The current issue and full text archive of this journal is available at
www.emeraldinsight.com/1066-2243.htm

Agyaat: mutual anonymity over structured P2P networks

Aameek Singh, Bugra Gedik and Ling Liu
College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA

Abstract

Purpose – To provide mutual anonymity over traditionally un-anonymous Distributed Hash Tables (DHT) based Peer-to-Peer overlay networks, while maintaining the desired scalability and guaranteed lookup properties of the DHTs.

Design/methodology/approach – Agyaat uses a novel hybrid-overlay design, a fully decentralized topology without any trusted proxies. It anonymizes both the querying and responding peers through the use of unstructured topologies, called *clouds*, which are added *onto* the structured overlays. In addition, it regulates the cloud topologies to ensure the guaranteed location of data and scalability of routing. A unique characteristic of the design is the ability of users to tradeoff between desired anonymity and performance. The paper presents a thorough performance and anonymity analysis of the system, and also analyzes few anonymity compromising attacks and countermeasures.

Findings – The results indicate that Agyaat is able to provide mutual anonymity while maintaining the scalability of lookups, affecting the costs only by a constant factor.

Research limitations/implications – While Agyaat is able to meet its mutual anonymity and performance goals, there exist other security vulnerabilities like possible Denial-of-Service (DoS) attacks, both due to its design and the underlying DHT overlay. This is fertile ground for future work.

Originality/value – Agyaat uses a novel topology architecture and associated protocols that are conducive to providing mutually anonymous services.

Keywords Computer networks, Privacy

Paper type Research paper

1. Introduction

Mutual anonymity refers to the property of peers being unable to identify the origin (initiator) or destination (responder) of a message. While most unstructured P2P systems like Gnutella (Gnutella, 2002), Kazaa (Kazaa, 2002) provide anonymity through the use of a random overlay topology and a flooding based routing protocol, they suffer from the lack of guaranteed lookup of data. In contrast, new generation structured P2P systems like Chord (Stoica *et al.*, 2001), Pastry (Rowstron and Druschel, 2001), CAN (Ratnasamy *et al.*, 2001), Tapestry (Zhao *et al.*, 2004) are Distributed Hash Table (DHT) based systems and provide guarantees that any stored data item can be found within a bounded number of hops. However, none of the DHT based P2P systems provide mutual anonymity.

This work is partially supported by the National Science Foundation under a CNS Grant, an ITR grant, a Research Infrastructure grant, and a DoE SciDAC grant, an IBM SUR grant, an IBM faculty award, and an HP equipment grant. Any opinions, findings, and conclusions or recommend actions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or DoE. The first author is also supported by an IBM PhD fellowship.

Internet Research
Vol. 16 No. 2, 2006
pp. 189-212

1066-2243
DOI 10.1108/10662240610656519

In this paper, we present Agyaat[1] – a decentralized P2P system, which promotes a generic non-cryptographic solution for mutual anonymity. Agyaat combines the best characteristics of both unstructured and structured P2P systems by providing mutually anonymous services, while maintaining the scalability and efficiency of DHT routing schemes. Compared with existing pure DHT based systems, the routing performance of Agyaat differs only by constants in terms of both the number of hops and the aggregate messaging costs. Agyaat provides mutual anonymity by adding unstructured cloud topologies onto structured DHT overlays. It breaks the standard data-to-peer DHT mapping into two steps and utilizes an important feature of local query termination within cloud topologies to facilitate mutual anonymity. Agyaat introduces a number of other mechanisms:

- to regulate the cloud topology;
- to increase efficiency of routing both between and within Agyaat clouds; and
- to ensure the guaranteed location of data.

As an interesting by-product, Agyaat allows for better management of data since its add-on cloud topologies provide greater data locality and can be used to support semantic grouping (Crespo and Garcia-Molina, 2002), a feature found lacking in DHT based systems (Section 6).

We validate Agyaat through two steps of analysis. First, we report a set of experiments conducted to analyze the system performance, comparing it with pure DHT-based systems. Second, as part of our security analysis, we discuss anonymity-compromising attacks under a passive logging model and possible defenses against these attacks.

It is important to emphasize that Agyaat addresses the problem of anonymizing the DHT lookup – the query message from a peer (service-requester) to the peer responsible for the requested data item (service-provider). This is unlike the process of anonymizing data transfer between two peers, for which, existing solutions (Xiao *et al.*, 2003, Clarke *et al.*, 2001) can be directly applied. As we discuss in Section-2.1, pure-DHT based systems, by their basic design, are in-conducive to supporting mutual anonymity during the lookup phase – the issue we solve using Agyaat.

2. P2P and mutual anonymity

This section defines the problem of mutual anonymity in the context of DHT based systems. We compare the DHT overlays with Gnutella-like random topologies and explore a topology feature, that plays a pivotal role in facilitating mutual anonymity. We use it as our design principle. Note that while we use Chord (Stoica *et al.*, 2001) as our example DHT based system, Agyaat is a generic solution which can be equally applied to other DHT based systems like Pastry (Rowstron and Druschel, 2001).

2.1 Mutual anonymity

To understand the basic properties of mutual anonymity in the context of P2P systems, we first examine unstructured P2P systems and then discuss the challenges for providing mutual anonymity in DHT based systems. The unstructured P2P systems provide mutual anonymity by cloaking the exact origin and termination of a query. The peers in these systems have local knowledge of the network, knowing only their immediate neighbors and follow a simple message forwarding mechanism. Whenever

Peer-A receives a message from Peer-B, the local-knowledge property ensures that Peer-A only knows that it came from Peer-B and nothing about any peer in the query path before Peer-B. As a result, nothing conclusive can be said of the origin of a particular message. Similarly, when a peer, say Peer-R, wishes to reply to a query, it just sends a reply to the neighbor (Peer-N), from which it received the query. The local-knowledge property prevents Peer-N to conclude that the reply was initiated by Peer-R[2], since Peer-R could just be forwarding the reply initiated farther in the network. Peer-N, then forwards the reply to its corresponding neighbor. This process continues till the query reply reaches the querying peer. Again, nothing conclusive can be said about where the reply terminated. Hence, mutual anonymity is assured.

However, unstructured P2P systems have some lookup drawbacks. Since a query cannot be kept active in the network for an infinite period of time, it is typically prematurely terminated after a certain number of application level network hops (called Time-To-Live, typically seven). As a result, there is no guaranteed lookup of data. Also, due to the broadcast of queries, the resource usage is large. This led to the emergence of a class of P2P systems that include Chord (Stoica *et al.*, 2001), CAN (Ratnasamy *et al.*, 2001) and Pastry (Rowstron and Druschel, 2001). In the context of DHT based systems, the problem of mutual anonymity reduces to protecting the identities of the peer issuing the query (specified by a key) and the peer responsible for that particular key. This is challenging because it contradicts the basic DHT routing-table based lookup mechanism; each peer has a routing table containing a set of peers responsible for certain keys, and each step in the lookup process brings the query closer to the destination peer. This ability to reach the peer responsible for a key by combining information from routing tables of various peers, is in contrast to the goal of mutual anonymity, which demands that it should not be possible to identify a peer responsible for any query item. As a result, any mechanism that ensures mutual anonymity would need to counter this basic DHT routing property. Furthermore, support for mutual anonymity should preserve levels of scalability, and continue to guarantee the location of data within bounded number of hops.

In the next subsection, we first look at how a DHT-based system, Chord, performs a lookup and discuss the desired properties of mutual anonymity over Chord.

2.2 Local termination and initiation

There are two ways in which a Chord lookup can proceed – *iterative* and *recursive* (Stoica *et al.*, 2001). In iterative Chord, at each hop, the querying node gets the address of the next hop node and is itself responsible for forwarding the query. This clearly takes away the anonymity of the querying peer. Also, since it would know the peer that finally responds to the query, the service provider is not anonymous either. On the other hand, in recursive Chord, the node in the lookup path forwards the query, much like the message forwarding mechanism of Gnutella and a reply can be traced back using the same path. This, in a first glance, looks to be sufficient for maintaining mutual anonymity. However, we show that it fails to ensure service provider's anonymity.

