Project no.	034084
Project acronym:	SELFMAN
Project title:	*Self Management for Large-Scale Distributed Systems based on Structured Overlay Networks and Components*

European Sixth Framework Programme
Priority 2, Information Society Technologies

Deliverable reference number and title:	D.1.3a First report on Security in Structured Overlay Networks
Due date of deliverable:	July 15, 2007
Actual submission date:	July 15, 2007
Start date of project:	June 1, 2006
Duration:	36 months
Organisation name of lead contractor for this deliverable:	NUS(P7)
Revision:	1
Dissemination level:	CO

First report on Security in Structured Overlay Networks

Felix Halim, Roland Yap and Yongzheng Wu
School of Computing
National University of Singapore

{halim,ryap,wuyongzh}@comp.nus.edu.sg

Abstract

Peer-to-Peer systems have emerged as a mechanism for sharing resources and computation which take advantage of distributed computers and the Internet. Its importance can be seen by the fact that peer-to-peer applications are very popular on the Internet and consume a significant fraction of the Internet bandwidth.

Peer-to-Peer systems bring new challenges in building self-organizing distributed systems. This survey studies peer-to-peer systems focusing on scalable mechanisms, peer-to-peer applications and the challenges which will be needed to build robust and secure solutions.

Contents

1 Introduction

Peer-to-Peer (P2P) applications are very popular on the Internet particularly for sharing content or files. In fact, so popular that a large proportion of the Internet traffic is due to such P2P applications. Rather than trying to define P2P systems or applications, we will instead characterize them by some general properties since it is hard to cover all systems. A peer-to-peer system or application usually has a number of interacting and cooperating entities which are called the *peers*. The peers are distributed and are decentralized. This means that the resources of the peers such as bandwidth, storage and computation are also distributed. (In the rest of this paper, we will use the term peers and nodes interchangeably). Unlike client-server systems, P2P systems tend to be self organizing rather than having centralized control and coordination. Peers may be symmetric with respect to their interaction, a peer might be both a client or server. Furthermore, often the peers are autonomous. The peers are connected by a network, usually the Internet, however there are dynamic and transient features which need to be taken into account. Peers may or may not be available all the time, e.g. a peer can simply fail or leave the system. Communication between peers may also be transient, e.g. the communication may not be reliable or some peers may be sometimes partially disconnected, etc. Thus the common elements of peer-to-peer are sharing of distributed resources, decentralization, self-organizing and ability to tolerate unavailability of resources and communications.

This paper is a survey of Peer-to-Peer architecturs and systems focusing on the aspects of P2P which relate to security. Other issues like the robustness and fault tolerance of the P2P system or applications are also close to the concerns of security and will be partly covered by this survey. We will also survey various P2P applications since ultimately P2P systems need to deal with security and robustness at the application level.

The survey is organized as follows. The remainder of this section describes the evolution of P2P systems. Section 2 surveys the work on structured overlay networks focusing on Distributed Hash Tables (DHTs). P2P applications which rely on DHTs are discussed in section 3. Other Peer-to-Peer applications which are not based on DHTs are described in section 4. Section 5 introduces the general issues of security which are applicable to P2P systems and section 6 describes in greater details various solutions to some of these security issues.

1.1 Peer-to-Peer Background

Peer-to-Peer was initially popularized as a mechanism for sharing content and files by Napster. Napster provided a centralized index server where peers could publish the locations of files which could also be searched. The centralized topology of Napster was its weakness since it became a single point of failure and ultimately was challenged by legal liability issues. Napster also assumed that files were not updated thus relieving the need to update changes to replicas (this is not necessarily the case for other applications). Although Napster is a P2P application, it is better

classified as Hybrid P2P since it is only partly decentralized and has a client/server architecture for the central index.

The second generation P2P file sharing applications moved away from a centralized index server to purely decentralized mechanisms. A prime example of this is the Gnutella P2P system. Every peer maintains its own files and peers can query and serve requests among each other. Gnutella is completely decentralized since each peer only maintains local information and communicates to its neighbours.

In order to search for a data item, a flooding mechanism is used. A query is propagated between peers by flooding the peers which are reachable from it. A query propagates until it has found a peer containing the data item or it stops because of a timeout or node failure. The advantage of the completely decentralized and unstructured nature of Gnutella is that very little effort is needed to maintain the network of peers. Queries are anonymous and provide deniability since a peer only knows its neighbours which can forwarding a query without being the originator of a query. The drawbacks that Guntella does not give any search guarantees. The flooding mechanism used does not scale and leads to a large number of messages and bandwidth usage as the number of nodes increase,

The second generation P2P systems were characterized by decentralization without structure on the topology of the peers. The third generation P2P systems differ in that in addition to being decentralized, the peers (we will also call them nodes) have a structure in how nodes communicate and where data items are placed. As a "virtual" network is created over nodes connected by an underlying network, this is usually called a *structured overlay network*. These are covered in more detail in Section 2.

2 Structured P2P Overlay Networks and Distributed Hash Tables

Structured P2P overlay networks are virtual networks which are meant to address the problems of second generation P2P systems. The network consists of a possibly large number of nodes (n) which are connected by a particular virtual network topology which usually forms a distributed data structure. Many of the algorithms are based on the organization of a distributed hash table (DHT). However, other distributed data structures have also been used. Most of the structured overlay networks are scalable in the sense that looking up a data item is better than the linear time complexity in the second generation schemes while dealing with the dynamism and failure in the nodes.

2.1 DHT Concepts

A DHT can be viewed as an abstract data type similar to a hash table. The difference from a regular hash table is that it stores key-value pairs among the n distributed nodes. The abstract generic DHT interface looks like that of a hash table:

- `put(`*key*`,` *data*`)`: inserts item *data* into the DHT
- *data* `= get(`*key*`)`: lookup the item with the given *key* in the DHT

The nodes are usually identified using some node identifier (node ID). It is common that node IDs and keys range over a common identifier space consisting of the integers from $\{0, 1, \ldots, N-1\}$. This is also called a Global UID (GUID). A get/put operation is routed through the overlay network to the node (or nodes) which hold the data item.

Different DHTs vary in how they address the following implementation challenges:

- scalability of the lookup
- topology of the network
- space requirements for the local node routing information
- balancing of the resources or load: this includes the mapping of data items to nodes
- maintenance of the DHT: how nodes joining and leaving the network
- handling of failures: this can also include resource replication
- awareness of the underlying network topology and performance

The remainder of this section surveys a variety of DHTs within the space of implementation possibilities.

In addition to the basic DHT algorithms, there are schemes which deal with the security issues. For example, there may be malicious nodes in the network, can the nodes be anonymous, are there trust mechanisms, etc. We will go into further detail in Section 5 and 6. In Section 3, we will survey applications of DHTs as well as particular application level security issues.

2.2 Chord

The Chord DHT uses an identifier space from 0 to $2^L - 1$ where $N = 2^L$. The nodes are organized into a ring which can be viewed as a distributed circular list where each node ID connects to the next higher ID modulo N. The mapping of node IDs and data items is using a form of *consistent hashing* [41]. Node IDs are obtained by hashing the node IP address. The key of the data is also hashed using the same hash function such as SHA-1. The data item is stored at the node whose identifier is equal to or follows the hash in the identifier space, this is called the *successor node*.

Consistent hashing is used so that dynamism caused by nodes entering and leaving the network only minimally disrupts the movement of data items. When a node leaves the network, its data items are transferred to its successor. Conversely, a node joining the network is assigned the data items which would have been assigned to its successor. Consistent hashing allows the data items to be load balanced among the nodes.

Looking up a data item can be found simply by following the successor nodes, e.g. following the distributed link list next pointer. To speedup the search by a logarithmic factor, a routing table is used. The routing table in a node, called the *finger table*, refers to: $+1$ (the successor), $+2$, $+4$, $+2^3$, $+2^4$, ..., $+2^{L-1}$ of the successors of the node. Lookup simply uses the finger table to jump in an exponential fashion to the nearest node that has its ID less than the key.

When nodes join and leave, the routing table needs to be maintained since the pointers may no longer be correct. Chord chooses to maintain the single successor pointer in the routing table correctly. The remaining pointers in the routing table might not be correct either. However, rather than maintaining them at the time of leave and join, Chord uses a periodic stabilization strategy. Periodically an entry in the finger table is updated by a lookup for the successor ID for a finger pointer at that level. Stabilization also deals with node failures.

