converge on the same set of potentially corrupt nodes until the last hop.

Halo was tested on a network of 10000 nodes, where it shows that it is able to tolerate up to 12% of colluding nodes, with a minimal of 1% of failed searches. Opposite the 50–60% failure in Chord for the same system. The authors also stated that when using Halo recursively, the tolerance of colluding nodes increases to 22% with 1% of sabotaged searches athwart Chord's 70-80% [36].

In [58], it indicates how Halo is able to defend against path construction attacks – *adversaries drop or mis-route path setup messages to other adversaries*- by providing redundancy mechanisms in the overlay and also using a random symmetric key. However, tt is subject to Mittal and Borisov's information leak attack [49].

4.4.4 PEER-TO-PEER ANONYMITY SYSTEMS

In this section, there have been included systems that are used over a P2P topology. Hence, there will be two subsections that contain: random paths in unstructured topologies and random lookups in structured topologies.

4.4.4.1 RANDOM PATHS IN UNSTRUCTURED TOPOLOGIES

These systems connect nodes relays into an unstructured topology and construct circuits along paths in it as a method for scalability. They also introduce mechanisms that randomize these actions in order to hide relays, users and messages from adversaries.

Tarzan

Tarzan [27] is a decentralized anonymous peer to peer network based on the IP protocol. Each node in the system has a set of neighbours –*mimics*- that are based on the IP address of this node, this way, nodes with IP addresses from the same subnet are grouped together. They are responsible for exchanging continuous dummy messages at a symmetric rate to cover traffic that is being exchanged.

The anonymous circuit construction in Tarzan commences with an initiator choosing the first hop randomly from their set of mimics. Then the second hop will be selected from the set of mimics chosen by the first hop, and so on. In each hop, symmetric keys are generated and encrypted with public keys of the servers in the circuit, in a similar way to onion routing [72]. These keys are used to send data over the circuit. All the users in the network relay traffic for other users, like in Crowds [61].

Since the initiator of a circuit is also exchanging mimic traffic with other nodes, someone watching the node has a greater difficulty identifying it as the source of a particular circuit. Also, the first hop in a circuit does not know whether the traffic it is receiving is cover traffic or application traffic [24] and therefore, the network would be stronger against traffic analysis attacks.

Tarzan introduced the peer-to-peer *gossip protocol* that allows clients to learn about other servers in the network by sharing the information about them. This way, when a node initializes, it will select a random neighbouring node that it already knows about and will ask it for all the other servers the neighbour knows about. In the same way, the requested node can then select another random node from the newly learned set of servers and repeat the process.

Due to the communication on Tarzan being carried out over links between mimics, each node needs a global view of the system in order to verify that the paths that are being built correctly and to keep the information updated in the gossip protocol. This causes that Tarzan gets limited to a network size of 10000 nodes or less.

Due to the peer to peer nature of the network, if a malicious node is contacted by the gossip mechanism, it would be in a position of launching an attack. Tarzan provides anonymity against malicious nodes and global eavesdroppers.

Rumor Riding

Han and Liu [31] introduced Rumor Riding (RR), a light weight mutual anonymity protocol for decentralized peer-to-peer systems. It uses a random walk scheme as the building block of the protocol. It allows messages to be sent via multiple anonymous paths without considering path construction.

Rumor Riding uses the AES algorithm to encrypt messages with a 128-bit size key. When an initiator wants to start an anonymous query, it generates the query content and a public key *K*. Then it encrypts that query message with the symmetric key into a cipher text and sends them both to different neighbourhoods. They follow different random walks –*rumors*- in the system. When the key-cipher text pair meet in a node, this node is capable of recovering the original query message. For this node to determine if the pair of key-cipher rumors matches, RR utilizes a *Cyclid Redundancy Check* (CRC) function to attach a CRC value. When the node decrypts both messages, it can compare results, being able to know if the message has been successfully recovered. Once this node has checked that the message is correct, it respond the query. In order to do so, it encrypts the plain text of the response message with the initiator public key *K*. Then, it encrypts, using AES algorithm, the cypher text and its key and split them into two response rumors, which will get an ID assigned, *IDrK* and *IDrC* for the key and the message, respectively. Finally, the initiator will send a confirmation message to the responder using the responder's public key. Following the same process, the message is splitted in two and sent following different paths.

There is a storage overhead concern raised by the need of each node to cache a number of received rumors before they are matched.
Also, RR does not sustain much storage overhead to individual peers, which is an issue since this aspect is highly related to the speed of query generation [29].

Rumor Riding is susceptible to timing attacks, since the adversary deduces the correlation between the timing of packets. RR is not subject to a predecessor attack, due to the sowers of an initiator or responder being randomly distributed over the system and not unique, the attacker will not be able to identify any of the parts. Even though RR is less vulnerable to traffic analysis attack than other anonymous systems, adversaries can easily discover the initiator identity by using reverse path [31].

Agyaat

In [66], Singh et al. introduced Agyaat, a decentralized peer-to-peer system, which promotes a generic non-cryptographic solution for mutual anonymity for both sender and receiver. When a node in a structured system wants to join Agyaat, it needs to join one or more unstructured clouds because messages in this scheme are addressed to clouds instead of nodes.

The routing begins when a sender's cloud initiates a random walk to hide the identity of the initiator. Then, the message is forwarded through the structured overlay to the rendezvous node of the recipient's cloud, which broadcasts the message through the cloud, secreting the identity of the recipient. To balance load among cloud members, Agyaat uses multiple structured overlays with independent key spaces; a given cloud may have a different rendezvous node in each overlay.

Agyaat offers three alternative resource discovery approaches*: semantic groups*, *centralized directory service*, and *dynamic services*. In the first case, nodes that host semantically similar resources are grouped into a cloud. Then, some sort of resource and provider privacies can be provided at the expense of resource flooding-based discovery. The second approach is a centralized directory service, which improves the discovery of the resources. Also dynamic services can be employed for this purpose. Then, a resource is mapped to a cloud and the index is stored at a central server or at the coordinator peers of the clouds in a distributed manner. However, Agyaat does not describe the anonymous construction of this index and it does not analytically quantify its effectiveness.