Note that the termination condition in original Chord algorithm (as proposed in (Stoica *et al.*, 2001)) is that if a node finds out that the query item hashes onto a region between itself and its immediate successor, then the successor is responsible for that item. Clearly, this fails to provide anonymity to the service provider, since its

predecessor would know where the query forwarding terminates. One may immediately think that a simple revision of the Chord protocol can make it more anonymous. Instead of letting the predecessor of a service provider peer to decide the termination of a lookup operation, the protocol can be revised to let the service provider terminate the lookup. This can be done by allocating the region between two peers to the predecessor, which is the one before the other on the identifier ring (following the clockwise order). Note that changing the termination condition implies making changes to the routing table as well. Now each entry would point to the node immediately before the query item on the ring.

However, even this revised scheme does not work. Consider the ring as shown in Figure 1. Based on the routing table for the node N48, it is responsible for query items 49 and 50 (modified termination condition). Even if we neglect the deterministic location of key value 51 (which will be on N51), there exist many keys which can be deterministically linked to the responsible peers. For instance, node N48 knows that node N51 is responsible for keys 52, 53, 54, 55 and 56, because it's routing table indicates that there does not exist any node between N51 and N57 (else N48 + 8 entry would have pointed to it). As a result, the revised scheme still does not anonymize the service provider.

There is one crucial insight learnt from the above analysis. Notice that when we changed the termination condition, we made certain termination scenarios local to the responding peer (i.e. the peer which is responsible for that query item). This made the system more anonymous. In fact the successful strategy for providing mutual anonymity is that such termination should always be local to the responding peer. Local termination of queries ensures that no other peer knows where the query terminates. Also, when the responding peer initiates a reply, peers in the path cannot ascertain its origin (since the peer could just be forwarding a message received from some other peer – local initiation). Any topology that possesses these local termination and local initiation properties will be conducive to mutual anonymity. We call such a topology an LTI topology (Local Termination and Initiation preserving topology). While it is relatively easy to provide local initiation in DHT based systems (e.g. recursive Chord), local termination is a complicated task. It is easy to see that Gnutella is an LTI topology. Only the query issuer knows where the query originates and only the responding peer knows where it terminates. Also note that the reply is symmetrical to the query, except that the message initiates at the responding peer and terminates at the querying peer.

This insight serves as an important motivation and design principle for the development of Agyaat. We add-on LTI topologies (clouds) over the underlying structured DHT system to provide mutual anonymity.

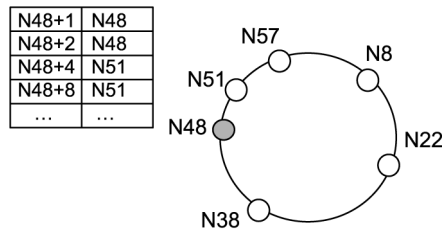

Figure 1.
Recursive Chord does not provide mutual anonymity

3. Agyaat

In Agyaat, we provide mutual anonymity by adding Agyaat clouds (LTI topologies) on top of a DHT-based P2P network. We guarantee local termination/initiation properties by enabling the query to initiate and terminate inside the Agyaat clouds, while using normal DHT-based routing to link service requester's cloud to service provider's cloud. Such a design associates with itself a host of challenging issues, including:

- how to maintain routing properties in spite of two different topologies – the unstructured Agyaat clouds and the underlying DHT overlay;
- how to ensure scalable and guaranteed lookups with routing performance comparable to Chord-like pure DHT systems; and
- how to defend against possible attacks.

We will discuss these challenges and our technical solutions in the following sections and begin with an overview of the complete design.

3.1 Design overview

For a querying peer to be made anonymous, we have to make sure that the origin of the query is not disclosed. As mentioned earlier, this is possible if the query originates from an LTI topology. Similarly if the query terminates in an LTI topology, the service provider anonymity can also be ensured. For anything between these two end points, the query can just proceed as on the DHT ring, which provides guaranteed location of the Agyaat cloud to which the responsible peer of the query item belongs. In this paper, for simplicity in exposition, we will use Gnutella as an example LTI topology. We let peer form small unstructured LTI topologies, which we call "clouds" for their cloaking effect, and initiate/respond to queries only through these clouds. Thus, peers in addition[3] to being a part of the DHT ring, connect to a few other peers (neighbors) in a Gnutella like fashion. This enables the formation of small clouds on top of the DHT ring. Figure 2 shows two clouds with nodes being part of both the cloud and the DHT ring. More precisely, the shaded nodes (A, B and C) form one unstructured cloud and white nodes (X, Y and Z) form another.

In Agyaat, each peer can initiate a query by forwarding the query to peers in its cloud. Then, one of those peers takes the query out of the cloud, onto the main DHT ring and a normal DHT lookup takes over to locate the cloud to which the responding peer belongs. At the responder's end, some peer in the cloud of the responding peer gets

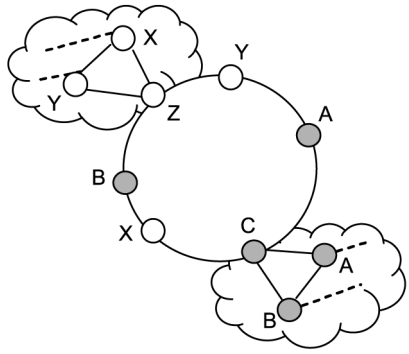

Figure 2.
Add-on clouds over DHTs

the query and broadcasts it in its cloud. The broadcast is required since the identity of the responding peer is not known, and so the message is sent to every peer in the cloud. When the responding peer gets the query, it initiates a reply in the response cloud, which follows a similar path back to the query originator's cloud. In order to make sure that firstly, the cloud does not become too large, that the cost of broadcasting becomes prohibitive and secondly, the query items, if existing will be found in a cloud (guaranteed location of data), we control the size of Agyaat cloud using system parameters like maximum cloud diameter and node degree. We also study the settings of these parameters and their impact on the overall performance of Agyaat (see Section 3.4.2).

At this point, it is important to point out that while normally DHTs map a key to a peer, Agyaat breaks that process into two steps – a key now maps to a cloud (through the DHT routing) and then the cloud links to the appropriate peer (through flooding). Agyaat uses the second step to anonymize the service provider.

3.2 Providing mutual anonymity: clouds

A cloud is a small unstructured network with local termination and initiation properties. Since a key is mapped to a cloud using normal DHT operations, it is essential to represent the clouds on the DHT ring i.e. to find an entry point into the cloud. This is accomplished using the concept of rendezvous nodes, similar to the work done in P2P multicast (Castro *et al.*, 2002, Ratnasamy *et al.*, 2001). Each cloud has a name (a string) and using some hash function, it is hashed onto the DHT ring. The node responsible for that region (according to the normal DHT policy) is found and it acts as an entry point into the cloud. This node is called the rendezvous node and is required to be a member of that cloud. Because of the consistent hashing (Karger *et al.*, 1997) properties of the DHT based systems, the load on a node due to its rendezvous properties will be approximately equally distributed amongst all the nodes. Also, in case some rendezvous node leaves, a new one is found by virtue of the dynamics handling of the DHT protocols and it simply replaces the old one in the cloud. Note that given a cloud name, it is always possible to find its rendezvous nodes, by just doing a DHT lookup for its hashed name. This DHT property prevents rendezvous nodes from becoming single points of failures.

In order to create a cloud, the desired name for the cloud is hashed onto the ring using multiple hash functions and thus multiple rendezvous nodes are found. These nodes connect to each other to form a small Gnutella network. The number of hash functions used depends upon the desired initial membership of a cloud and can be set as a system parameter[4]. For example, for two hash functions, h_m and h_n, we will select two nodes which will serve as rendezvous nodes for that cloud. The rendezvous node which was selected when h_m was used to hash the cloud name on the ring is called *RN*-h_m. Any node wishing to join a cloud uses any one of the hash functions to get to a rendezvous node which then bootstraps it into the cloud. This is similar to the bootstrapping process of Gnutella-like systems, in which nodes give out IP addresses of other recent members of the cloud and they are contacted for any open slots. A willing member would accept the incoming peer as a neighbor and make it a member of the network (cloud).