In the steady state (no nodes joining or leaving the network), in an n-node Chord system, each node maintains information about only $O(log\ n)$ other nodes (the size of the routing table), and resolves all lookups via $O(log\ n)$ messages to other nodes. Performance degrades gracefully when there are changes in the network. This is important in practice because nodes will join and leave arbitrarily. Only one successor node in the finger table need be correct in order for Chord to guarantee correct (though possibly slow) routing of queries.

Chord deals with failures by using replication of the ring pointers. In addition

to the routing table, each Chord node maintains a r-successor list to its nearest r successors. This allows routing to succeed even when nodes have failed. Data can also be replicated on the r successors.

An example application of Chord is CFS [20] which is described in Section 3.2.3.

2.3 Pastry

Pastry differs in the structured overlay network topology, Pastry uses numeric closeness of the key to locate where the request message should be forwarded while Chord uses a fixed routing table (there are only O(log N) possible nodes to route a message). Pastry also takes into account network metrics such as proximity of nodes, latency, bandwidth, etc.

Like Chord, Pastry uses numbers in the range 0 to $2^L - 1$ for node IDs and keys. IDs and keys are treated as numbers in base b with L/b digits. For example, if $L = 12$ and $b = 2$ then the identifier space $N = 2^L = 2^{12}$ (4096 possible node IDs) and since the base is $2^b = 2^2 = 4$, so every node can be represented in $L/b = 12/2 = 6$ digits (where each digit ranges from 0 to 3).

Each node maintains a routing table, *neighbourhood set* and *leaf set*. The neighbourhood set is the set of nodes closest to the node by some network metric other than the ID, e.g. network proximity. The leaf set are the nodes closest in the identifier space, this is like a union of a successor and predecessor list. Keys are mapped to the numerically closest node.

The routing table has L/b rows (exactly the same as the number of digits to represent a Pastry node ID). The first row of the routing table contain links to node IDs that differ at the first digit with the current node ID (i.e. no common prefix digit), the second row contains links to node IDs that differ at the second digit (i.e. one prefix matches), and so on until L/bth row which contain links to node IDs that differ at the last digit (i.e. $L/b - 1$ prefix matches).

To route a key to a node responsible for it in Pastry, first a node checks whether the key k is within the range of its leaf set. If it is covered by the leaf set then the node forwards the query to the leaf set node numerically closest to k. In case it's the node itself, the routing process is finished. If k does not belong to leaf set nodes, this node will route the query to a node which shares a longer common prefix with k in the routing table. If there is no such entry in the routing table, the query is forwarded to a node which shares a prefix with k of the same length as the node but which is numerically closer to k. This scheme ensures that routing loops do not occur because the query is routed strictly to a node with a longer common identifier prefix than the current node, or to a numerically closer node with the same prefix.

Pastry has some advantages over Chord as it takes advantage of network locality. The neighbourhood set is not involved in the routing itself but in maintaining network locality in the routing information. By exploiting network locality, Pastry routing optimizes not only the number of hops but also the cost of each individual hop. The criteria to populate a node's routing table allow a choice among a number of nodes with matching ID prefixes for each routing table entry. By selecting nearby nodes in term of network locality, the individual routing lengths are minimized.

Pastry allows the base b to be configurable which determines the size of the routing table. The bigger the base, the faster the routing but it also increases the amount of state that needs to be managed at each node. A Pastry implementation can chose appropriate trade-off for specific applications. Pastry uses lazy detection failure that only fixes an out-of-date routing table when it encounters failure.

2.4 DKS

The Distributed k-ary System (DKS) [9] is a generalization of Chord which allows for faster routing with a larger routing tables. In Chord, a node with ID p the furthest jump in the routing table points to the successor at $p+2^{L-1} \bmod N$. So the maximum partitioning of the identifier space in a single step is two. DKS generalises this by partitioning the space in k ways rather than two as in Chord. Routing becomes faster with $O(log_k\ N)$ hops and the routing table size is $(k-1)log_k\ N$.

Instead of using periodic stabilization, DKS maintains the routing table using a correction-on-use technique. Routing messages contain information about the sender so the receiver can determine that the sender has erroneous routing entries and correct it. Later versions of DKS use atomic ring maintenance and failure detectors. This has the property of guaranteeing lookup consistency in the presence of dynamism. Lookups in Chord, on the other hand, might fail even when the key is present because the routing tables are not yet consistent.

2.5 CAN

Most DHTs use a one dimensional key and ID space. Content-Addressable Networks (CAN) [57] use a topology based on a virtual d-dimensional torus which can be thought of as a generalisation of a 1-dimensional ring. A node in CAN is a zone which is responsible for a sub-space within the d-dimensional space (in ring-based DHTs, a node is responsible for a ring segment). A key in CAN is represented as a point which is obtained by mapping a hash of the key to a d-dimensional point. Each node is responsible for the set of keys (points) which are contained in its zone.

The topology of the routing table is based on the neighbouring zones of a node. CAN nodes have $2d$ neighbours which are adjacent zones. Note that the size of the CAN routing table is based on d rather than N. A lookup in CAN works by routing to the closest neighbour in the direction of the key. Intuitively, this is like following a straight-line path from source to destination coordinates. For a d dimensional space which is partitioned into n equal zones, the average routing path length is $O(dn^{1/d})$.

When a new node joins, it randomly picks a point in an existing zone. The zone is split in two: one for the old node, the other is for the new node. Both nodes update their neighbours and routing tables. When a node leaves, it hands its zone to be merged with a takeover node, one of its neighbours if the merger produces a single valid zone, Otherwise the takeover node temporarily handles the deleted zone.

CAN can be optimized to avoid bottlenecks and to improve the availability of the system by having multiple instances (reality) of the DHT. Alternatively, by mapping the key to more than one point, this also avoids hot spots, provides replication and avoids bottlenecks as well as being faster by allowing parallel queries (additional node state is needed). CAN also can incorporate routing metrics based on Round-Trip Time (RTT) to neighboring nodes and use this information to forward to messages to those neighbors with the best ratio of RTT and ID space to destination.

2.6 Viceroy

A network with a large routing table can reduce lookup path length but also increase the periodic maintenance cost for the routing table itself, especially when a lot of joins/leaves occur. The Viceroy DHT uses constant size routing tables which can be advantageous in the following situations:

1. Practical costs for updating links far exceeding the normal lookup costs since it involves coordination between nodes or might require locking to maintain consistency of the overlay network.

2. It reduces the ambient traffic associated with pings and control information. Maintaining a constant degree network relieves the concern about the cost of opening too many connections at the nodes.

3. The degree of the network directly relates to the load incurred by join and leaves.

Even though low degree networks are not suitable for failure-prone environments, Viceroy justifies it by handling the issue separately from the routing design using clustering techniques.

Viceroy's structured overlay network topology combines an approximate butterfly network with a Chord ring. Nodes also have a new state called a *level* which is chosen randomly from 1 to $log\ n$. Each node at level l maintains the following links:

- two down edges to nodes at level $l + 1$: one at a distance roughly $1/2^l$ away and one close-by;

- an up edge to level $l - 1$ if $l > 1$;

- level-ring links to the next and previous nodes at the same level.

By using a random network construction, routing can be achieved in $O(log\ n)$ hops with high probability.

2.7 Kademlia

Kademlia improves on the Pastry routing algorithm by removing the second routing phase of Pastry which reduces performance, and complicates the analysis for the worst case. A feature of Kademlia is that the configuration information spreads automatically as a side-effect of key lookups. Kademlia also introduces a concurrency parameter α that is used in the routing process and lets the users trade a constant factor in bandwidth for asynchronous lowest-latency hop selection and delay-free fault recovery. Kademlia maintains a list of k-bucket in the routing table where k is a system-wide replication parameter which is chosen such that any given k nodes are unlikely to fail within an hour of each other.

Kademlia uses a 160-bit GUID for node identifier space. A binary tree topology can be constructed from the bits (i.e. bit 0 corresponds to the left child, bit 1 correspond to the right child). The nodes are the leaves in the binary tree. Each bit in the 160 bit node ID will have a subtree correspond to it. Links to the nodes in the subtree are stored in a k-bucket which contains contact information to several nodes in that subtrees, providing k replication of contacts to improve robustness as well allowing $\alpha \leq k$ concurrent queries for lookup. Kademlia protocol ensures that every node knows of at least one other node in each of its subtrees, guaranteeing that any node can locate (can be reach from) any other node.