4.4.4.2 RANDOM LOOKUPS IN STRUCTURED TOPOLOGIES

The systems defined in this sub-section use DHT secure lookups in order to find a random node in the system. The most important anonymous systems and lookup mechanisms have been studied and they are:

AP3: Anonymous Peer-to-Peer Protocol

AP3 [47] is an anonymous protocol built on top of Pastry [64], which is responsible for handling the membership overlay. Its design is similar to Crowds [61] with paths being formed by performing a stochastic expected-length random walk. The stochastic nature of AP3 makes it difficult for a rogue node to decide whether its preceding hop is the initiator or simply a relay in the path; however, for low-latency communication, timing attacks may make this decision simpler.

AP3 offers three different services to the node that are: *Anonymous Message Delivery*, it is sent one particular message to a node through a random path in the network; *Anonymous Channels*: allow a persistent tie between a node and an ID while maintaining the nodes anonymity in the network; and *Secure Anonymous Pseudonyms*: allow a node to have a persistent id within the network which is authenticated through public key encryption.

To deliver anonymous messages, like Crowds, AP3 trusts on a network of peers to forward messages while hiding the originator. In fact, an intermediate node does not know if the node that it received a message from is the originator or just another forwarding node. Therefore, the only node that knows the initial node's identity is the one that handed the message.

To send an anonymous message, first, the anonymous request object is created comprised of the message and address of the recipient, without revealing any information about the originator's identity. Then, the request is forward to a node selected by drawing a random key. When the request is received, AP3 toss a coin to decide whether to forward it to the destination node or to another intermediated one, following a *forward probability mechanism* (p_f) to provide a random path through the network that gets built from a variable number of random hops. The path length follows a geometric distribution, with the expected length being $\frac{1}{1-Pf}$ [49].

AP3 allows users to have pseudonyms. Each user is able to generate as many public and private key pairs as necessary without a Public Key Infrastructure (PKI), creating anonymous pseudonyms that cannot be linked to

each other. The owner of a pseudonym establishes an anonymous channel and must also periodically refresh the anonymous channel associated with the pseudonym just in case the nodes along the channel have died. Messages targeted at pseudonyms are encrypted with their public key, so the user who owns the pseudonym is the only able to access the contents of the messages.

In [49] Mittal and Borisov presented two attacks that they used to analyse AP3. The results showed that the limit on the number of attackers that AP3 can handle while providing *probable innocence* is only 8% in the typical case, while the theoretical limit with increased path lengths is 10%. This is in contrast to the conventional analysis, which puts these figures at 33% and 50% respectively. These results prove that the lookup used in AP3 reveals a lot of information about the lookup initiator, making the user vulnerable to passive information leak attacks; and therefore, the anonymity in the path construction can be compromised.

Cashmere

Cashmere [83] is a recursive DHT based network overlay on top of Pastry DHT. It uses mix relay groups over single node mixes to improve system reliability. A node has *k*-bit signed *nodeID*. The nodes are divided in groups with *m*-bit IDs, where *k* is equal or greater than *m*. The *groupID* serves as a prefix of the nodes. A node participates in a group if the ID of the group is a prefix of its ID. Every group has assigned public/private keys and the group members receive them. The nodes obtain also the public keys from the other groups. Messages in the overlay are routed using group-IDs. The first found node that has this group-ID as a prefix is responsible for forwarding the message to all other members in this group. The forwarding of messages is done through n groups. The sender encrypts separately the path and the message in layers using the public keys of the groups. Thus the same path can be used multiple times. A group in the path decrypts the path message with its private key and finds out the next group in the path and the symmetric key to decrypt the outer layer of the message.
Cashmere uses end-to-end acknowledgments to detect failures and malicious nodes: if the source receives no acknowledgments, it can use timeouts to guide retransmission. Return messages are sent by including an encrypted return path, along with the symmetric keys necessary to encrypt the responder's payload, in the initiator's first payload [73].

Borisov et al [6] analysed Cashmere by using a passive adversary and dishonest nodes to observe the system behaviour and to measure its anonymity. It was witnessed how even in this unwished scenario, if at least one honest node acts as the relays group root for every relay until the destination, the connection remains reliable. However, since any node in the group can decrypt the current layer of the message, the connection can be insecure in cases where there is a malicious node in each relay group. Luckily, since the destination is not revealed in the message, the final node would also have to be malicious to compromise the sender anonymity.

When an adversary controls any of the relay group roots, it would be in a position to launch a DoS attack, dropping any connection that goes through them. Because of that, in order to keep the anonymity either every relay root and the destination node are honest or the whole path is compromised. Zhuang et al [83] did not consider DoS a security concern while designing Cashmere.

In 2009, Tran et al. [73] tested Cashmere anonymity; they showed that in a system with 64,000 nodes an adversary that controls 20% of the nodes can completely compromise 42% of the circuits, whereas the analysis in [83] suggested that 90% of malicious nodes are required for effective traffic analysis.

Salsa

Salsa [53] is an anonymous communication system designed to overcome the scalability problems in traditional mix systems, and to perform robust and reliable lookups. Salsa is based on a Chord-like DHT that maps nodes to a point in a spaceID corresponding to the hash of their IP address. This spaceID is divided into groups following a binary tree structure where each node knows the rest of the nodes in its group -*local contacts*- and some random nodes in different groups -*global contacts*-. Salsa's design was made to resist attacks on its DHT functionality by using verifiable IDs, bounds checking and redundant lookup.

In the lookup process, the initiator must reach its global contact in the sub-tree as the destination ID to continue the lookup. The lookup proceeds in a recursive manner until the destination identifier is in the same subgroup as the intermediate requesting node; if so, this node returns the IP address and public key of the closest node to the target ID.

The Salsa binary tree architecture is designed to ensure that redundant paths have very few common nodes between them, and therefore not becoming a target for attackers. In this redundant lookup, the initiator uses some local contacts to execute lookups for a random key. Then when the closest value to the key was returned, a bound checking is performed. If the bound check fails, the key is rejected and the process to find a new random key will be repeated.

Salsa, aiming to keep robust against attacks, has incorporated circuit-building into the redundant lookups. In this way, the process will start with the initiator choosing *r* random IDs and redundantly looking up the corresponding nodes. Each of the first set of nodes will launch a simple lookup for *r* nodes. Finally, a circuit is built to each of the nodes in the second group through one of the nodes in the first group. The same simple lookup process is repeated for the second set of nodes for a final node that will be added to the circuit.