To allow peers to get a list of active clouds, each peer caches the cloud names it ever sees (e.g. when it is in the lookup path of a query on the DHT). This list is shared with an incoming peer, so that it can choose a cloud to join.

An important question still remains. How can we link the service provided by any member of the cloud to that cloud? It is essential since we need a mechanism ensuring that a query for a service reaches the cloud of the service provider. In other words, for a key k, how do we find the cloud which contains the peer responsible for k? This is the first step of our data-to-peer mapping scheme and can be a complicated task depending upon the kind of services being supported by the system. We have classified the types of services into three categories:

(1) *Semantic groups*. This is a class of services in which clouds are formed in a semantic manner, i.e. the services being offered by peers in a cloud are semantically linked to each other. For example, for a file sharing application, the peers belonging to a same cloud could be sharing music from a single artist. The artist's name is used as the cloud name and queries for its songs are tagged with the cloud name. The DHT lookup will lead to the rendezvous node for that cloud and the query is forwarded to it. Although one can infer the *type* of services provided by a peer from its cloud membership, specific instances of services are still anonymized. In addition, there are various means by which we can cloak the *type* by using a single cloud for multiple semantic categories or by a peer being a member of multiple clouds.

(2) *Services with discovery mechanisms*. This is a class of services which have an existing discovery mechanism, for example, a directory service, that can be leveraged to link a query to an appropriate cloud. This category is actually a generalization of the semantic groups category, in which the discovery was due to the semantic nature of the services being offered.

(3) *Dynamic services*. This is a class of services where there does not exist any discovery mechanisms or it is infeasible to use one. As an example, consider a performance critical application like a decentralized web crawler (Singh *et al.* 2004), where peers distribute web crawling tasks by using URLs as DHT keys and assigning responsibility to peers according to the DHT system policy. It is not feasible to have a directory service due to the prohibitory performance costs. Thus, for a given URL there is no easy way to determine the name of the cloud responsible for crawling that URL. For such services, we use R-Rings (Section 3.3).

It is important to notice that for the first two categories, it is the responsibility of the peer to join an appropriate cloud. This is acceptable, since peers using Agyaat want to offer services and remain anonymous while doing so. For the third category, the peer can join any cloud. However, as we discuss next, R-Rings are in fact, sufficient for all three kinds of services.

3.3 Mapping services to clouds: R-Rings

Let us take a look at mechanisms that can be used to support dynamic services. We again assume the web crawling application, where a cloud is responsible for crawling a certain set of domains. Figure 3 shows an example scenario. Peer-A, B and C belong to Cloud-1 and Peer-X and Y belong to Cloud-2. Let us assume that we assign a query

item (a URL in this case) to a cloud if it hashes onto any of its members. In the figure, URL-p and URL-q hash onto Cloud-1 since they hash on Peer-A and Peer-C respectively.

Now assume that Peer-Z enters the DHT ring at the position shown in Figure 4 and joins Cloud-2. As a result, URL-q will map to Cloud-2 as opposed to Cloud-1, which was initially responsible for it. We would have lost all information about all such URLs (required to prevent duplicate crawling), thus violating the guaranteed lookup principle of DHT systems. Note that keeping forwarding information at Peer-Z will not scale and eventually become unmanageable.

Closer inspection will reveal that this problem occurs since the routing of query items is based on peers, which tend to be very dynamic in nature. On the other hand, a cloud is static and it persists even when existing members leave or new ones join. Therefore, any mechanism in which the routing occurs based on cloud names, will be successful to handle this issue. This leads to the idea of Rendezvous Rings (R-Rings).

An R-Ring is a special DHT ring, consisting of one rendezvous node from each cloud. The idea is to create a smaller ring comprising only of one node per cloud, and route queries on that ring. For this, we select a new hash function, say h_t which will be used in the lookup of queries. Then, we select one rendezvous node from each of the clouds, create a new DHT-based R-Ring, where each selected rendezvous node is placed at the position specified by the h_t hash value of its corresponding cloud name. For example, assume Peer-A and Peer-Y are the RN-h_m rendezvous nodes for Cloud-1 and Cloud-2 (Figure 5). Then we create a new R-Ring consisting of only these two nodes. An important point in the construction is that the position that a rendezvous node occupies on that ring is the h_t-hash of its cloud name as shown in Figure 6. Notice that Peer-A and Peer-Y occupy different positions from their original ones in Figure 5. This is done because a rendezvous node can change for a cloud, but the cloud name remains static,

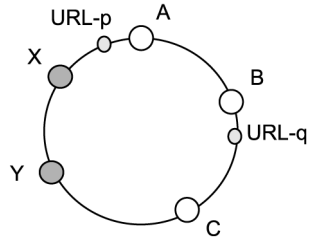

Figure 3.
Dynamic case

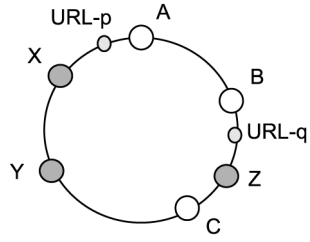

Figure 4.
Peer entry

which is what we desire. So, even when a rendezvous node leaves the new rendezvous node occupies the same position on the ring and consistent routing can be ensured.

The process of creating an R-Ring is similar to that of the main DHT rings and the protocols are well understood. Given such a system, we can route the query items on the R-Rings rather than the main DHT rings. This will always take items to particular clouds and not peers. Any change in memberships of the clouds has no effect on the lookup of items since there always will be a rendezvous node for the cloud at precisely the same position on the R-Ring. As a result, for our example application, URL-q will always map to Cloud-1 even when it maps onto a peer of Cloud-2 on the main DHT ring.

This can lead to significant loads on the rendezvous nodes selected to take a part on the R-Ring. To prevent this, we balance load across various rendezvous nodes of the cloud by constructing an R-Ring for each rendezvous node of the cloud. For example, if we used two hash functions, say h_m, h_n for creating rendezvous nodes, then we will have two R-Rings: one R-Ring consisting of *RN-*h_m and another consisting of *RN-*h_n nodes from all clouds. All rendezvous nodes will occupy the positions specified by h_t(*cloudname*) in each of the R-Rings they belong to, so that all of them yield the same mappings of keys to clouds. Then a query can be routed on any of these R-Rings thus balancing the load between different rendezvous nodes for each cloud.

Notice that the design of R-Ring can work for any kind of services, and not just dynamic services. Even in case of semantic or services with discovery mechanisms, routing on R-Rings would lead to the rendezvous node of the appropriate cloud.

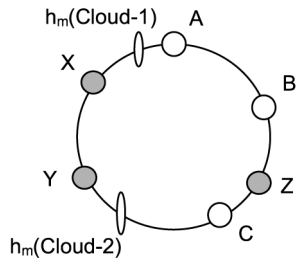

Figure 5.
Main DHT ring

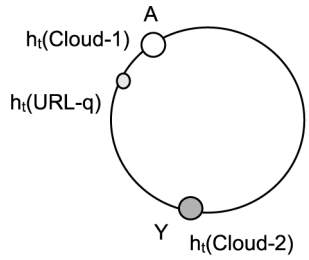

Figure 6.
R-Ring for h_m

3.4 Agyaat protocols

In this section, we discuss the exact protocol with which a query is generated, routed and responded to. We also discuss other issues like size of clouds and the system parameters used to control it.