When a node looks up an ID, it checks to which subtree does the id belong and forwards the query to α randomly selected nodes from the k-bucket of that subtree. Each node possibly returns back a k-bucket of a smaller subtree closer to the id. If the node responsible is not found, other α randomly selected nodes are contacted, and so on. The returned k-bucket information can be used to update the node's state thus a side-effect of the lookup is also an update for the node state. The buckets are managed using a least-recently seen eviction policy, except that live nodes are never removed from the list. Based on Gnutella trace data, the longer a node has been up, the more likely it is to remain up for another hour. This maximizes the probability that the nodes in a bucket will remain online. Another benefit of k-buckets is that they provide resistance to certain DoS attacks (one cannot flush the node's state by flooding the system with new nodes).

Nodes in Kademlia ensure the persistence of key-value pairs by periodically republishing keys. The reasons are some of the k nodes that initially get a key-value pair when it is published may leave the network and new nodes may join network with ID closer to some published keys than the key-value pair was originally published.

2.8 Kelips

Kelips [32] allows increased efficiency and stability through increased memory usage and communication overhead. The idea is that the presence of slow logical hops in a logarithmically long path is likely if there are a significant fraction of nodes with high latency/low bandwidth links. Kelips tries instead to avoid logarithmic lookup and gives O(1) lookup cost with $O(\sqrt{N})$ space. Furthermore, it provides

for quick convergence after join/leaves and exhibits stability under churn. This resilience is achieved through the use of a lightweight Epidemic multicast protocol for replication of system membership data and file indexing data [12, 23].

Kelips consists of k virtual affinity groups, numbered 0 through $k-1$. Each node lies in an affinity group determined by using a consistent hashing function to map the node ID into the integer interval $[0, k-1]$. A node state consists of the following entries:

- Affinity Group View: containing a set of nodes lying in the same affinity group along with their round trip time, etc.
- Contacts: For each of the other affinity groups in the system, a small (constant-sized) set of nodes lying in the foreign affinity group.
- Filetuples: A (partial) set of tuples, each detailing a file name and host IP address of the node storing the file (called the file's homenode). A node stores a filetuple only if the file's homenode lies in this node's affinity group.

If a node wants to query for a file, it maps the file name to the appropriate affinity group by using the same consistent hashing used to decide node's affinity groups. It then sends a lookup request to the topologically closest contact among those it knows for that affinity group. A received lookup request is resolved by searching among the filetuples maintained at the node, and returning to the querying node the address of the homenode storing the file. This scheme returns the homenode address to a querying node in O(1) time and with O(1) message complexity. Finally, the querying node fetches the file directly from the homenode.

If a node wants to insert a file, it maps the file name to the appropriate affinity group, and sends an insert request to the topologically closest known contact for that affinity group. This contact picks a node h from its affinity group, uniformly at random, and forwards the insert request to it. The node h is now the homenode of the file. The file is transferred from the origin node to the homenode. A new filetuple is created to map the file to homenode h, and is inserted into the gossip stream (at a fixed interval, a node selects a small set of nodes to which information is multicast). Thus, filetuple insertion also occurs in O(1) time and with O(1) message complexity. The origin node periodically refreshes the filetuple entry at homenode h in order to keep it from expiring.

2.9 SkipNet

SkipNet [35] focuses on data placement and access route path. Most DHTs do not have control over the data placement. Thus data may be stored far away from its users, perhaps outside its domain. Also the routing path to access local data may leave the local organization. SkipNet is designed with content locality and path locality in mind. Content locality provides the ability to explicitly place data on a single node or on a set of nodes. Path locality guarantees that local traffic remains local.

Data controllability is needed for organizations that want to control over their data even if local data is globally available. They can manage data administration easily by having the data constrained in a set of nodes (data center / cluster). Local data can survive network partition providing local availability and performance since data is stored near clients that use it also can be used as a building blocks for dealing with certain external attacks.

Useful consequence of SkipNet's locality properties is that partition failures, in which an entire organization disconnects from the rest of the system, can result in two disjoint, but well-connected overlay networks. SkipNet can efficiently re-merge these disjoint networks when the partition heals.

SkipNet achieves constraint placement and uniform data distribution by using two separate ID spaces: a string name ID space and a numeric ID space. SkipNet uses the Skip List [54] data structure as the structured overlay network topology. The skip list is a sorted linked list with subset of nodes having additional links to skip over some list nodes. The topology is similar to Chord's ring topology with finger tables replaced with links to skip over data records in Skip List. The difference with Chord is that SkipNet leverages sorted name IDs in the list and thus supports range queries on the keys.

SkipNet can route efficiently in both address spaces (name ID space and numeric ID space) with a simple rule: forward the message to node that is closest to destination, without going too far by skipping as many nodes as possible. The complexity is O(log N) with high probability (because SkipNet uses probabilistic Skip List thus the search, insert, delete complexity is O(log N) with high probability).

The primary security benefit for SkipNet is that content locality ensures that data stays within organization and path locality prevents malicious forwarders, analysis of internal traffic and external tampering. However the weakness is that it's easier to target organizations, for example if someone creates one million nodes with name prefixes "a" and "c", most traffic to/from "b" will go through "a" or "b" intermediate nodes.

3 DHT Applications

This section surveys various applications of DHTs organized together with a classification. In some cases, there is some overlap in the classification, e.g. file sharing, storage systems and content distribution share some common properties. But file sharing focuses on downloading and files are not persistent in the system. Storage systems focuses on persistently storing files and files are named uniquely and persistently so that they can be located easily. Content distribution focuses on efficient data dissemination.

The DHT applications are intended to highlight application level considerations of P2P systems. In some cases, some of these applications will be further discussed in the security sections. In others, the application survey will highlight common issues which will come up later in the security section.

3.1 File Sharing

File sharing is the most successful and widely used among the various kinds of DHT applications. File sharing provide a way to distribute large amounts of data with P2P protocol thus the costs of hardware, hosting and bandwidth resources are distributed among the peers, reducing the cost for the original distributor at the same time providing redundancy against failure, and reducing dependence upon the original distributor.

There are several P2P file sharing software and protocols such as BitTorrent, eDonkey2000, Gnutella, FastTrack, etc. Some of them require a central server or a few servers, others can be purely distributed. The original BitTorrent, for example, requires a central server, also known as the tracker. Later, DHT was adopted into BitTorrent and the tracker became unnecessary. In this section will see how DHT helps file sharing protocols by keeping track of the status among peers as well as the items.

3.1.1 BitTorrent

In the original form of BitTorrent, in order to publish a file in BitTorrent, a tracker must be setup for the peers to find each other and coordinate the file transfer. More details of BitTorrent can be found in Section 4.5. Here, we discuss the applications of DHTs in BitTorrent.

The use of a centralized tracker is obviously a single point of failure and bottleneck. With help of DHT, the need for a centralized tracker can be eliminated. In this case, the tracker-less torrents uses the Kademlia DHT. The information of all the peers downloading a torrent is stored in a Kademlia node whose ID is “closest” to the hash of the torrent.

When a node wants to find peers for a torrent, it uses the distance metric to compare the infohash of the torrent with the IDs of the nodes in its own routing table. It then contacts the nodes it knows about with IDs closest to the infohash and asks them for the contact information of peers currently downloading the torrent. If

a contacted node knows about peers for the torrent, the peer contact information is returned with the response. Otherwise, the contacted node must respond with the contact information of the nodes in its routing table that are closest to the infohash of the torrent. The original node iteratively queries nodes that are closer to the target infohash until it cannot find any closer nodes. After the search is exhausted, the client then inserts the peer contact information for itself onto the responding nodes with IDs closest to the infohash of the torrent.

3.1.2 GNUNet

GNUnet is a framework for decentralized, secure peer-to-peer networking. The primary application is anonymous, censorship-resistant file-sharing, allowing users to anonymously publish or retrieve information of all kinds. Anonymity is achieved by obfuscating requests and responses through means of encryption and indirections. We will discuss anonymity and censorship-resistant issues in Section 6.5. Properties of the content encoding and the routing protocol allow GNUnet to reward contributing peers with better service using an excess-based economic model for resource allocation. Peers in GNUnet monitor each others behavior with respect to resource usage; peers that contribute to the network are rewarded with better service. We will look at free riding attacks and reputation systems in Section 6.3.2 and Section 6.6.

GNUnet's DHT implementation is based on ideas from Kademlia. There are various differences between Kademlia and the implementation in GNUnet. The primary difference results from the fact that GNUnet extends the protocol with subtables. Other differences result from adaptations of the original work to GNUnet.

3.2 Storage Systems

3.2.1 PAST

PAST [62] is a peer-to-peer archival storage system implemented using the Pastry DHT. Since PAST is used for archiving, files are immutable. Replicas of a file are stored in the nodes whose IDs are "near" the file's ID. File can be *inserted*, *looked up* and *reclaimed*.