Nambiar and Wright [53] introduced an attack against Salsa that proves how the system will be compromised if there is at least one malicious node in each stage of the tunnel or even if the first and the last forwarding nodes are compromised.

Mittal [48] analysed the security of Salsa. The results to his tests show that Salsa is robust to node churn and to a number of messages due to lookups. The use of bounds checking decrease the lookup failure a 85%, where this mechanism is able to eliminate a large percentage of biased results – only a 1.3% of malicious results are capable of fooling the bound checking procedure.

In [6][73], it is clear that the redundancy in Salsa's path building mechanism functions well against active attacks but

it provides more opportunities for passive attacks. What is more, Salsa starts losing privacy in scenarios where the fraction of the malicious nodes reaches the value 0.12 ($f > 0.12$). Borisov et al. [49] conclude that a DoS attack in Salsa reduces the anonymity considerably because only fully honest or fully compromised paths will survive. |Also it has been showed in [49][73] that Salsa does not perform anonymous lookups and can compromise the anonymity of its users.

ShadowWalker

ShadowWalker [50] is an anonymous communication system based on a random walk over redundant structured topologies. The authors introduce *shadows nodes* whose tasks are: to verify if the routing table from a given node is correct and certify it as correct. It tries to avoid information leaks [51] by using these certificates to check the different stages of a random walk. These walks are always performed over a secure Chord-like. Shadows are the neighbour nodes that vouch for responses to lookups and try to stop adversaries from obtaining them. They provide digital signatures on routing tables whose task is to perform random walks and maintain DHT routing.

A node's shadows are chosen verifiably at random, making it unlikely that an attacker controls a node and all its shadows; a mechanism for preventing Sybil attacks [23] is assumed. As with the DHT, each node must periodically run a stabilisation protocol to find its correct neighbours; the addition of shadow nodes makes this more expensive.

ShadowWalker uses a secure lookup protocol specially designed for redundant structured topologies. This process starts when a node *n* want to find an identifier ID. Then, being *m* the closest node to the ID in the finger table, *n* will query *m* for its finger *p*, which is the closest preceding node for ID. Now, *n* knows all the shadows of *m* so it is able to check that the information is correct. This process will be repeated iteratively until ID is found. As long as one of *m* shadows is honest, *n* will learn the true identity of *p*. The lookup will be successful if at least there is one honest node in each step of the lookup. This secure lookup is based on the assumption that adversaries will never obtain a corrupted neighbourhood.

In Singh et al's eclipse attack [67], the attacker poisons the routing tables by returning dishonest instead of the honest ones as a reply to the lookup query. This attack would affect ShadowWalker because each step of the lookup depends on the previous step to provide the correct shadows for the node currently being queried.

A single intermediate node cannot launch a *route capture attacks* because its info is verified by the shadows. However, if these nodes and all its shadows are compromised, they can launch this attack by returning colluding malicious nodes as next hops or by modifying the public keys of the remaining nodes to emulate them. This means that if an intermediary node and its shadows are dishonest, the remaining nodes in the circuit are also dishonest. In case of the first node and its shadows are malicious, the initiator's identity is compromised. ShadowWalker is also vulnerable to end-to-end timing analysis, due to if both ends of the circuit are compromised, the circuit anonymity is broken.

Schuchard et al. [68] put ShadowWalker through an Eclipse and *Denial of Shadows* attack. The eclipse attack in ShadowWalker permits the adversary to gain control of a full neighbourhood of the network and thus, corrupting the shadow mechanism: When a malicious node is asked about another node in the network, they will provide a false ID and its corresponding false shadows. Unfortunately, this mechanism is used as well to build the routing tables in DHT, and it is determined in [68] that only with 10% of the nodes compromised, the attack can corrupt up to 90% of the circuit. The Denial of Shadows attack exploits the number of signatures that are required for a node to participate in the circuit construction. A secure lookup is considered successful when a node provides at least one signature. As during the random walk nodes must present a full set of signatures in order to build the circuit, an adversary could refuse to provide a node with a matching signature to make it non-viable for circuits. Even though, ShadowWalker uses symmetry in the shadows (*n* is shadow of *m* and *m* is shadow of *n*) to try to beat this attack, the reality is that the solution is not good enough, making this scheme vulnerable to the denial of shadows attack.

NISAN: Network Information Service for Anonymization Networks

Panchenko et al. [56] introduce a new scheme based on DHT to deal with the scalability issues in anonymous distributed systems. It proves that it provides a better redundancy than other previous mechanisms by introducing NISAN, a system based on Chord-like DHT which uses the following methods to restrict the malicious behaviour: Aggregated Greedy Search, Hiding the Search Value and Bounds Checking in Finger Tables.

In a case of *nodeID*s uniformly distributed in the spaceID, the *aggregated greedy search* starts with an initiator *v* generating a random ID *x* and, per each round, choosing the closest α nodes to *x* that it knows, and send the query to them. The search finishes when after one iteration, the list of α closest peers[2] has not changed. The knowledge is available on each of the different branches of the network. It has been proven to work well in networks up to 50000 nodes, any higher and it produces an increased redundancy of considerable proportions. Panchenko et al explain that NISAN will keep its scalability in a case of malicious colluding nodes located arbitrarily in the system, although it will be vulnerable to high attacks rates.

Another NISAN's strong feature is *Hiding the Search Value*. In order to gain extra redundancy while executing the lookup and avoiding spiteful nodes next to the ID, *v* would request the entire finger table instead of just the ID *x*. FTs contain $\log_2 N$ aggregated entries and the best will be selected for the next iteration. This process is repeated until the top of the list or the closest peer is found. It then will return the result of the search.

The final step in order to provide an anonymous lookup is to perform *Bounds Checking* in the retrieved Finger Tables to detect if colluding nodes have introduced malicious nodes in the FT. This process consists of calculating the mean of the distance between an ID in its FT and the optimal ID -mean distance-, and multiply this value with a factor – FT tolerance factor-. This test will be run in

each step of the lookup and only nodes that have passed it will be considered.

The results show how NISAN's new methods are strong against active attacks but the leaks in its lookup lead to a reduction in the user anonymity and how the lookup path is more likely to involve a malicious node as it increases its size.