3.4.1 Query: crossover peers. A query in Agyaat originates in a cloud and is later brought out on the R-Ring and it terminates in another cloud. An important issue in this regard is the selection of the peer which will perform the first crossover from the cloud to the ring. There are a number of important considerations for the crossover:

- It should not be done by the querying peer (or only randomly), else it compromises its anonymity.
- It should be done in such a way that the load is balanced among all members of the cloud.
- It should be done by one peer at a time (except for redundancy purposes) to prevent multiple copies of the query in the system.

We enable this using a random walk in the cloud. The querying peer forms the query message, sets up a random TTL for the message and forwards it to one of its neighbors, selected randomly. The TTL is selected randomly to cloak the origin of the message. The neighbor decrements the TTL by 1 and again selects one of its neighbors randomly and forwards the message to it. Note that, it could not have determined that the message originated at the querying peer by just looking at the message and the TTL. Now after a few hops, the TTL will reduce to zero. The peer, at which that happens, is the crossover peer and is responsible for crossing over and taking the query to the ring. This mechanism ensures that there is only a unique peer performing the query. Secondly, because of the random walk the load will be distributed equally amongst the members of the cloud. Also, similar to Gnutella, all peers in the path of the query will cache the query and remember the neighbor from which they received it. This helps in sending back the reply through the same path.

Next, the query goes out on an R-Ring. The crossover peer might not have its routing table (only the rendezvous nodes have routing tables for R-Rings). In that case, it can query a rendezvous node of its cloud for only the first step of the lookup. The rendezvous node just needs to give the address for the first hop on the R-Ring and the crossover peer can continue the rest of the query in the normal iterative fashion. We call this phase the ring phase. At the end of the ring phase, the crossover peer finds the rendezvous node for the cloud of the service provider. The query is then forwarded to the rendezvous node, which will broadcast the query in its cloud. This broadcast, similar to Gnutella broadcasts, can be an expensive step affecting the scalability of the system. (see Section 3.4.2).

We further expedite the manner in which the reply is traced back to the querying peer. The crossover peer, when forwarding the query to the rendezvous node of the response cloud, includes its IP address and a port number where a reply can directly be sent. Clearly, this does not compromise any anonymity because the crossover peer is usually not the peer initiating the query[5] and many peers in the DHT ring would have seen it performing the query (in the ring phase) anyway. However, this saves us critical time and number of messages, since it shortens the response path by eliminating the intermediate ring phase. The reply is then forwarded back to the querying peer. To prevent loss of replies because of an exit of a peer in the query path,

we can allow multiple random walks either sequentially or concurrently. Also, if the crossover peer exits, the reply can be sent to the rendezvous node of the querying cloud which can then transmit the query into its cloud.

In the response cloud, every peer would get the query message because of the broadcast. The peer wishing to provide the desired service can then form the reply message (with IP address and port information for the querying crossover peer) and start a random walk with a random TTL, similar to the walk used while initiating a query. Because of the random walk, again the load of anonymizing the responding peer is evenly distributed to all members of the cloud and there is a unique peer taking the reply out of the response cloud. However, there is no caching done in the intermediate path and the peer at which TTL reduces to zero just forwards the reply to the crossover peer of the querying cloud. This way the service provider is also anonymized.

It is important to note that Agyaat provides guaranteed lookup of data. For every data item available in Agyaat, a lookup will always succeed. This is because the query is first routed to the appropriate cloud and then the rendezvous node will broadcast it in that cloud, which ensures that every peer in the cloud receives the query. In addition, the protocol provides for customizable per-request levels of anonymity. For example, a node desiring better performance can start with a smaller TTL, whereas a node requiring more anonymity can start with larger TTLs (or use an extended version of the protocol, as discussed in Section 3.4.3).

3.4.2 Size of the Clouds. As mentioned before, since the query is broadcasted in the response cloud, we need to make sure that the size of the cloud does not become too large to make the broadcast costs prohibitive. In order to control the size of the cloud, we use two parameters:

(1) *R-Diameter*. It is the maximum distance of a peer from any rendezvous node. The distance is measured in number of application level hops in the underlying topology. It is denoted by r_{diam} and serves as the length dimension of the cloud. This parameter is important, since any query broadcasted by the rendezvous node will have an upper bound of r_{diam} on the number of hops required to reach any peer of the cloud. Also, the distance from any rendezvous node is used, since the query can be broadcasted by any of the rendezvous nodes (depending upon which R-Ring is used).

(2) *Degree*. It is the maximum number of neighbors a peer can have in the cloud. It is denoted by m and serves as the breadth dimension, controlling the density of the cloud.

Restricting r_{diam} and m to reasonable values will restrict the size of the cloud. In Section 4, we will show empirically how these parameters effect the overall costs for Agyaat.

To enforce these parameters, each peer keeps a vector of its distance from all rendezvous nodes and stops accepting new neighbors when the limits are reached. The distance vector is easy to compute in a recursive fashion. In every ping cycle[6], a peer computes its distance vector from the distance vectors of its neighbors. For example, for a cloud with three rendezvous nodes, the distance vector of a peer with k neighbors is equal to:

$$\left[1 + \min_i(d_{i1}),\ 1 + \min_i(d_{i2}),\ 1 + \min_i(d_{i3})\right]$$

where $1 \leq i \leq k$ and d_{ij} is the distance of ith neighbor from jth rendezvous node. Now a peer which has m neighbors or is at r_{diam} distance from a rendezvous node will not accept any new incoming peers and the cloud is said to be saturated. Small temporary aberrations can be tolerated, since they will be short-lived due to the eventual convergence of the distance vectors and hence, not cause a major performance dip.

3.4.3 Extended Agyaat protocol. In the basic version of the Agyaat protocol, the query stays in only one cloud before the ring phase. For greater levels of anonymity and especially for greater resiliency against malicious attacks (as discussed in Section 5), the protocol can be extended to allow the query to pass through multiple clouds before the ring phase. For example, for a 2-cloud extension, the crossover peer can forward the query to some node in another randomly selected cloud, where a similar forwarding process takes place. This makes Agyaat more resilient to attacks since malicious nodes need to be present in significant number of clouds to determine the exact originating cloud and then the originating peer.

4. Performance analysis

In this section, we evaluate Agyaat's performance and discuss the effect of system parameters.

4.1 Scalability

We simulated various Chord networks varying the number of nodes. We then added-on Gnutella clouds to each topology. Each peer becomes a member of some cloud and a cloud is created when all existing ones are saturated. Then we ran 10,000 queries and computed the average costs. The queries were selected randomly, that is, a peer is randomly selected to query for a service by another random peer. Figure 7 shows the various costs based on R-Rings. As can be seen the costs for Agyaat and Chord follow the same trend and differ only by a constant. In this figure, we used an r_{diam} of 7, m of 5 and bounded the TTL of random walks by r_{diam} (since that almost traverses the length of the cloud – increasing it does not provide extra anonymity). We have also depicted the number of hops on the R-Ring. This number is smaller than for Chord, since the number of clouds formed (size of R-Ring) is smaller than the total number of nodes in the system (size of Chord ring). The constant difference is due to the random walks in the query cloud and the response cloud and the tracing back of the reply.

Next we look at scalability in terms of aggregate number of messages transmitted for a single query transaction. This includes the messages exchanged during the random walks, tracing back of the reply and most critically the broadcast of the query.

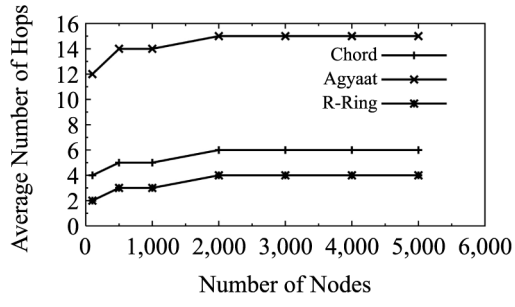

Figure 7.
Agyaat scalability: hops

Figure 8 shows that the overall trend for Agyaat is similar to Chord, in which case number of messages is equal to the number of hops.