Each PAST node and each user of the system hold a smartcard (although read-only clients do not need a card). A private/public key pair is associated with each card. Each smartcard's public key is signed with the smartcard issuer's private key for certification purposes. The smartcards generate and verify the various certificates and they maintain storage quotas. We can see that the node identities and quota management is done through the smartcard. We will see the two issues in Section 6.1 and Section 6.3.1

3.2.2 Ivy

Ivy [50] is a multi-user read/write peer-to-peer file system. Ivy has no centralized or dedicated components, and it provides useful integrity properties without requiring

users to fully trust either the underlying peer-to-peer storage system or the other users of the file system.

An Ivy file system consists solely of a set of logs, one log per participant. Ivy stores its logs in the Chord DHT (Ivy could in principle work also with other DHTs). Each participant finds data by consulting all logs, but performs modifications by appending only to its own log. This arrangement allows Ivy to maintain meta-data consistency without locking. Ivy users can choose which other logs to trust, an appropriate arrangement in a semi-open peer-to-peer system.

Ivy presents applications with a conventional file system interface. When the underlying network is fully connected, Ivy provides NFS-like semantics, such as close-to-open consistency. Ivy detects conflicting modifications made during a partition, and provides relevant version information to application-specific conflict resolvers. Performance measurements on a wide-area network show that Ivy is two to three times slower than NFS.

3.2.3 CFS

The Cooperative File System (CFS) [20] is a peer-to-peer read-only storage system that provides provable guarantees for the efficiency, robustness, and load-balance of file storage and retrieval. CFS does this with a completely decentralized architecture that can scale to large systems.

CFS servers provide the DHash DHT for block storage. CFS clients interpret DHash blocks as a file system. DHash distributes and caches blocks at a fine granularity to achieve load balance, uses replication for robustness, and decreases latency with server selection. DHash finds blocks using the Chord location protocol, which operates in time logarithmic in the number of servers.

CFS bases quotas on the IP address of the publisher. More precisely, each IP address is given a quota on each storage node. For example, each CFS node limits any one IP address to using 0.1% of its storage. CFS does not support an explicit delete operation. Publishers must periodically refresh their blocks if they wish CFS to continue to store them. A CFS server may delete blocks that have not been refreshed recently. More security issues on storage fairness and quota management is discussed in Section 6.3.1.

3.2.4 Keso

Keso [10] is a distributed and completely decentralized file system based on the peer-to-peer overlay network DKS. The main goals for the design of Keso has been that it should make use of spare clients' resources, avoid storing unnecessarily redundant data, scale well, be self-organizing and be a secure file system suitable for a real world environment.

Keso provides means for access control and data privacy despite being built on top of untrusted components. The file system utilizes the fact that a lot of data stored in traditional file systems is redundant by letting all files that contains a data block with the same contents reference the same data block in the file system.

(CFS also uses the same kind of data block sharing idea). This is achieved while still maintaining access control and data privacy.

3.2.5 Mnemosyne

Mnemosyne [33] is a peer-to-peer steganographic storage service. Mnemosyne provides a high level of privacy and plausible deniability by using a large amount of shared distributed storage to hide data. Blocks are dispersed by secure hashing, and loss codes used for resiliency.

3.2.6 PeerStore

Backup is cumbersome. To be effective, backups have to be made at regular intervals, forcing users to organize and store a growing collection of backup media. PeerStore [46] allows the user to store his backups on other people's computers instead. The system consists of two layers: metadata layer and symmetric trading layer.

Metadata management is accomplished by using a DHT. By storing the metadata records this way, duplicate detection can be done efficiently. At the same time no real data needs to be migrated when nodes join and leave the network; only the information contained in the metadata records needs to be transferred and updated which largely saves the maintenance cost. Data storage, on the other hand, relies on a symmetric trading scheme. A peer that wants to backup its data must also store some data from each of its trading partners.

By decoupling the metadata management from data storage, the system offers a significant reduction of the maintenance cost and preserves fairness among peers. PeerStore also realizes fairness because of the symmetric nature of the trades.

3.2.7 OceanStore

Oceanstore [44] provides a large-scale, incrementally-callable persistent storage facility for mutable data objects with long-term persistence and reliability in a constantly changing network and computing resources. It's intended to be used as an implementation of NFS file service, email service (MINO), web caching (Reptide), databases (Palm database), Anonymous file storage (Nemosyne) and other applications involving persistent storage of large number of data objects. Privacy and integrity are achieved through encryption and Byzantine agreement protocol for updates to replicated objects.

3.3 Host Discovery and Mobility

Previous applications focus on data storage and sharing. DHTs can be applied on host discovery or to support mobility. Participants in a system may be identified using a permanent symbolic name instead of its network address. There must be a mechanism to translate the symbolic name in to the network address. The translation is known as host discovery. In case that the network address is dynamic,

host discovery can be used to provide mobility. For example, the identity of a web server is usually represented as its DNS name. In this case, the DNS protocol provides both host discovery and mobility.

In this section we will look at some systems which use DHT to provide host discovery and mobility.

3.3.1 HIP

An IP address describes a topological location of a node in the network. The address is used to route the packet from the source node to the destination. At the same time the IP address is also used to identify the node, providing two mixed functions in a same thing. This works well with static network topology. However when network topology changes, (for example, links are added/removed, or hosts are moved to other locations) It is difficult to achieve both stability and dynamic changes at the same time.

The Host Identity Protocol (HIP) [39] is proposed to separate the location and identity roles of IP addresses by introducing a new name-space, the Host Identity. In HIP, the Host Identity is basically a public cryptographic key of a public-private key-pair. The public key identifies the party that holds the only copy of the private key. A host possessing the private key of the key pair can directly prove that it "owns" the public key that is used to identify it in the network. The separation also provides a means to handle mobility and multi-homing in a secure way. Nikander et al. [53] proposed the Host Identity Indirection Infrastructure (Hi^3) which uses DHT to implement HIP.

3.3.2 P6P

P6P [75] is an another host discovery system built on top of a DHT overlay. P6P helps the migration of IPv6 by translating an IPv6 address into an IPv4 address. One can consider the IPv6 address as the Host Identity in HIP or the domain name in DNS. Thus the key in the DHT is the IPv6 and the value is the IPv4 address.

3.3.3 SIP

DHTs can be used in IP telephony systems to map an IP phone number to an host's IP address. Kundan Singh et al. [64] proposed an distributed version of the Session Initiation Protocol using DHTs. In their system, peers register their identities, known as "screen names", by putting their IP address under the hash of their identities in the DHT. Peers call other peers by looking up their IP address in the DHT.

3.4 Indirection Services

3.4.1 Internet Indirection Infrastructure

Attempts to generalize the Internet's point-to-point communication abstraction to provide services like multicast, anycast, and mobility have faced challenging tech-

nical problems and deployment barriers. The Internet Indirection Infrastructure (*i3*) [68] offers a rendezvous-based communication abstraction in order to ease the deployment of such services. Instead of explicitly sending a packet to a destination, each packet is associated with an identifier; this identifier is then used by the receiver to obtain delivery of the packet. This level of indirection decouples the act of sending from the act of receiving, and allows *i3* to efficiently support a wide variety of fundamental communication services.

3.4.2 Untangling the Web from DNS

The Web relies on the Domain Name System (DNS) to resolve the hostname portion of URLs into IP addresses. While the use of DNS has made URLs easy to use, the Web is also constrained by the limitations of the web. Semantic Free Referencing (SFR) [71] is a reference resolution infrastructure for URLs based on DHTs which provides location and contention independence.

3.5 Web Caching and Web Servers

3.5.1 Squirrel

Squirrel [37] is a distributed web cache system designed to share the web caches of participating users. It uses the Pastry DHT to store cached Web objects and directory information. When a user requests a file, Squirrel contacts the member of a DHT who should be the owner of that file, and gets it from that member (referred to as home-store). Alternately, it can lookup, from that "owning" member, and return a list of other users who have recently downloaded the file, and get it from them (referred to as directory-store).

3.5.2 FeedTree

FeedTree [4] is a peer-to-peer system for distributing Web feeds. Instead of polling feeds from the publisher independently, FeedTree uses the Scribe multicast in Pastry to disseminate Web feeds. Comparing to traditional Web feeds, FeedTree saves the bandwidth of the publisher.