In [78], both active and passive attacks are used to observe NISAN behaviour against them. The analysis consists of using NISAN lookup to test three different circuit creation mechanisms in anonymous communications. In order to analyse the behaviour, the authors consider a partial adversary who controls a fraction *f* of all the nodes in the network, where f will be less than 0.2.

Wang et al. [78] analysed the use of the NISAN lookup in three circuit construction mechanisms for anonymous communication. They demonstrate how the NISAN lookup leaks information that leads to an important reduction in user anonymity in all the three strategies. They also tested NISAN with their own passive attack model, in a network of 10000 nodes with a 20% of colluding nodes. The results show that this attack is able to reduce the entropy of the circuit initiator by 2.2 bits, which is close to the ideal passive attacks whose entropy reduction is 2.6 bits. NISAN is also weak against a rage estimation attack.

Torsk

McLachlan et al. [46] introduced Torsk, a DHT-based anonymous communication protocol that uses a combination of two peer-to-peer mechanisms to avoid attacks to the integrity and confidentiality of the lookups. It includes an interesting alternative design for circuit construction by using what they called *secret buddy* nodes that serve as proxies during the lookup. Torsk requires each node to privately select a number of random buddies from the network and then a lookup initiator will request one of its buddies to perform the lookup on its behalf, so that the attacker cannot associate the final target with the initiator. Every time a malicious node returns an invalid certificate, the buddy selection process needs restarting.

Torsk uses an iterative lookup scheme based on Kademlia DHT [45] and Myrmic [77]. An initiator who wants to find a target *x*, starts the lookup process by selecting the closest *t* fingers to *x* from its Finger Table (FT), and uses them as starting points for *t* independent lookup branches. The initiator keeps a best list of closest fingers to x for each lookup branch. In each iteration, the *t* fingers that have not been contacted are selected from each best list and queried with *x* in parallel. Any requested node returns *k* fingers closest to *x*. The parallel lookup process finishes when any best list is unchanged at the end of one iteration [46].

Each node in Torsk keeps a certificate issued by a trusted Central Authority (CA). It includes all its fingers and its information. In this manner, a querier is able to verify if the returned node is responsible for the *x* ID or, on the contrary, it is a forged node.

A very specific attack is run against Torsk, it is the *buddy exhaustion attack* [56]: an attacker can prevent the circuits that have honest entry nodes from being extended, they can do this by exhausting the buddies of the end relays, of these partial circuits. This attack can ally with a DoS attack to affect the circuits. The results in [56] show how the adversary has a good chance to interrupt the buddy selection process, because when an invalid certificate returns, the random walk is restarted. Then the adversary can let malicious nodes get involved in a random walk return to invalidate a certificate, preventing the querier to find new buddies.

In [78], Wang et al also analysed Torsk and determined that its lookup is vulnerable to both passive and active attacks. They suggest some changes in the random walk process for buddy selection in cases where a node returns an invalid certificate, indicating that Torsk should not start the process again but instead, just return one hop back and choose a random finger as the next hop. In addition, if all fingers in a hop are malicious, the system could return another hop and repeat the process. This way, an attacker will not interrupt a random walk and honest nodes can find honest buddies. Unfortunately, even with this improvement, Torsk is still vulnerable to the buddy exhaustion attack.

Bifrost

Kondo et al. introduced Bifrost [38], an anonymous communication system that separates the Node Management Layer (NML) from the Anonymous Routing Layer (ARL), combining multiple encryption and node management using Chord as DHT. The advantage of separating NML from ARL is that the node management remains independent from anonymity. Bifrost scheme is comprised of multiple nodes and a Public Key Server (PKS) that manages the public key for each node. When a node wants to connects to the system, it follows a participation procedure and registers its public key in the PKS.

The NML layer manages all *nodeIDs* using Chord. It also uses its algorithm. In Bifrost, nodes only keep the IP address of their successor and predecessor. On the other hand, the ARL layer, as mentioned before, uses multiple encrypted messages. To avoid traffic analysis attacks, the final receiver is located half way on the route. Also the relay nodes are chosen by the initiator, basing its choice in their number and round trip time. This layer uses three types of messages: construction, data and control. The relay nodes and the final receiver memorize the connection information on each anonymous route.

When a node secedes, NML assigns automatically a backup node (*BN*). The *BN*, which is a successor of the seceded node, will need a private key and common keys owned by it to reconstruct the route but because it does not have keys owned by the seceding node. The new relay node then begins to entrust its own keys to a successor and a next-successor immediately, being capable of restoring the anonymous path independently to ARL.

Bifrost presents three features that are: A source that can locate an arbitrary destination position in an anonymous communication channel, different routes from and to a destination; and a *receiving area* (RA) to reduce the processing time of message encryption and searching for a next node. This RA is a subspace of a *spaceID* that contains consecutive Chord IDs. The first node of a RA receives a message, which is repeatedly relayed to successors by nodes in the RA until the message reaches the RA's end node. Finally, the end node sends the message to the first node of

the next RA. A source can arbitrarily set the number of receiver areas and a beginning and ending nodes form them.

Kondo et al. [38] determined that conspiracy attacks can deduct the sender and the final receiver by sharing information by considering two or more nodes. They also analysed how final terminal nodes of a RA affects to its security and anonymity: *If the final terminal node of a RA and a node are honest*, the attacker will not be able to trace the relationship between RAs, not being able to learn the entire route, sender or final receiver. If, on the contrary, *a final terminal node of a RA is malicious*, this node will be able determine the connections between this RA and the next RA, due to its capability of decoding the message and detecting that they are the same before and after decoding the message. Finally, if *all terminal nodes of RAs conspire*, an attacker will be able to learn all RAs. However, they will not be able to discover the final receiver because the final receiver and the sender have disappeared somewhere on an anonymous route.

The downside of the performance is that generates delay due to some nodes relay a message. The results show how the processing cost of a terminal node of a RA is larger than the cost of a general relay node due to the big increase at the decoding processing time. Hence, when the number of RAs increase, the communication delay increases more than if the number of relays were increased instead. Unfortunately, Bifrost **has not been tested** on Internet yet.