Also, a big component of the costs of Agyaat is the number of messages in the Response Cloud, which includes the primary costs of broadcasting. The number of messages in the Query Cloud and on the R-Ring are small in comparison. We discuss mechanisms to further reduce this messaging costs in Section 6.2.

4.2 Effect of system parameters

Figure 9 shows the average number of hops when r_{diam} is varied from 3 to 9 with m fixed at 5 for a 5,000 node topology. Increasing r_{diam} allows more peers to be added in a single cloud, increasing the cloud size. From the figure, we see an interesting trend. While the cost of Chord stays the same (r_{diam} does not effect the main DHT ring), Agyaat begins to take more number of hops even when the number of hops on the R-Ring decreases.

This happens because with the increase in r_{diam}, even though the total number of clouds drops (each cloud can accommodate more), the average lengths of the random walks and the number of hops before the broadcast reaches the responding peer in the response cloud also increases. This increase offsets the decrease in R-Ring lookup hops. While this would indicate that we should keep r_{diam} to a minimum, notice that it effects the level of anonymity offered by a cloud. Very small r_{diam} values lead to small clouds and that provides little anonymity.

Next, we look at how r_{diam} effects the total number of messages transmitted. Figure 10 shows the average number of messages transmitted. The number of

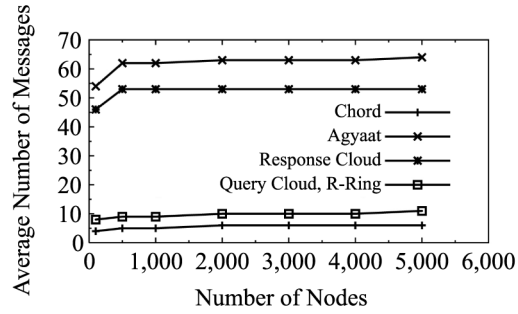

Figure 8.
Agyaat scalability: messages

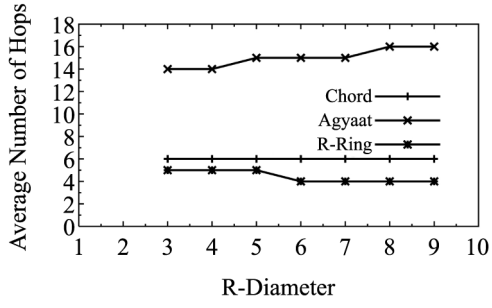

Figure 9.
R-diameter vs number of hops

messages increases linearly with r_{diam} due to a linear increase in the membership of the cloud (Response-Cloud plot in Figure 10).

We plot similar graphs for varying values of the degree m with r_{diam} fixed at 7 (5,000 nodes). Figure 11 depicts the effect of the degree on the average number of hops. Note that the increase in m decreases the average number of hops for Agyaat and in fact there is a drastic decrease in the number of hops on the R-Ring. This occurs since increasing m allows clouds to become very dense and allows more and more peers to join the same cloud. This reduces the number of clouds very quickly.

However, with the clouds becoming more dense, there is a penalty to be paid in regards to the number of messages transmitted. This is because now, a large number of messages will be transmitted in a response cloud because of the broadcast of query messages. This can actually be seen in Figure 12, where increasing m drastically increases the total number of Agyaat messages, with the biggest component being the broadcasting. In addition, larger m demands more resources from the peers, since they keep m open connections at all times.

This analysis indicates that we can control system performance by varying the two parameters with varying m providing fast changes and varying r_{diam} allowing for smaller fine tuning. This gives us a dynamic mechanism to adapt to system changes, for example, a sudden large influx of nodes can be handled by increasing m while smaller changes can be accommodated by varying r_{diam}.

Notice that varying the system parameters have an effect on the system in an indirect way. Changing parameters changes the number of clouds required to accommodate all peers and that effects the overall performance.

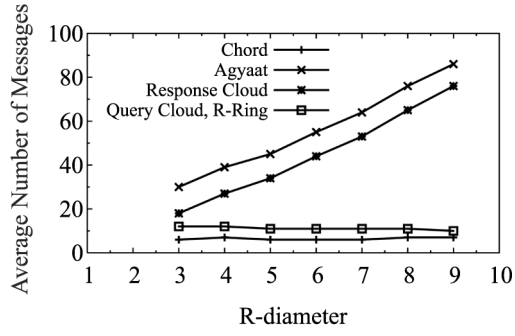

Figure 10.
R-diameter vs total messages

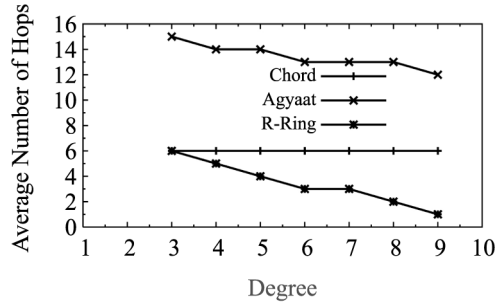

Figure 11.
Degree vs number of hops

Figure 13 shows how the two parameters effect the number of clouds for a 5,000 node topology. As can be seen from the graph, m has a much more drastic effect.

This leads to another possible mechanism to control anonymity/performance trade-offs. A system can fix the number of clouds, and control its characteristics that way. It would be implemented by creating all the clouds in advance and peers joining any cloud that is not saturated. This scheme requires a-priori knowledge of the capacity of the system and might over/under provision the network with a wrong estimate. We plot the system performance using this mechanism in Figures 14 and 15. As the number of clouds increases, there are two opposing effects – size of the R-Ring increases, while size of each cloud shrinks. Figure 14 shows that initially due to clouds getting smaller, total number of hops decrease, however slowly increase in R-Ring size becomes the dominating factor. In Figure 15, the messaging costs of query cloud and

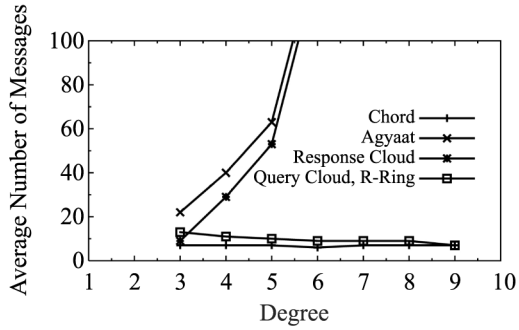

Figure 12.
Degree vs total messages

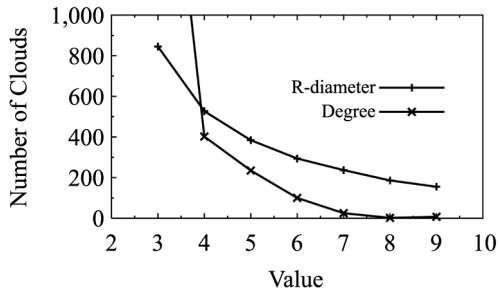

Figure 13.
Number of clouds

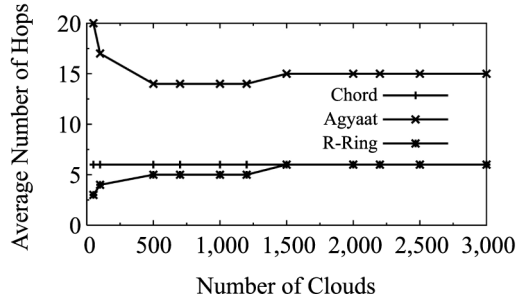

Figure 14.
Number of clouds vs hops

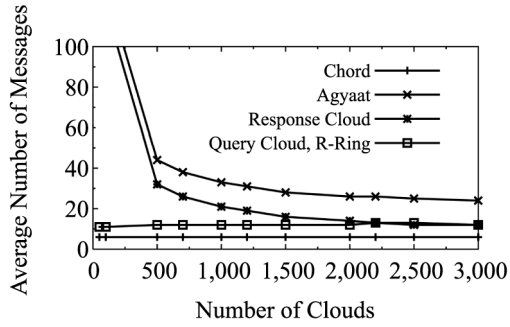

Figure 15.
Number of clouds vs messages

R-Rings nearly remain the same (increase in R-Ring, but smaller random walks), but due to the sharp decrease in costs in the response cloud initially, the overall messaging costs decrease until they stabilize when clouds have already become pretty small.