3.6 Content distribution

3.6.1 SplitStream

SplitStream [14] is a high-bandwidth content distribution system based on application-level multicast. It distributes the forwarding load among all the participants, and is able to accommodate participating nodes with different bandwidth capacities. The key idea in SplitStream is to split the content into k stripes and to multicast each stripe using a separate tree. Peers join as many trees as there are stripes they wish to receive and they specify an upper bound on the number of stripes that they are willing to forward. The challenge is to construct this forest of multicast trees such that an interior node in one tree is a leaf node in all the remaining trees and

the bandwidth constraints specified by the nodes are satisfied. This ensures that the forwarding load can be spread across all participating peers. For example, if all nodes wish to receive k stripes and they are willing to forward k stripes, SplitStream will construct a forest such that the forwarding load is evenly balanced across all nodes while achieving low delay and link stress across the system.

3.6.2 Coral

CoralCDN [29] is a peer-to-peer content distribution network that allows a user to run a web site that offers high performance and meets huge demand, all for the price of a cheap broadband Internet connection. Volunteer sites that run CoralCDN automatically replicate content as a side effect of users accessing it. Web viewers can access the webpage by appending "`.nyud.net:8080`" to hostname of a URL. For example, one can use `http://www.cnn.com.nyud.net:8080/index.html` to view `http://www.cnn.com/index.html`. A peer-to-peer DNS layer transparently redirects browsers to nearby participating cache nodes, which in turn cooperate to minimize load on the origin web server. One of the system's key goals is to avoid creating hot spots that might dissuade volunteers and hurt performance. It achieves this through Coral, a latency-optimized hierarchical indexing infrastructure based on a novel abstraction called a distributed sloppy hash table, or DSHT.

The Coral DSHT differs from DHTs in several ways. First, Coral's locality and hot-spot prevention properties are not possible for DHTs. Second, Coral's architecture is based on clusters of well-connected machines. Clusters are exposed in the interface to higher level software, and in fact form a crucial part of the DNS redirection mechanism. Finally, to achieve its goals, Coral provides weaker consistency than traditional DHTs.

3.7 Naming systems

3.7.1 CoDoNS

CoDoNS [2] is a high-performance, failure-resilient, and scalable name service for the Internet. It serves as both an alternative and a safety-net for the legacy Domain Name System (DNS). Built on top of Beehive (a general replication framework that operates on top of any DHT that uses prefix-routing), it provides clients low latencies for name resolution, automatic load-balancing during flash-crowds and denial of service attacks, and quick dissemination of changes in DNS mappings. It is currently deployed across the globe on Planet-Lab.

3.7.2 SENS

SENS [52] is a scalable and expressive naming system which can retrieve information of computing and content resources distributed widely on the Internet by exact queries and multi-attribute range queries over resource names. SENS utilizes a descriptive naming scheme to name resources and a multi-dimensional resource ID space for message routing through the overlay network of name servers (NSs).

The resource ID space is constructed on the overlay network based on CAN routing algorithm. For example, the resource ID representing a computer can be: (string OS = "linux", string CPU-name = "Pentium 4", int CPU-clock = 1000, int memory = 1024, int harddisk-unusedspace = 20, int network-bandwidth = 1000). And a query can be: (string OS = "linux", string CPU-name = "Pentium 4", int CPU-clock $\geq$ 1000 and int CPU-clock $\leq$ 1200, int memory $\geq$ 512, int harddisk-unusedspace $\geq$ 10, int network-bandwidth $\geq$ 100). It uses multicast routing algorithm to deliver resource information and a broadcast routing algorithm to route query messages to corresponding NSs at minimum cost.

3.8 Communication

3.8.1 POST

POST [49] is a decentralized messaging infrastructure that supports a wide range of collaborative applications, including electronic mail (ePOST), instant messaging, chat, news, shared calendars and whiteboards. POST is built on top of Pastry overlay network with desktop computers as the peers. POST offers three simple and general services: (i) secure, single-copy message storage, (ii) metadata based on single-writer logs, and (iii) event notification.

POST assumes the existence of a certificate authority. This authority signs certificates binding a user's unique name (e.g., her email address) to her public key. The POST infrastructure allows users to create messages and insert them in encrypted form into the system. To send a message to another user or group, the notification service is used to provide the recipient(s) with the necessary information to locate and decrypt the message. The recipients are then notified on message arrival.

3.9 Query and indexing

3.9.1 XPeer

XPeer [56] is an XML-based content query system built on DHT systems in this case XPeer is based on Pastry. XPeer utilizes XML to implement content-based query using XPath as the query language so that it can query more than just the simple keyword searching. XML data in XPeer can be totally heterogeneous and support range query over DHT.

3.9.2 PIER

The largest database systems in the world scale up to at most a few hundred nodes which is a lack of scalability. Database technology has not become "an integral part" of massively distributed systems like the Internet.

PIER [36] is designed as a query engine that comfortably scales up to thousands of participating nodes. It is built on top of a DHT. PIER presents a "technology

push" and "application pull" for massive distribution: the querying of Internet-based data *in situ*, without the need for database design, maintenance or integration.

3.10 Chat services

DHT can be used as chat services to bootstrap the friends, finding and sending messages to people who are not currently online, create channels, etc. Example of chat services applications are Retroshare Instant Messenger [5] and The Circle [1].

3.11 Application-layer multi-casting

3.11.1 Bayeux

The demand for streaming multimedia applications is growing at an incredible rate. Bayeux [77] provides an efficient application-level multicast system that scales to arbitrarily large receiver groups while tolerating failures in routers and network links. Bayeux also includes specific mechanisms for load-balancing across replicate root nodes and more efficient bandwidth consumption. The simulation results indicate that Bayeux maintains these properties while keeping transmission overhead low. To achieve these properties, Bayeux leverages the architecture of Tapestry.

3.11.2 Scribe

Scribe [15] is a scalable application-level multicast infrastructure. Scribe supports large numbers of groups, with a potentially large number of members per group. Scribe is built on top of Pastry, and leverages Pastry's reliability, self-organization, and locality properties. Pastry is used to create and manage groups and to build efficient multicast trees for the dissemination of messages to each group. Scribe provides best-effort reliability guarantees, and has extensions for applications to provide stronger reliability. Simulation results, based on a realistic network topology model, show that Scribe scales across a wide range of groups and group sizes. Also, it balances the load on the nodes while achieving acceptable delay and link stress when compared to IP multicast.

3.12 Search Engines

Search Engine applications can be built on top of DHT allowing anonymous, uncensored search in a distributed network of search-engine peers. Peers automatically or manually crawled the Internet and every web page visited will be automatically included in the distributed index. Rank could be assigned based on the distributed usage statistics of the web page visited by users. Examples of these applications are YaCy [6] and FAROO [3].

4 Other P2P systems

This section surveys other popular P2P applications which are not necessarily based on DHTs and do not necessarily deal with some of the scalability, dynamism and fault tolerant issues. Nevertheless, given that these are significant as they are important P2P applications. We have selected P2P systems which are relevant to the security issues in this survey.

4.1 Napster

The earliest P2P application, Napster, was built to capitalise on the interest of users in getting mp3 files from the Internet. Before Napster, the main distribution mechanism for mp3 files was using client/server architecture: users upload their mp3 files on a server/website and other users search for the website and content there. Since the process of finding websites and content was not very reliable and often search results had broken links, a common mp3 search engine and file sharing like Napster became popular very rapidly.

Napster introduced a new way for distributing mp3 files by having the mp3 files stored on the users computers and maintaining a central server that lists the locations of users that have a particular mp3 files. Thus, a user (peer) can locate other peers who have the mp3 files by querying the central server. The actual file is transferred directly between the peers.

The architecture of Napster is a hybrid P2P since there is the centralized server which still behaves more like a client-server and it was only the content transfer itself which was between the peer machines. Napster did not attempt to directly address P2P issues like scalability, bandwidth management, load balancing, fault tolerance, etc. However, the demise of Napster was less because of technical issues but stemmed rather from the legal problems of copyrighted content served by the Napster index.

From a technical perspective, the problem which Napster did not deal with and led to its demise was that it did not address P2P security issues which arise from P2P systems. The primary ones which arise naturally in a content distribution context are anonymity and censorship resistance. For example, Napster's centralized index lists which users have what mp3 files.

4.2 Gnutella

As Napster became popular, they faced the problem that the content residing on Napster indexes included copyrighted material. Eventually Napster had to shut down its network due to legal challenges. Gnutella gained popularity from the demise of Napster as it providing anonymity to its users.