Octopus

Wang et al. [79] introduce a novel, anonymous and secure DHT lookup mechanism, which provides guarantees for both security and anonymity. Its strong point is a novel attacker identification mechanism used to discover and remove malicious nodes.

Octopus uses three techniques to achieve its security and anonymity goals. These are: Anonymous path construction while hiding their initiator –acquiring the whole finger table, splitting the individual queries and introducing some dummy ones to disguise the lookup target, and using secret security checks to find and remove malicious nodes.

It uses two mechanisms to secure the random walk: First, *bound checking* like NISAN [56] is used, and the second one identifies malicious nodes that manipulate finger tables (FT) and remove them from the network. A combination of them provides strong security and keeps the scalability.

Also, to avoid the range estimation attack [], Octopus introduced two features which aim to preserve the lookup anonymity. They are: *Multiple anonymous paths* in the lookup and *add dummy queries* in the lookup to blur the adversary's observation.

Against finger table manipulation, Octopus introduces a novel security mechanism called *secret neighbour surveillance*, that allows nodes to monitor the possible manipulation in their successor lists by a malicious node. In order to do so, each node maintains a predecessor and a successor list, both with the same size, so that a node *n* is contained in the successor list of any predecessors. After a short period of time, *n* sends randomly an anonymous lookup query to a random predecessor *p* through an anonymous path and check if it is included in the returned successor list. Because of *p* cannot see the source of the query, it cannot distinguish the testing query from the real look up queries and therefore, if *p* tries to bias lookup, n will detect it and report it to the CA. Then each routing table needs to be signed and attached a time stamp by its owner. An alternative method would be to to get the successor lists signed by their owner and keeps a queue of latest received successor lists in each node as proof, to verify that the successor list is not intentionally manipulated.

Wang et al [79] ran some tests over Octopus in order to evaluate its performance. The results show a success in the detection of malicious nodes. It also manages to preserve privacy in networks with 20% of malicious nodes, only leaking 0.57 bits of initiator information and 0.82 of target information. They also analysed some security issues and proved that Octopus is robust against end-to-end timing analysis attack only leaking less than a 5% of information where the network contains a 20% of malicious nodes; that a Selective DoS attack can be constrained by identifying malicious droppers; and that it is totally resistant to a relay exhaustion attack. This scheme is very recent and because of this there is not much information about it. Results are promising and the scheme presents novel mechanisms to deal with attacks that are effective in DHT lookups.

5. SYSTEMS COMPARISON

After having studied all these different systems, some of the schemes have proved to be stronger than others not only in the lookup mechanism but also in how they keep their anonymity. An analysis of some characteristic present in the lookup/search mechanisms previously studied will be analysed below:

A. Scalability

The first aspect that comes in mind is how peer-to-peer anonymity systems are more scalable than the centralized ones. This is not a surprise, since it was a known problem. However, as a ray of hope, PIR-Tor [52], a client-server structure is capable of achieving a promising scalability while keeping it robustness. Yet any P2P system will be able to provide the same degree of scalability.

B. Censorship-resistance:

The systems for censorship-resistance need to provide censorship at storage level. Their task is to keep the contents within a node free from undesirable presences. If these mechanisms detect a malicious presence, block their content so it cannot be deleted or modified by attackers. While Free Haven [21] uses pseudonyms to hide the servers and onion routing, Freenet [13] provides anonymity by using probabilistic routing. Endsuleit and Mie [26] presented promising results by using Castro et al's secure lookup, however, there have not been any recent improvements and it seems is not being in use. On the other hand, Achord [33] that uses Chord lookup, is scalable and able to keep the user anonymity, nevertheless, it is subject to correlation attacks.

C. Unobservability

In this paper we have only found two systems capable to achieve unobservability in the communication. They are *p5* [69] and Herbivore [29]. While *p5* utilises broadcast hierarchy as a binary tree and a noise mechanisms to cover the traffic, Hervibore routes messages between *cliques* over a structured overlay and computational puzzles to prevent nodes from joining arbitrary cliques. P^5 trades-off anonymity for communication efficiency, this way it can offer different levels of anonymity depending on the environment. Hervibore, on the other hand uses 128-bit keys in each clique which increases the chances that a dishonest node appears in any given clique, giving an adversary the chance to launch a DoS or Sybil attack.

D. Security

The security in the lookups/searching mechanisms is the main aspect to take into account when designing a new scheme. The amount of different attacks in the literature makes it almost impossible for a system to be strong against them all. Dos and timing attacks are the ones that our systems struggle from more. While most of the designs in P2P-based topologies do not include any defence mechanism against Sybil, instead, they expect an external mechanism to deal with it.

E. Anonymity

After studying all these anonymity systems, it can be easily seen that all the newer designs are focused in creating secure and anonymous random path creation and anonymous random lookup P2P. The reason to this concentration could be due to the fact that these systems are more scalable than centralized schemes, and therefore there would be more room for larger anonymity sets, allowing nodes and users to "hide". On the other hand, the larger the system, the higher the churn will be, so systems will have to introduce strong mechanisms to deal with this issue.

Several techniques have been presented for preserving anonymity. Freenet [13] protects the communication channel, instead of anonymize it. Onion routing [30] and Tor [22] use a recursive layer data structure –onion- where each layer is encrypted with the public key of the nodes it was routed through. This way, the secrecy of the path is preserved. PIR-Tor [52] has improved Tor by introducing Private Information Retrieval (PIR) to its design. Also, it is able to obtain random relays both securely and anonymously regardless of the fraction of compromised nodes in the network. Tarzan [27] uses layered encryption and multi-hop routing. While Crowds [61] hides nodes in groups and they are controlled by a trusted server that knows all nodes in its group set. Agyaat [66] introduce clouds as a way to hide the nodes identity and does not use a CA. p^5 [69] architecture is prepared to trade efficiency and anonymity degree.

Mittal and Borisov [49] studied AP3 [47] and Salsa [53] and found that both had significant leaks in the lookups that compromise the route-selection security by an adversary composed of a much smaller fraction of the total network than had previously been thought.

Some of the newer mechanisms are NISAN [56] and Torsk [46]. NISAN uses redundancy to improve security and aggregates greedy search to reduce information leakage. It also retrieved the entire finger table to protect the anonymity of the lookup targets. However, it can only offer very limited anonymity protection because a passive adversary is still able to analyse the locations of the queries. In [79], it shows that NISAN does not preserve the initiator anonymity. The same problem is found in Torsk [46], a range estimation attack easily obtains information about the lookup targets.