5. Security analysis

In this section, we look at possible anonymity-compromising attacks on Agyaat and propose some pragmatic defenses. We begin with our security model.

5.1 Model

In this paper, we focus on anonymity compromising mob attacks. We intend to investigate other attacks (Wright *et al.*, 2002) along with non-anonymity attacks like DoS, routing attacks, impersonation attacks in our future work. Since we anonymize the system through clouds at the end points, all attacks target these cloud topologies.

As used in other anonymity literature (Ratnasamy *et al.*, 2001, Wright *et al.* 2002), we assume the passive logging model for the analysis of our system. In this model, cooperating nodes passively log the messages going through them and share the logs offline to try and find out origins/destinations of the messages. In such a system, the malicious nodes will follow all protocols and there is no way to identify a malicious node by scrutinizing its behavior in the system. Also, we call a node to be compromised, if the malicious nodes can conclude with probability 1 that a message originated/terminated at that node. In what follows, the term "bad" stands for malicious intent and "good" for no malicious intent. We use the percentage of compromises and probability of exposure as our two anonymity metrics.

For a good node to prevent its compromise, all that is required is one good edge. That is, any connection with a non-malicious node is enough. This is because malicious nodes cannot conclusively say (with probability 1) that the message originated at a particular peer. This implies that for malicious nodes to compromise a good node, they need to completely surround the victim node. There is another subtle requirement. Not only do the malicious nodes need to surround the good node, they need the knowledge that the good node does not have any other edge (other than the connections to the malicious group), since the good node might also be connected to some other good node in the cloud. This knowledge is tough to obtain in a Gnutella like network, since the peers only hold information about their immediate neighbors. However, given significant resources at disposal of malicious nodes, some cloud regulation features of Agyaat can provide this critical information. For example, each peer maintains a

distance vector from the rendezvous nodes. With enough bad nodes surrounding the good nodes, looking at the distance vector, it may be possible to rule out edges between two good nodes. Also, if a good node has m bad connections already, it rules out that possibility.

There is one more scenario. Assuming there are less than m bad connections, the malicious nodes cannot rule out that the good node is not connected to some degree-1 good node, i.e. a node which is only connected to that good node and thus is unobservable from the bad nodes' perspective. Unfortunately, it is sometimes possible to exclude even this case due to Gnutella pings. When a node receives a ping, it returns a list of peers (node's neighbors) which can be used to establish more connections (the return message is called a pong). As a result, it is highly likely that enough information is available to know all the nodes in the cloud and thus rule out a phantom node. It might appear that a node can randomly avoid sending the list of peers in a pong. We do not advocate this since it is a critical step of bootstrapping and the only way incoming peers establish connections within the cloud.

5.2 Attacks

A malicious group of nodes can mount an attack by surrounding the good nodes in the cloud. Given a fixed amount of resources, a topology like Figure 16 (for three good nodes and $m = 2$), in which malicious nodes (solid) surround maximum number of good nodes possible, gives the most damaging attack scenario. It can be proved that the minimum number of bad nodes required to fully compromise G good nodes is equal to G (Singh and Liu, 2004). However, this topology requires malicious nodes to join in a particular manner and is not possible without on-the-fly manipulation, which is not permissible under the passive logging model. Next we consider few mob attacks.

First, we consider a simple attack during cloud creation. In this attack, the group of malicious nodes creates a new cloud and join en masse, waiting for good nodes to join. This way, they can surround good nodes as and when they come in. Notice that when a node joins a cloud, it immediately establishes a few connections and keeps some available for incoming peers. We simulated this situation and measured the effects. Figure 17 shows the graph for $m = 5$. The graph is plotted to indicate the evolution of clouds. Assume the fraction of malicious nodes in the cloud (when fully created) to be f. The graph plots the quality of the cloud, while it is being created, measuring percentage of good nodes in the cloud which are compromised (Y-axis) when some percentage of good nodes are added in the system (X-axis).

As the graphs indicate, less and less percentage of nodes remain compromised as and when more good nodes join the cloud. This occurs since with more good nodes added, there is a greater chance for a good node to have a good edge. For large fraction of bad nodes, the attacks have greater effects.

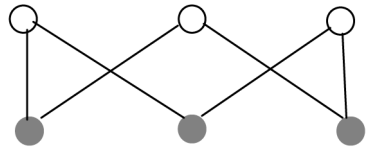

Figure 16.
Most damaging attack

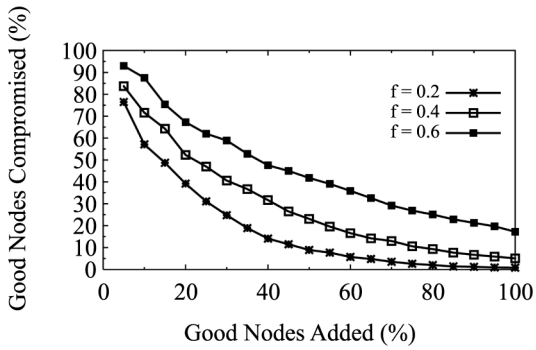

Figure 17.
Simple attack ($m = 5$)

There is another possible attack in which malicious nodes can cause greater damage. Note that the previous attack decayed since with more good nodes coming into the system, there was a greater chance of each having a good edge. This was aided by the fact that the bad nodes had joined the cloud before the good nodes started coming in. So when the good nodes entered, they left some slots available for later use and that was primarily responsible for the good edges. Any attack that surrounds the good nodes when they join the cloud by using up all possible slots would certainly be more successful. For example, an attack, in which bad nodes come in blocks of a few arriving after intervals of good node arrivals. This should work better since there is a greater chance of surrounding the node (a block of bad nodes receives the good nodes and another block joins later to saturate the good nodes). As shown in Figure 18, this attack is certainly more successful and more good nodes remain compromised at the end. In this simulation, bad nodes arrived in 10 equal sized blocks after every 10 percent of good node joins.

In addition, we also measured the probability of guessing the originator/destination of a message with varying system parameters. To measure this impact, we simulated running queries in a randomly created cloud and calculated the probability of guessing the correct good node to be the originator of a message. For this, the bad nodes analyze the logs to find the last bad node in the query path. Assuming that the TTL was bounded by r_{diam}, if the last bad node received TTL t_l, all good nodes within distance $(r_{diam} - t_l + 1)$

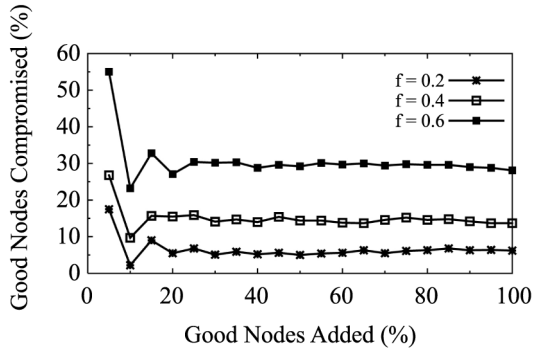

Figure 18.
Block attack ($m = 5$)

can be potential originators. Then, depending upon the other bad nodes in the path, some of the good nodes are ruled out. The remaining number of good nodes determines the probability of exposure. Figure 19 and Figure 20 plot the probability of exposure for various fractions of malicious nodes with varying cloud R-diameter and degree.

Figure 19 presents an interesting analysis. With increasing R-diameter, even though the circle of suspicion grows, the probability of exposure slightly increases. This is because there is a greater probability of a bad node being included in (now longer) query paths and thus the bad nodes can rule out more good nodes. With increasing cloud degree, the probability of exposure decreases since there are more good edges between good nodes, thus increasing the possible originators.