Gnutella works in a decentralized way without a central index. A peer (node) that wants to join a Gnutella network first finds a live node in the network and connects to it (the bootstrap phase). After it is connected it will broadcast about its existence by using a flooding to the nodes connected to it. The same technique

is applied when a peer wants to search for a file in the network, The search query is broadcasted into the network as a flood request. Nodes that receive the flood request can forward it to other reachable nodes. It can also consume/process the request. Messages have a message identifier which is used to prevent loops. A time-to-live field in messages which is decremented at every hop limits the spread of messages in the network.

Anonymity is achieved by through deniability: no node knows who requested the information since decentralization makes it difficult to determine whether a user requested the data for himself or simply requested the data on behalf of another user. Every node in the network acts as a universal sender and universal receiver to maintain anonymity. The anonymity can be broken if all nodes cooperate to break it, e.g. by saving local logs and correlating them.

This protocol initially could not scale to handle large number of peers but later it was improved by promoting some nodes as *Ultrapeers*. Ultrapeers work by connecting to other Ultrapeers and routing search and query hit packets. The idea is that the Ultrapeers' internet connection is sufficiently fast that it can serve to route search for a number of non Ultrapeers (called Leaf nodes) connected to it. A leaf node keeps only a small number of connections open to Ultrapeers. An Ultrapeer acts as a proxy to the Gnutella network for the leaf nodes connected to it.

A drawback of Gnutella is that search results are often unreliable. Nodes that are far away and have rare items cannot be reached from certain nodes because of the timeout limit. The bandwidth cost of searching on Gnutella is inefficient as it grows quadratically to the number of connected nodes. Slow connections are also a problem and may cause search requests to be dropped.

Gnutella has a number of security problems when it comes to malicious nodes. Malicious nodes can spoof responses, advertise inaccurate content and topology information. Queries can also be easily spoofed since requests are anonymous, which leads to DoS flooding attacks.

4.3 FastTrack

FastTrack is similar to Napster where the central server is replaced by powerful nodes (supernodes) to improve scalability. The supernode functionality is built into the client. If a powerful computer with a fast network connection runs the client software, it will automatically become a supernode, effectively acting as a temporary indexing server for other, slower clients.

The bootstrap process contains a list of supernode IP addresses to be contacted stored in the program and updates the supernodes list if possible. Nodes connected to a supernode upload a list of files it intends to share to that supernode and they also send search requests to that supernode. The supernode communicates with other supernodes in order to process search requests. Nodes then can connect directly to a hosting node to download the file.

One major vulnerability is the use of UUHash as a quick checksum for files. The UUHash algorithm is flawed as it only covers a fraction of the file. This makes it trivial to create a hash collision which allows a large portion of the file to be altered

without altering the checksum. Thus it is easy to corrupt the FastTrack network.

Applications that uses FastTrack network are Kazaa and its variants. Another security problem is that many of these applications are usually bundled with some form of malware.

4.4 Freenet

Freenet [17] provides a decentralized file-storage. It concentrates more on providing anonymity and security. Freenet was designed to work in untrustworthy and unreliable P2P environment with assumption that participants can operate maliciously or fail without warning at anytime thus Freenet implements strategies to protect data integrity and prevent privacy leaks and also provide graceful degradation and redundant data availability.

Privacy in Freenet is maintained using a variation of Chaum's mix-net scheme [16] for anonymous communication. Messages travel trough node-to-node chains with each link is individually encrypted until the message reaches the recipient. Routing in Freenet uses a steepest-ascent hill-climbing search: it forwards request to node that is believed to be the closest to the target. It's similar with Gnutella flooding search, except in Freenet the flood is not a blind flood but using a routing table with a key closeness heuristic.

Each node on the search route stores a replica of the file to be able to process future search queries more quickly. When node storage exceeds the capacity, files are deleted according to the least-recently used principle. This results in a correspondingly large number of replicas of popular files being created in the network, whereas, over time, files which are requested less often are removed.

Freenet searches/stores files within the network using a document routing model. Files are not stored in the providing peers but are intentionally stored at other locations in the network. The reason is to create a network in which information can be stored and accessed anonymously. This requires that the owner of a node does not know what documents are stored on the local storage of the node. The document routing model has been shown to be suitable for use in large communities, however it can result in network partitioning.

4.5 BitTorrent

BitTorrent was designed as a P2P file sharing system to cope with flash crowds, for example, a new popular file can be in high demand very quickly but after some time the demand dies down. In its original form, BitTorrent was a hybrid P2P system. BitTorrent used centralized servers called trackers which manage the file sharing process. To participate in a BitTorrent network to share files (either to download or upload files), a "torrent" which contains the information about the file, its length, name, hash, and the tracker's URL file must be created. Peers can get together in the same network by using the "torrent" file and locate the tracker. The tracker keeps track of which peers are uploading/downloading the file and whether they contain partial or complete copies of the file.

The file distribution protocol uses "equivalent retaliation" (tit for tat): a node using this strategy will initially cooperate, then respond in kind to other node's previous action. If the other node previously was cooperative, the node is cooperative. If not, the node is not. The protocol is designed to discourage free riding by having the peers respond to other peers that send some data (cooperation among peers). Thus peers which upload more content (portions of files) to other peers are more likely to also download more content.

Since TCP congestion control behaves poorly when a node sending too many connections, flow control is needed by choking the file transfer process. Choking also helps in achieving tit-for-tat protocol ensuring peers to have consistent download rate. Choke algorithm is designed to keep good TCP performance and avoid fibrillation (frequent choking and un-choking) and also should reciprocate to peers who are cooperating.

As the torrent file contains a hash of the contents, BitTorrent has some self-certification of authenticity of file contents as long as the tracker information is correct. This makes it harder to corrupt the network.

More recent BitTorrent implementations take advantage of DHTs to create a tracker-less BitTorrent system (see Section 3.1.1).

5 General security issues

5.1 Comparing to traditional computer security

In this section, we look at security issues in P2P (distributed) systems by comparing them to their counterparts in traditional server-client (centralized) systems. Confidentiality, integrity and availability are three fundamental objectives of security. We will categorize the security issues in P2P systems into these three fundamental security objectives.

5.1.1 Confidentiality

Generally speaking, confidentiality is assurance of data privacy. Only the intended and authorized recipients: individuals, processes or devices, may read the data. In the P2P context, there are some system, such as Freenet [17] and Tarzan [28, 30] which focus on the confidentiality of their users. These systems provide anonymous storage or anonymous communication channels where participants cannot be easily identified. There are also hybrid (partially distributed) systems, such as the onion routing [31], which provides the same functionality. Interestingly, the hybrid system can be modified to a distributed system by replacing the server with DHT-based peers. However, there are some security precautions which need to be taken into account with this modification. We will look at them in Section 6.5.

5.1.2 Integrity

Integrity ensures that the information is authentic and complete. In DHTs, for example, a "get" on a value of a key should produce the value of the most recent "put" on the key. In P2P storage systems, the receiver should download the file uploaded by the publisher and not anything else. Because the adversary model in a P2P system is different from a centralized system, the mechanism used to ensure integrity is also different. In a centralized system, clients implicitly have to trust the integrity of the server. In a P2P system, on the other hand, it might not be reasonable for participants to make any assumptions about the integrity of other participants unless there are reasons to do so. Integrity of participants in a P2P system can be achieved with a reputation system, while integrity of data can be achieved by replication. In the real world, trust is often based on personal or corporate reputations. The higher the reputation of an entity, the more trustworthy and reliable it is believed to be. The same idea can be applied on P2P systems. For example, in P-Grid [38], the reputations are expressed as complaints. The number of complaints a peer receives is inversely proportional to its trustworthiness. We will look at reputation systems in Section 6.6. Replication can be used to ensure the integrity of data. In most DHT systems, each key-value entry can be replicated onto a number of nodes. One may retrieve the values from all replicas and use a majority vote to increase the likelihood of getting the correct value.

5.1.3 Availability

An significant advantage of P2P system is availability or fault tolerance. Thus many of P2P systems are designed to maximize availability. There are a few security issues related to availability:

- **Sybil Attack**
  The security of P2P systems often relies on assumptions that most of the participants are legitimate. Even when one assume that most users (people) are legitimate, a problem can still arise because users and participants do not have one to one correspondence. A malicious user may appear as many participants in the system. We will expand on this attack (also known as the Sybil attack) in Section 6.1.

- **Robust Routing**
  In a P2P system, each node only knows a fraction of the whole system. This fraction is usually called the node's neighbor set. Even when most of the nodes are correct, it may be possible for a small number of malicious nodes to dominate the neighbor sets of other correct nodes. This means that the correct nodes can be isolated from the system or even controlled by the malicious nodes. We will expand on this attack (also known as the Eclipse attack) in Section 6.2.