Wang et al. [79] presents Octopus, a really new anonymous and secure DHT lookup which introduces a mechanism to find and remove malicious nodes from the system. It is also resistant to many different attacks.

Out of all of these proposals, PIR-Tor, NISAN and Octopus are the most outstanding. The three of them are interesting schemes that have improved previous work by adding strong mechanisms to preserve security and privacy. Even though they still have goals to achieve.

6. CONCLUSIONS

This paper has analysed both centralized and peer-to-peer anonymous systems and they have been categorized by their architecture, client-server and peer-to-peer. Also, a study over the degree of anonymity of the system or scheme was completed, where the systems were grouped by their characteristics according to if they were low-latency, unobservable or censorship resistance. Also, a deeper analysis on the peer-to-peer communications was provided, where the systems were sectioned in scalable, secure, random path-based and random lookup-based.

After this study, it can be said that P2P architectures provide a better degree of anonymity than traditional ones. However, they leak much more due to the redundant lookup. They both have serious issues against global attackers, but also with timing attacks, DoS and Sybil attacks. In order to provide a secure and anonymous design, there are still too many issues to solve. Although newer schemes are working in the right direction, one step at a time.

7. REFERENCES

[1] ANDROUTSELLIS-THEOTOKIS, S. AND SPINELLIS, D. 2004. A survey of peer-to-peer content distribution technologies. ACM Comput. Surv., vol. 36, no. 4, pp. 335–371, 2004.

[2] BACKES, M., GOLDBERG, I., KATE, A., AND TOFT, T. 2011. Adding query privacy to robust DHTs. Tech. rep., arXiv:1107.1072v1 [cs.CR], July 2011.

[3] BAUER, K., MCCOY, D., GRUNWALD, D., KOHNO, T., and SICKER, D. 2007. Low-resource routing attacks against anonymous systems. Technical Report CU-CS-1025-07, University of Colorado at Boulder.

[4] BAUMGART, I. AND MIES, S. 2007. S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing. In Proc. 13th International Conference on Parallel and Distributed Systems (Hsinchu, Taiwan). IEEE [51]Computer Society Press, Los Alamitos, CA., 2:1–8.

[5] BORISOV, N. 2005. Anonymous routing in structured peer-to-peer overlays. Doctoral Dissertation. University of California, Berkeley, CA.

[6] BORISOV, N., DANEZIS, G., MITTAL, P. AND TABRIZ, P. 2007. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In ACM Conference on Computer and Communications Security, Oct. 2007

[7] CASTRO, M., DRUSCHEL, P., HU, Y.C., AND ROWSTRON, A. 2002. Exploiting network proximity in distributed hash tables. In International Workshop on Future Directions in Distributed Computing (FuDiCo) (June 2002), O. BABAOGLU, K. BIRMAN, AND K. MARZULLO, Eds., 52-55.

[8] CASTRO, M., DRUSCHEL, P., GANESH, A., ROWSTRON, A. AND WALLACH, D.S. 2002. Secure Routing for structured peer-to-peer overlay networks. In OSDI, December 2002.

[9] CHAUM, D.L. Untraceable electronic mail, return addresses and digital pseudonyms.Communications of the ACM, 84-88. 1981.

[11] CHOLEZ, T., CHRISMENT, I., AND FESTOR, O. 2010. Efficient DHT attack mitigation through peers' ID distribution. In HotP2P 2010.

[12] CHONOV, H. Exploiting Anonymity in P2P-based Systems. 2011. Seminar P2P im Wintersemester 2010.

[13] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability (Berkeley, California, June 2000).

[14] CONDIE, T., KACHOLIA, V.,SANKARARAMAN, S.,HELLERSTEIN, J. AND MANIATIS, P. Induced churn as shelter from routingtable poisoning. In Proc. 13th Annual Network and Distributed System Security Symposium (NDSS), 2006.

[15] DAI, W. 1998. Pipenet 1.0. Post to Cypherpunks mailing list.

[16] DANEZIS, G. AND CLAYTON, R. Route Fingerprinting in Anonymous Communications. Proceedings of the Sixth
IEEE International Conference on Peer-to-Peer Computing, pages 69–72, 2006.

[17] DANEZIS, G., DIAZ, C., KASPER, E. AND TRONCOSO, C. The wisdom of Crowds: Attacks and optimal constructions. In Proceedings of the 14th European Symposium on Computer Security (ESORICS 2009), St Malo, France, volume 5789 of Lecture Notes in Computer Science, pages 406–423, September 2009.

[18] DANEZIS, G., LESNIEWSKI-LAAS, C., KAASHOEK, M.F. AND ANDERSON, R. 2005. Sybil-Resistant DHT Routing. In European Symposium On Research In Computer Security, 2005.

[19] DATTA, A., GIRDZIJAUSKAS, S. AND ABERER, K. On de Bruijn routing in distributed hash tables: there and back again, Proc. Fourth IEEE Int'l Conf. on Peer-to-Peer Computing, , 25-27 August 2004.

[20] DE BRUIJN, N. A combinatorial problem. In Proc. Koninklijke Nederlandse Akademie van Wetenschappen (1946), vol.49, pp.758–764. 1946.

[21] DINGLEDINE, R., FREEDMAN, M. J. AND MOLNAR, D. 2000. The Free Haven project: Distributed anonymous storage service. In Proc. Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, July 2000.

[22] DINGLEDINE, R., MATHEWSON, N. AND SYVERSON, P. 2004. Tor: The second-generation onion router. In Proceedings of the 13th USENIX Security Symposium, August 2004.

[23] DOUCEUR, J. 2002. The Sybil attack. In First IPTPS, March 2002.

[24] EDMAN, M. AND YENER, B. 2009. On anonymity in an electronic society: A survey of anonymous communication systems. ACM Computing Surveys (CSUR), Volume 42 Issue 1, 35 Pages, December 2009.

[25] EL-ANSARY, S.AND HARIDI, S. 2005. An Overview of structured P2P Overlay Networks. In: Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks, CRC Press, 2005.