5.3 Defenses

In this section, we discuss defenses against above mentioned attacks. We show that it is tough to design a group strategy, which the good nodes can follow collectively, to reduce the risk of attacks. We then propose few pragmatic individual defenses.

It is important to recognize the fact that it is impossible to identify a malicious node under our security model. All malicious nodes behave completely identical to the good nodes. Hence, a good node while trying to defend itself from a compromise can only do so probabilistically, hoping that all its neighbors are not malicious. For example, it may decide to move within the network (dump its existing connections and create new ones), hoping that it will have at least one good edge. Assuming that there are G good nodes and B bad nodes with available slots[7], then this will happen with probability

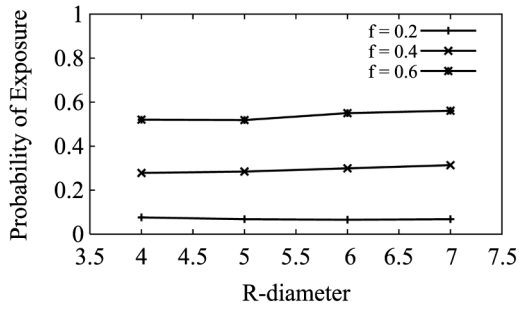

Figure 19.
Vary R-diameter

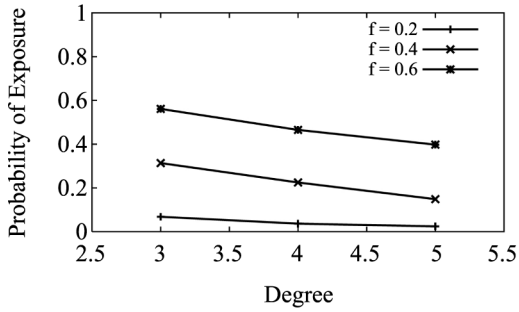

Figure 20.
Vary degree

for degree m. However, if all nodes (including malicious) move randomly, this effect will wither out, since that brings it to the same situation as during the creation of the cloud. Any similar group behavior can always be countered by malicious nodes, because of their greater knowledge about the network and ability to identify the good nodes. Therefore, it is tough to design any effective group defense. However, individual nodes can still try to move around in the cloud (maybe immediately before querying) hoping to connect to a good node.

There is one complete defense to a compromise. We call this a Get A Buddy Along (GABA) defense. In this, groups of good nodes join a cloud together and establish edges between them. This would require the good nodes to trust each other that they will not be sharing logs with any other peer. Other than that requirement, this is an excellent defense. The nodes need to trust each other only for not sharing the logs. No other querying/replying information needs to be shared. This is because for any message that a good node, Peer-A, receives from another good node, say Peer-B, it cannot conclusively ascertain that it originated at Peer-B. This defense is similar to the concept of K-anonymity (von Ahn *et al.*, 2003), where K is the size of the group.

Among other defenses:

- the extended version of the protocol (Section 3.4.3) provides much higher levels of anonymity by trading off on performance;
- limiting responses to Gnutella pings can prevent malicious nodes from getting a complete picture of the network, thus reducing traffic analysis abilities;
- connecting to the rendezvous nodes, since they are randomly selected based on their positions on the DHT ring and are less likely to be part of a malicious group active in the cloud; and
- if one assumes that malicious nodes are highly likely to be a group of nodes from a same local area network, it might be beneficial to connect to nodes from different domains; even though it is not accurate defense, it does make the task of malicious nodes a little tougher.

6. Design benefits

In this section, we will discuss other benefits of the Agyaat design, which can be seen as by products.

6.1 Data management

Let us first look at how Agyaat influences the management of data. For the moment, we do not consider provisions for mutual anonymity, rather just how the topology compares to DHT overlays. In typical DHT systems, a data item is stored at a peer that immediately succeeds the hash of the data item on the DHT ring (or appropriate meta information is stored at such a peer). This leads to a common drawback of DHT systems, in which the publisher does not necessarily control the access to its data and some work has been done (N. Harvey *et al.*, 2003, Zhang *et al.*, 2004) to address this problem. It is interesting to see that Agyaat can avoid this issue to a great extent. For example, a company wishes to provide some services (say publish some content) and needs to use a cluster of machines. Now, if it uses a simple DHT based mechanism, its data may get hashed to various different nodes in the network and it in turn, may be required to host some other company's data (maybe its competitor!). With Agyaat, the company can

create an appropriate cloud and its cluster of machines can join that cloud. In such a manner, queries for its services can be routed to its own cloud of machines.

Also, greater semantic grouping is possible with various companies offering similar services belonging to a single cloud. This leads to a more structured management of data, which ironically, structured P2P systems do not provide. It is still possible to have a few outside nodes – the rendezvous nodes, since they are selected based on the DHT ring, though the cloud name can be manipulated to select a particular node as the rendezvous node (Castro *et al.*, 2002). Also, for advertisement of services, clouds can tag both queries and replies with cloud based information like statistics (parameter settings) and services offered. This can be cached at intermediate peers and made available to peers looking to join a cloud.

6.2 Cloud topologies and DHT membership

As discussed earlier, Agyaat has higher messaging costs primarily due to the broadcast of the queries in the response cloud. It was from the fact that it is required that messages reaches all peers in the cloud. We can alleviate this problem by changing the topology of a cloud. Recall that our only requirement for a cloud was that it should be an LTI topology. If we take an overall look at Agyaat, it itself is an LTI topology (since both the query initiation and termination happen locally in the clouds). So we can imagine a two level hierarchical topology in which the top-level of the cloud is another DHT ring. The queries coming into the cloud are hashed onto this top level ring and appropriate level-2 rendezvous nodes are found, which broadcast the query in the lower level Agyaat cloud. The idea is to reduce the number of nodes receiving the message by appropriately routing it to a smaller subset of nodes. This can be extended to greater than two levels as well (Ganesan *et al.*, 2004). However, note that the anonymity provided will be less since now the anonymizing components of the clouds will be smaller. Hence there is a lesser cloaking effect. Also note that now, joining a cloud would be a little more complicated, since based on the services being offered, the peer would join a particular unstructured component. It is also possible to have clouds with different topologies in the same Agyaat system, offering variable levels of anonymity and performance benefits. The clouds can advertise this feature through mechanisms described earlier. Then,incoming peers can tradeoff between the two based on their desired applications.

Another interesting idea is to allow peers to be part of a cloud without being on the main DHT ring (hybrid super-peer architecture!). This will offload the load on the DHT ring.

7. Related work

A significant amount of research has been done to provide anonymity solutions in various distributed systems. Projects like Crowds (Reiter and Rubin, 1998), Mix (Chaum, 1981), Onion (Syverson *et al.*, 1997) and others (Freedman *et al.*, 2002, Freedom, 2003) have been very popular. There has also been work on analysis of these various anonymous protocols (Wright *et al.*, 2002, Syverson *et al.*, 2000). The P2P domain has also seen work in this area with systems like Freenet (Clarke *et al.*, 2001), P5 (Sherwood *et al.*, 2002), APFS (Scarlata *et al.*, 2001), (Xiao *et al.*, 2003). However, none of these projects target the mutual anonymity problem of structured P2P systems. The structured DHT based systems like Chord (Stoica *et al.*, 2001), Pastry (Rowstron and

Druschel, 2001) were developed to tackle the lack of scalability and guaranteed location of data in unstructured P2P systems (Gnutella, 2002, Kazaa, 2002) and mutual anonymity was not even a design goal.

Our work is distinguished from the prior anonymity research because of our target domain of DHT based overlays. As described in Section 2.1, DHT based systems by their basic design, are in-conducive to anonymity (especially service provider's). Till recently, there have been no significant attempts at addressing this problem. Though (O'Donnell and Vaikuntanathan, 2004) formally analyzes the information leak in the Chord protocol, it provides no solutions. Achord (Hazel and Wiley, 2002) developed only for Chord fails to provide a generic solution and also, as pointed out by the authors, is susceptible to information leakage during the stabilization process. TAP (Zhu and Hu, 2004) provides a tunneling approach which uses an expensive cryptographic approach to carry messages. In the basic protocol of TAP, each hop on the DHT requires setting up query and response tunnels and deploying cryptographic keys at the nodes in the path. In contrast, Agyaat derives its anonymity features from the basic topology design and is much lower in costs. Also, as discussed in Section 6, there are substantial other benefits with a topology design like Agyaat and we believe that overall this makes Agyaat a better suited solution.