- **Redundancy**
  Redundancy plays an important role in fault tolerance. Not only data can be replicated, query, routing, can also be replicated. We will see that redundancy appears in many forms in the following discussion.

- **Fairness & Load Balance**
  In an ideal case of distributed file sharing systems, a file that is downloaded by a peer is automatically opened for sharing with other peers. However, peers can, and frequently do, turn off this property and stop sharing a downloaded file to economize on their own resources such as bandwidth. Therefore, the primary advantage of P2P file sharing systems, the implicit or explicit functional cooperation and resource contribution of peers, may fail and lead to a situation called *free riding*. We will look at some countermeasures in Section 6.3.

  There are also other approaches such as the Collective Intelligence (COIN) framework. for designing better mechanisms for maximising global utility given self-interested agents. This has been shown to be a promising approach to better align agent utilities and avoid free riding for a number of distributed problems.[1]

[1] A mini-course on collective behavior and agoric systems was conducted to to introduce the SELFMAN project to COIN.

- **Censorship Resilience**
  It may be not difficult to block a single server or identify and block documents for centralized or well known servers. P2P systems however can be much harder to block as they lack a single point of attack to block or destroy documents. In Section 6.5 we will look at some censorship resilience systems.

5.1.4 Assumption and Effectiveness

The adversary model of traditional centralized systems is very different from decentralized systems. In centralized systems, the border between trusted and untrusted entities is clear. In decentralized systems, it is usually not clear whether a peer can be trusted. Trustworthy can sometimes be measured probabilistically.

In centralized systems, the server is usually assumed to be trusted. For example, in PKI, the certificate authority is assumed to be trustworthy. In decentralized systems, usually we assume at least some fraction of peers are trusted. For example, in a distributed file sharing system with replication, more than half of the nodes need to be correct in order to produce retrieve the correct file.

Since the adversary model is probabilistic, the correctness of a distributed system is also statistical. For example, given that 7/8 of the nodes are correct in a DHT, a key lookup with 4-hop route will succeed with probability 58.6%.

5.2 Attacker Incentives

Over the last few years, attacks as a means of financial gain are becoming more common.[2] Since money is involved, the attackers tend to have larger resources at their disposal. For instance, media owners can try to pollute the content in the overlays by putting up nodes with corrupt chunks of data but with correct file names to degrade the service. One intent would be to frustrate users and get them to stop using the service. An example is *MediaDefender* which is a company which offers services designed to prevent and stop people who engage in peer-to-peer copyright infringement. The tactics used include flooding peer-to-peer networks with decoy files that tie up a user's computer. MediaDefender is estimated to have between 2000 servers around the world using 9GBps of bandwidth. Similar companies include OverPeer and MediaSentry.

[2]This does not necessarily refer to criminal activities but merely activities which result in monetary incentives.

6 Security in P2P Systems

We will now examine various specific security related attacks and defenses in P2P systems.

6.1 Sybil Attack

A *Sybil attack* [25] is a attack where an attacker can present multiple identities, and uses them gain a disproportionately large influence in the system.[3] Although the Sybil attack in the P2P context has been recently named by Douceur, this flavour of attack appears in many forms both in academia and in the real world. For example, it is possible to rig Internet polls by using multiple IP addresses to submit votes. Trying to manipulate the Google Pagerank rating of a page is also another form of Sybil attack.

Many P2P systems assume an upper bound on the fraction of corrupted participants. For example, in the Byzantine Generals Problem, the number of traitors must be less than one third of the total number of generals. Many systems replicate computational or storage tasks among several remote sites to protect against integrity violations (data loss). If a certain fraction of the replicas are corrupted, the system cannot provide the correct data. Others fragment tasks among several remote sites to protect against privacy violations (data leakage). If a certain fraction of the fragments are controlled by a single entity, that entity is able to discover the data by itself.

Levine et al. [48] surveyed 90 papers on Sybil attack and categorized then into eleven categories. Approximately half of the published papers either suggest certification as a solution to the Sybil attack, following [25]'s approach, or simply state the problem without giving a solution. He concluded that there is no general solution to the Sybil attack, but there are a variety of solutions that can limit or prevent the attack in several individual application domains.

In the following sub-sections, we look at different approaches to prevent the Sybil attack.

6.1.1 Centralised identifier assignment

In this approach, participants use an identity authority, a trusted common third party, to authenticate other participants' identity. Usually the authentication is done using public key infrastructure (PKI). This approach has several problems:

- The identity authority has to ensure each entity is assigned exactly one identity. However, this may itself be an issue for P2P systems.
- It uses a centralised mechanism which results in a single point of failure.
- The identity authority can be the performance bottleneck as it needs to be contacted during enrollment, revocation checking.

[3] It is named after the subject of the book "Sybil" by Flora Rheta Schreiber, a case study of a woman with multiple personality disorder.

Castro et al. [13] suggests the use of certificate authorities (CAs) to issue certificates of identities. They propose two ways to prevent the attacker from easily obtaining certificates of identities. One way is to require an attacker to pay money for certificates. The money can also fund the operation of the CAs. The other way is to bind node identities to real-world identities. However, both of the methods raise a barrier which discourages users from joining the system.

6.1.2 ID Based On the Network Address

One way to prevent the Sybil attack is to only allow one identity per IP address (or using the IP address as the identity). The reason of choosing the IP address is i) that it is easy to verify the participant's IP address[4] and ii) the fact that attacker usually has only a few IP addresses. This approach has several problems:

- The attacker can control many IP addresses, e.g a zombie network.[5]
- In IPv6 nodes can obtain a huge number of IP addresses, e.g. through using the privacy extensions defined in *RFC3041*.
- Computers behind Network Masquerading or Network Address Translation (NAT) share IP addresses with other computers in the same NAT domain. If one identity per IP address is required, only one of the computers in the same NAT domain can join the system.
- Dynamic IP causes a problem when the IP address itself is used as the identity.

The Chord DHT [66, 67] uses a hash of the node's IP address as the node identity.

In the Tarzan system, Freedman et al. [28, 30] observes that a single machine likely controls only a contiguous range of IP addresses, typically by promiscuously receiving packets addressed to any IP address on a particular LAN or by acting as a gateway router. Therefore, when selecting relays, one should consider distinct prefix of IP addresses (the first n bits of the address), instead of distinct IP addresses.

6.1.3 Resource Testing & Cryptographic Puzzles

If we assume the attacker has limited computing power,[6] cryptographic challenges can be used prevent Sybil attacks. In this approach, each node challenges it peers by sending cryptographic puzzles and expecting correct solutions within a time threshold. A genuine node should have no difficulty computing the solution. However, a Sybil node receives a large number of challenges and is assumed not to be able to compute all the solutions. The challenges can be performed simultaneously

[4] Of course, there is IP spoofing, but it is another issue.

[5] A zombie network is a collection of zombie computers which have been compromised, e.g. by a virus.

[6] Besides computing power, other resources include bandwidth and storage. However those resources are rarely used for testing purposes. One can consider also IP addresses as the resource. In this case, it broadens this approach to include the previous one.

and periodically, randomly, or during node joining. Calculating the inverse of a cryptographic hash function[7] is commonly used as the puzzle.

Douceur [25] model this approach into two categories: accept identities which have been directly challenged (direct test) and accept identities vouched for by other already accepted identities (transitive accept). Douceur proves that both are ineffective:

- For the "direct test" case, each entity has to simultaneously challenge its peers. A faulty entity can counterfeit a constant number of multiple identities. Otherwise, if it is not simultaneous, a faulty entity can counterfeit an unbounded number of identities.

- For the "transitive accept" case, Each entity also has to simultaneously challenges its peers. A single faulty entity can counterfeit a constant number of multiple identities. A sufficiently large set of faulty entities can counterfeit an unbounded number of identities.

One difficulty in the approach is deciding the time threshold on solving the puzzle. The computing power of peers can vary significantly by one or more orders of magnitude. Network delay in the Internet also varies. It is difficult to come up with a time threshold which permits genuine users and limits the Sybil identities.

Aspnes et al. [11] propose two algorithms: *Democracy* and *Monarchy*. In Democracy, each node i first broadcasts a random string s_i to every other node. Each node, after collecting all the the s_i, calculates as many x_j as possible, where the first w bits of $Hash(s_1|s_2|...|s_n|x_j)$[8] are 0. Each node then broadcast all its x_j and validates the x_j received from other nodes. The Monarchy algorithm is different from Democracy in the sense that the challenges are separated into n rounds instead on one round. In each round, one node, called the *king node*, challenges every other node. The reason for separating the challenges is that the cost of solving the hash puzzle in Democracy is not deterministic. In Monarchy, the time-lock puzzle [60] which has a fixed running cost can be used. Both of the algorithms cannot scale well because each node sends and receives $O(n)$ messages, and in total there will be $O(n^2)$ messages.