[26] ENDSULEIT, R. AND MIE, T. 2006. Censorship-Resistant and Anonymous P2P Filesharing. In Int. Conf. on Availability, Reliability and Security (ARES), Vienna, Austria, April 2006.

[27] FREEDMAN, M.J. AND MORRIS, R. 2002. Tarzan: A peer-to-peer anonymizing network layer. In 9th ACM Conference on Computer and Communications Security (CCS 2002), Washington, DC, November 2002

[28] GHEORGHE, G., LO CIGNO, R. AND MONTRESOR, A. 2010. Security and privacy issues in P2P streaming systems: A survey. Peer-to-Peer Networking and Applications (on-line 23 April 2010), Springer.

[29] GOEL, S., ROBSON, M., POLTE, M. AND SIRER, E.G.2003. Herbivore: A scalable and efficient protocol for anonymous communication. Technical Report TR2003-1890, Cornell University Computing and Information Science, February 2003.

[30] GOLDSCHLAG, D., REED, M. AND SYVERSON, P. 1999. Onion routing for anonymous and private Internet connections. Communications of the ACM 42(2). 39-41.

[31] LIU, Y., HAN, J., AND WANG, J. Rumor Riding: Anonymizing unstructured peer-to-peer systems. IEEE Transactions on Parallel and Distributed Systems. Volume 22, Issue 3. Pages 464-475. March 2011.

[32] Harvey, N. J. A., Jones, M. B., Saroiu, S., Theimer, M., and Wolman, A. Skipnet: A scalable overlay network with practical locality properties. In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems -Seattle, WA.2003.

[33] HAZEL, S. AND WILEY, B. Achord: A Variant of the Chord Lookup Service for Use in Censorship Resistant Peer-to-Peer Publishing Systems. In Proceedings of the First Workshop on Peer-to-Peer Systems, Cambridge, MA, 2002.

[34] HOPPER, N., VASSERMAN, E. Y., AND CHAN-TIN, E. 2007. How much anonymity does network latency leak? In Proceedings of CCS. 2007.

[35] KAASHOEK, F. AND KARGER, D. Koorde: A Simple Degree-Optimal Hash Table. Proc. 2nd Int'l. Wksp. Peer-to- Peer Systems (IPTPS'03), Berkeley, CA, USA, Feb. 20–21, 2003.

[36] KAPADIA, A. AND TRIANDOPOULOS, N. 2008. Halo: High-assurance locate for distributed hash tables. In C. Cowan and G. Vigna, editors, Network and Distributed System Security Symposium, pages 61–79, Feb. 2008.

[37] KHAN, S. M., MALLESH, N., NAMBIAR, A. AND WRIGHT, M. The Dynamics of Salsa: A Robust Structured P2P System. Network Protocols and Algorithms, ISSN 1943-3581, Vol. 2, No 4. 2010.

[38] KONDO, M., SAITO, S., ISHIGURO, K., TANAKA, H. AND MATSUO, H. 2009. Bifrost: A novel anonymous communication system with DHT. In Proceedings of PDCAT '09, 2009, pp. 324–329

[39] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An architecture for global-scale persistent storage. In Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000) (November 2000) pp. 190–201.

[40] LEVINE, B.N., SHIELDS, C. AND MARGOLIN, N.B. 2006. A survey of solutions to the Sybil attack. Technical Report 2006-052, Univ. of Massachussets Amherst, 2006.

[41] LI, J. 2007. *A Survey of Peer-to-Peer Network Security Issues.* Retrieved November 29, 2010 from http://www.cse.wustl.edu/~jain/.

[42] LUA, E.K., CROWCROFT, J., PIAS, M., SHARMA, R. AND LIM, S. 2005. A survey and comparison of Peer-to-Peer Overlay network schemes. IEEE Communications Surveys and Tutorials 7 (2005), 72–93.

[43] MALKHI, D., NAOR, M., AND RATAJCZAK, D. Viceroy: A scalable and dynamic emulation of the butterfly. In Proceedings of ACM Principles of Distributed Computing (PODC) Monterey, CA (July 2002).

[44] MANILS, P., CHAABANE, A., BLOND S.L., KAAFAR, M.A., CASTELLUCCIA, C., LEGOUT, A. AND DABBOUS, W. 2010. Compromising Tor Anonymity Exploiting P2P Information Leakage. Technical report, INRIA, 2010.

[45] MAYMOUNKOV, P. AND MAZIERES, D. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In Proceedings of IPTPS02, Cambridge, USA, Mar. 2002.

[46] MCLACHLAN, J., TRAN, A., HOPPER, N. AND KIM, Y. 2009. Scalable onion routing with Torsk. ACM CCS, November 2009.

[47] MISLOVE, A., OBEROI, G., POST, A., REIS, C., DRUSCHEL, P. AND WALLACH, D.S. 2004. AP3: Anonymization of Group Communication. In ACM SIGOPS European Workshop, Sept. 2004

[48] MITALL, P. 2010. A security evaluation of the Salsa anonymous communication system. Thesis. University of Illinois at Urbana-Champaign.

[49] MITTAL, P. AND BORISOV, N. 2008. Information leaks in structured peer-to-peer anonymous communication systems. In CCS'08: Proceedings of the 15th ACM conference on Computer and Communications Security (New York, NY, USA, 2008), ACM, pp. 267–278.

[50] MITTAL, P. AND BORISOV, N. 2009. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. ACM CCS (2009).

[51] MITTAL, P., BORISOV, N., TRONCOSO, C. AND RIAL, A. 2010. Scalable anonymous communication with provable security. In 5th USENIX Workshop on Hot Topics in Security (HotSec 2010), page 7. USENIX, 2010.

[52] MITTAL, P., OLUMON, F., TRONCOSO, C., BORISOV, N. AND GOLDBERG, I. 2011. PIR-Tor: Scalable anonymous communication using private information retrieval. In Proceedings of the 20th USENIX Security Symposium, San Diego, CA, August 2011.

[53] NAMBIAR, A. AND WRIGHT, M. 2006. Salsa: A structured approach to large-scale anonymity. ACM CCS, 2006.

[54] NEEDELS, K. AND KWON, M. Secure Routing in Peer-to-Peer Distributed Hash Tables[C]//Proc. of ACM Symposium on Applied Computing. Hawaii, USA: [s. n.], 2009: 54-58.