8. Conclusions and future work

We presented Agyaat, a decentralized system that provides mutually anonymous services over structured P2P networks while ensuring scalable lookup and guaranteed location of data. We identified critical topology properties (local termination/initiation) which are essential for mutual anonymity and introduce clouds (LTI topologies) to incorporate such properties into Agyaat. We described a number of mechanisms to enhance the scalability and efficiency of routing between and within Agyaat clouds, ensuring the guaranteed lookup of data. We showed that Agyaat is as scalable as DHT based systems in terms of both number of hops and aggregate messaging costs (differing only by constants). We also studied the effect of various system parameters and performed a security analysis of the system including possible anonymity-compromising attacks and proposed defenses to such attacks.

In future, we intend to work on other security aspects of Agyaat, especially Denial-of-Service (DoS) attacks and DHT routing vulnerabilities as discussed in (Castro *et al.*, 2002, Sit and Morris, 2002), aiming at providing mutually anonymous and secure services over structured and decentralized overlay networks.

Notes

1. Hindi for "anonymous"
2. Some form of timing attack is possible to find out where the reply originated, but it can be thwarted by adding random delays before initiating the reply.
3. This condition can be potentially relaxed (see Section 6).
4. This number effects the amount of initial anonymity of the cloud. For example, if there was only one rendezvous node, then clearly at the time of cloud creation, no immediate anonymity can be provided.
5. Even when it is, i.e. querying peer started with TTL 0, that fact can not be conclusively ascertained by other peers.

6. Gnutella requires each member to periodically ping its neighbors to check for node failures. As a result, no extra messages are used.
7. Note the term available slots. Saturated nodes cannot accept new connections and hence cannot contribute.

References

Castro, M., Druschel, P., Kermarrec, A. and Rowstron, A. (2002), "SCRIBE: A large-scale and decentralized application level multicast infrastructure", *IEEE Journal on Selected Areas in Communication*, Vol. 20 No. 8, pp. 1489-99.

Chaum, D. (1981), "Untraceable electronic mail return addresses, and digital pseudonyms", *Communications of ACM*, Vol. 24 No. 2.

Clarke, I., Sandberg, O., Wiley, B. and Hong, T. (2001), "Freenet: a distributed anonymous information storage and retrieval system", *Springer Lecture Notes in Computer Science*.

Crespo, A. and Garcia-Molina, H. (2002), "Semantic overlay networks for p2p systems", Stanford Technical Report, Stanford University, Stanford, CA.

Freedman, M.J., Sit, E., Cates, J. and Morris, R. (2002), "Introducing tarzan, a peer-to-peer anonymizing network layer", *International Workshop on Peer-to-Peer Systems, Cambridge, MA, March 7-8*.

Freedom (2003), available at: www.freedom.net/

Ganesan, P., Gummadi, K. and Garcia-Molina, H. (2004), "Canon in G major: designing DHTs with hierarchical structure", *International Conference on Distributed Computing Systems, Tokyo, March 23-26*.

Gnutella (2002), available at: http://gnutella.wego.com/

Harvey, N., Jones, M., Saroiu, S., Theimer, M. and Wolman, A. (2003), "Skipnet: a scalable overlay network with practical locality properties", *USENIX Symposium on Internet Technologies and Systems, Seattle, WA, March 26-28*.

Hazel, S. and Wiley, B. (2002), "Achord: a variant of the chord lookup service for use in censorship resistant peer-to-peer publishing systems", *International Workshop on Peer-to-Peer Systems, Cambridge, MA, March 7-8*.

Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D. (1997), "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web", *Symposium on Theory of Computing, El Paso, TX, May 4-6*.

Kazaa (2002), available at: www.kazaa.com/

O'Donnell, C. and Vaikuntanathan, V. (2004), "Information leak in the chord lookup protocol", *IEEE International Conference on P2P Computing, Zurich, August 25-27*.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S. (2001), "A scalable content-addressable network", *ACM SIGCOMM, San Diego, CA, August 27-31*.

Reiter, M. and Rubin, A. (1998), "Crowds: anonymity for web transactions", *ACM Transactions on Information and System Security*, Vol. 1, pp. 66-92.

Rowstron, A. and Druschel, P. (2001), "Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems", *International Conference on Distributed Systems Platforms, Heidelberg, November*.

Scarlata, V., Levine, B. and Shields, C. (2001), "Responder anonymity and anonymous peer-to-peer file sharing", *International Conference on Network Protocols, Riverside, CA, November 11-14*.

Sherwood, R., Battacharjee, B. and Srinivasan, A. (2002), "P5: A protocol for scalable anonymous communication", *IEEE Symposium on Security and Privacy, Oakland, CA, May 12-15*.

Singh, A. and Liu, L. (2004), *Agyaat: Providing Mutually Anonymous Services over Structured P2P Networks*, GIT-CERCS-04-12.

Singh, A., Srivatsa, M., Liu, L. and Miller, T. (2004), "Apoidea: a decentralized peer-to-peer architecture for crawling the world wide web", *Springer Lecture Notes in Computer Science 2924*.

Sit, E. and Morris, R. (2002), "Security considerations for peer-to-peer distributed hash tables", *International Workshop on Peer-to-Peer Systems, Cambridge, MA, March 7-8*.

Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H. (2001), "Chord: a scalable peer-to-peer lookup service for internet applications", *ACM SIGCOMM, San Diego, CA, August 27-31*.

Syverson, P., Goldschlag, D. and Reed, M. (1997), "Anonymous connections and onion routing", *IEEE Symposium on Security and Privacy, Oakland, CA, pp. 44-53*.

Syverson, P., Tsudik, G., Reed, M. and Landwehr, C. (2000), "Towards an analysis of onion routing security", *Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, July 25-26*.

von Ahn, L., Bortz, A. and Hopper, N. (2003), "k-anonymous message transmission", *ACM Conference on Computer and Communications Security, Washington DC, October 27-30, pp. 120-130*.

Wright, M., Adler, M., Levine, B. and Shields, C. (2002), "An analysis of the degradation of anonymous protocols", *Symposium on Network and Distributed System Security, San Diego, CA, February 6-8*.

Xiao, L., Xu, Z. and Zhang, X. (2003), "Low-cost and reliable mutual anonymity protocols in peer-to-peer networks", *Transactions on Parallel & Distributed Systems*, Vol. 14 No. 9, pp. 829-40.

Zhang, Z., Mahalingam, M., Xu, Z. and Tang, W. (2004), "Scalable, structured data placement over p2p storage utilities", *International Workshop on Future Trends in Distributed Computing, Suzhou, May 26-28*.

Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A. and Kubiatowicz, J. (2004), "Tapestry: a resilient global-scale overlay for service deployment", *IEEE Journal on Selected Areas in Communication*, Vol. 22 No. 1, pp. 41-53.

Zhu, Y. and Hu, Y. (2004), "A novel tunneling approach for anonymity in structured P2P systems", *International Conference on Parallel Processing, Montreal, August 15-18*.

Further reading

Castro, M., Drushel, P., Ganesh, A., Rowstron, A. and Wallach, D. (2002), "Secure routing for structured peer-to-peer overlay networks", *Symposium on Operating Systems and Implementation, OSDI'02, Boston, MA, December*.

Ratnasamy, S., Handley, M., Karp, R. and Shenker, S. (2001), "Application-level multicast using content-addressable networks", *Lecture Notes in Computer Science, 2233*.

Corresponding author

Aameek Singh can be contacted at: aameek@cc.gatech.edu