[55] O'DONNELL, C. W. AND VAIKUNTANATHAN, V. Information Leak in the Chord Lookup Protocol. In P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing, 2004.

[56] PANCHENKO, A., RICHTER, S. AND RACHE, A. 2009. NISAN: Network Information Service for Anonymization Networks. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, ACM Conference on Computer and Communications Security (CCS 2009), pages 141–150. ACM, 2009.

[57] PLAXTON, C.G., RAJARAMAN, R. AND RICHA, A.W. Accessing nearby copies of replicated objects in a distributed environment. Theory of Computing Systems, 32:241–280, 1999.

[58] PUTTASWAMY, K., SALA, A., WILSON, C., AND ZHAO,B. Protecting anonymity in dynamic peer-to-peer networks. In IEEE International Conference on Network Protocols (ICNP) (Oct. 2008), pp. 104–113. 2008.

[59] RABIN, M.O. Efficient dispersal of information for security, load balancing and fault tolerance. Journal of the ACM, 36(2): 335-348. 1989.

[60] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R. AND SHENKER, S. 2001. A scalable content-addressable network. In Proc. ACM SIGCOMM (San Diego,CA, August 2001), pp. 161–172.

[61] REICHTER, M., RUBIN, A. 1997. Crowds: Anonymity for Web Transactions (preliminary announcement), DIMACS Technical Reports 97-15, April 1997.

[62] REN, J. AND WU, J. 2010. Survey on Anonymous Communications in Computer Networks. Comput. Commun. (2010), Vol. 33, No. 4 pp. 420-431.

[63] RHEA, S., GEELS, D., ROSCOE, T. AND KUBIATOWICZ, J. Handling churn in a DHT. In *USENIX Annual Tech. Conf.*, June 2004.

[64] ROWSTRON, A. AND DRUSCHEL, P. 2001. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001) (Nov. 2001).

[65] SARMADY, S. 2007. A survey on Peer-to-Peer and DHT. Grid Lab, School of Computer Science, University Sains Malaysia, Penang, Malaysia, 2007.

[66] SINGH, A., GEDIK, B. AND LIU, L. 2004. Agyaat: Providing Mutually Anonymous Services over Structured P2P Networks. Technical Report GIT-CERCS-04-12, Georgia Inst. Of Tech. CERCS, 2004.Agyaat: mutual anonymity over structured P2P networks. Internet Research, 16(2), 189-212.

[67] SINGH, A., CASTRO, M. DRUSCHEL, P. AND ROWSTRON, A. 2004. Defending against Eclipse attacks on overlay networks. Proc . SIGOPS European Wksp., Leuven, Belgium, Sept. 2004.

[68] SCHUCHARD, M., DEAN, A., HEORHIADI, V., HOPPER, N. AND KIM, Y. 2010. Balancing the shadows. ACM WPES, 2010.

[69] SHERWOOD, R., BHATTACHARJEE, B. AND SRINIVASAN, A. 2002. p5: A protocol for scalable anonymous communication. In IEEE Symposium on Security and Privacy, pages 58–70. IEEE CS, 2002.

[70] STOICA, I., MORRIS, R., LIBEN-NOVELL, D., KARGER, D., KAASHOEK, M.F., DABEK, F., BALAKRISHNAN, H. 2001. Chord: A Scalable Peer-to-peer Loopup Service for internet applications. In Proceedings of SIGCOMM 2001, August 2001.

[71] SUNG, L.G.A., AHMED, N., BLANCO, R., LI, H., SOLIMAN, M.A. AND HADALLER, D. 2005. A

survey of data management in peer-to-peer systems. Web Data Management, Winter 2005, pp.1–50 (2005).

[72] SYVERSON, P., TSUDIK, G., REED, M. AND LANDWEHR C. 2000. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[73] TRAN, A., HOPPER, N. AND KIM, Y. 2009. Hashing it out in public: Common failure modes of DHT-based anonymity schemes. In Proc. of WPES, 2009.

[74] URDANETA, G., PIERRE, G. AND STEEN, M.V. 2011. A survey of DHT security techniques. ACMComputing Surveys (CSUR) , Volume 43 Issue 2, January 2011.

[75] VASSERMAN, E.Y. 2010. Towards freedom of speech on the Internet: Censorship-resistant communication and storage. Doctoral dissertation. University of Minnesota.

[76] WANG, L. 2006. Attacks against Peer-to-peer Networks and Countermeasures. Technical report, Helsinki University of Technology, December 2006.

[77] WANG, P., OSIPKOV, I., HOPPER, N. AND KIM, Y. 2007. Myrmic: Secure and Robust DHT Routing. Tech. rep.,Digital Technology Center, University of Minnesota at Twin Cities.

[78] WANG, Q., MITTAL, P. AND BORISOV, N. 2010. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. E. AL-SHAER, A.D. KEROMYTIS AND V. SHMATIKOV, editors, ACM Conference on Computer and Communications Security (CCS 2010), pages 308-318. ACM, 2010.

[79] WANG, Q. AND BORISOV, N. 2011. Octopus: Anonymous and secure DHT lookup. CoRR, March 2012.

[80] WESTERMANN, B., PANCHENKO,A. AND PIMENIDIS, L. A kademlia-based node lookup system for anonymization networks. In Advances in Information Security and Assurance: Proceedings of the Third International Conference on Information Security and Assurance (ISA 2009), volume 5576 of LNCS, pages 179–189, Seoul, South Korea, Jun 2009. Springer.

[81] WRIGHT, M.K., ADLER, M., LEVINE, B.N. AND SHIELDS, C. 2004. The predecessor attack: An analysis of a threat to anonymous communications systems. ACM Trans. Inf. Syst. Secur., 7(4):489–522, 2004.

[82] ZHAO, B., KUBIATOWICZ, J.AND JOSEPH, A."Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Comput. Sci. Div., Univ. California, Berkeley, Tech. Rep. UCB/CSD-01-1141, 2001.

[83] ZHUANG, L., ZHOU, F., ZHAO, B.Y. AND ROWSTRON, A. 2005. Cashmere: resilient anonymous routing. In NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (Berkeley, CA, USA, 2005), USENIX Association, pp. 301–314.