Rowaihy et al. [61] propose an admission control system to mitigate Sybil attacks by adaptively constructing a hierarchy of cooperative admission control nodes. Implemented by the peer-to-peer nodes, the admission control system vets joining nodes via client puzzles. A node wishing to join the network is serially challenged by the nodes from a leaf to the root of the hierarchy. Nodes completing the puzzles of all nodes in the chain are provided a cryptographic proof by the root node. This approach is not fully decentralized, because the root node imposes a single point of failure.

Jaffe et al. [38] developed a protocol called *Sisyphus* to prevent the Sybil attack. There are two roles in the Sisyphus protocol, *player* and *voucher*. Each node plays

[7] Aspnes et al. [11] suggest using problems such as the time-lock puzzle [60] which cannot be computed faster in parallel.

[8] "|" denotes string concatenation.

a *player* role and may or may not play a *voucher* role. Each *player* is challenged by k *vouchers*. The k *vouchers* for the *player* is determined by the IP address of the *player*, e.g. generate k random node IDs using the *player*'s IP address as the random seed. When node A wants to determine whether node B is a Sybil node, A will consult B's *vouchers*. They claim that when the random k *vouchers* are uniformly generated, an adversary cannot become its own *voucher*.

6.1.4 Social Networks

Yu et al. [74] proposed a protocol named SybilGuard to prevent Sybil attacks using a social network. In the social network, an edge between two nodes indicates a human-established trust relationship. While malicious users can create many Sybil identities and it might even be easy to do so, the same does not hold for trust relationships. The assumption is that malicious users can only create a few trust relationships. The basic insight is that if a malicious users create too many Sybil identities, the graph becomes "strange" in the sense that it has a small *quotient cut* i.e. a small set of edges (the attacker's human relationships, also called attack edges) whose removal disconnects a large number of nodes (all the Sybil identities created by the attacker) from the rest of the graph.

However, it it not trivial to find the small cut without the complete topology. In SybilGuard, each node performs a random route[9] (starting from itself) of a certain length w (w is suggested to be 2000 for the one-million node topology). The honest node only accepts another node whose random route intersects with the honest node's random route. Because of the limited number of attack edges, with an appropriate w, the verifier's route will remain entirely within the honest region with high probability.

6.1.5 Others Approaches

Instead of assuming a computation bound on attackers, S-Chord [27], a variant of Chord, makes no assumption on their computation power. More precisely, S-Chord assumes there is an omniscient and computationally unbounded adversary controlling the Byzantine peers and that the IP-addresses of all the Byzantine peers and the locations where they join the network are carefully selected by this adversary. The assumption that S-Chord makes on attackers is on the rate which attackers join the network. For any fixed $\epsilon_0 > 0$, it is resilient to $(1/4 - \epsilon_0)z$ Byzantine nodes joining the network over a time period during which: (i) there are always at least z total nodes in the network; and (ii) the number of correct peers joining and leaving is no more than z^k for some tunable parameter k.

6.2 Routing Attacks

A robust routing protocol should ensure the correct message is routed to the correct destination with high probability even when a small fraction of nodes do not follow

[9] In the full protocol, each node performs multiple random routes.

the protocol. Here, we discuss some attacks and defenses to the routing protocol.

Atul Singh et al. [63] defined the *Eclipse* attack[10] and proposed a way to prevent Eclipse attacks. In an Eclipse attack, the attacker controls a large fraction of the neighbor sets of correct nodes, it can "eclipse" correct nodes by dropping or rerouting messages that attempt to reach them. In the extreme, the Eclipse attack provides the attacker with full control over all overlay traffic. One might think that Eclipse attack requires the attacker to have large number of nodes (or identities), thus solving the Sybil attack solves the Eclipse attack. However, even if attackers control only a small fraction of overlay nodes, they may still be able to launch an Eclipse attack by exploiting the overlay maintenance algorithm. For example, in an overlay like Gnutella, nodes replace faulty neighbors with nodes obtained by traversing neighbor links. If the attacker controls a fraction f of the nodes in the overlay, attacker nodes can return other compromised nodes whenever they are asked to for a neighbor and correct nodes may still return a compromised node with probability at least f. Therefore, the fraction of neighbors of correct nodes that is controlled by the attacker tends to grow until the attacker has full control over all overlay traffic.

In [63], the "degree bounding" method is proposed to prevent Eclipse attacks against unstructured overlays such as Gnutella where the degree of nodes is not constrained. The degree of a node is also referred to as the size of the routing table or the size of the neighbor set. The basic rational behind the method is: the degree of attacker nodes must be much higher than the average degree of nodes in the overlay during an Eclipse attack. Thus, one way to prevent the Eclipse attack is to bound the degree. The way to bound the degree is to periodically audit the degree of its neighbor. Each node X periodically audits its neighbors for their list of neighbors. If X is not in the list or the list is too large, the neighbor is considered performing an Eclipse attack. Note that auditing needs to be anonymous, i.e. the neighbor does not know that it is X who is auditing. Otherwise, the attacker can simply give a list consisting only the auditor.

Castro et al. [13] proposed *redundant routing* at *routing failure*. They used a general method which can be applied to different DHTs. In the model, each node maintains a routing table with node IDs of other nodes and their associated IP addresses. Moreover, each node maintains a neighbor set, consisting of some number of nodes with node IDs near the current node in the ID space. In addition, data is stored at more than one node in the overlay. A replica function maps an object's key to a set of replica keys. They provided a method to: (i) detect routing failure; and (ii) if the routing is considered failed, redundant routing is applied.

Condie et al. [19, 18] proposed the use of *induced churn* as a defense against Eclipse attacks. Induced churn consists of three techniques: *periodic reset of routing tables* to less efficient but more attack-resistant ones, *forced unpredictable identifier changes*, and *rate limitation on routing table updates*.

Reidemeister et al. [58] proposed three alternative routing protocols for the CAN DHT to defense against Eclipse attacks.

[10]It is also called routing-table poisoning in other papers. Actually, this attack is not specific to DHTs. For example, there are similar attacks to the link-state routing protocol.

1. **Multiple Identities**
   Each node joins as k identities independently. When issuing a query, the node issues the query using all k identities. It is hard for the attacker to eclipse all query routes.

2. **Multi-Path Routing**
   A node issues the query using k independent routes.

3. **Proximity Routing**
   The route is randomized so that it can not be predicted by the attacker.

Danezis et al. [22] proposed a Sybil-resistant routing strategy. In the proposed strategy, lookup routing is implemented iteratively (instead of recursively). When choosing the next hop, both the distance to the destination and the level of trustworthy are considered. In the original Chord, the next hop is the closest node in the finger table to the key. I.e. Only the distance to the destination is considered. However, that node may be an malicious node because either the node id is carefully chosen by the attacker or the finger table is "poisoned" by means of "eclipse" attack. Because the "eclipse" attacker usually appears in many nodes' neighbor-list, the level of trustworthy of a node is measured by the number of times the node appeared in other nodes' neighbor-list. The larger the number, the less trustworthy.

Harvesf et al. [34] proposed an equally-spaced replication scheme and showed that it can be tuned to produce any desired number of disjoint routes. More specifically, they showed that d disjoint routes can be produced by placing 2^{d-1} replicas around a fully populated Chord ring in an equally-spaced fashion. In this situation, there exists a route to at least one replica, which contains only compromised nodes, even if an attacker controls more than a quarter of the contiguous identifier space in the system.

6.3 Fairness & Load Balance

6.3.1 Storage Fairness

In P2P systems which deal with storage, it is obvious that the amount of storage consumed by a given node must be in proportion to the amount of storage space they will provide to the network. Otherwise, free riding will result in too much data for the storage space available. One way to ensure fairness is to have a centralized quota administrator. However this results single point of failure and does not scale well. We shall look at several different decentralized methods.

CFS [20] bases quotas on the IP address of the publisher. For example, if each CFS server limits any one IP address to using 0.1% of its storage, then an attacker would have to mount an attack from about 1000 machines for it to be successful. This mechanism also limits the storage used by each legitimate publisher to just 0.1%, assuming each publisher uses just one IP address.

CFS stores data for an agreed-upon finite time interval. Publishers that want indefinite storage periods can periodically ask CFS for an extension; otherwise, a

CFS server may discard data whose guaranteed period has expired. CFS has no explicit delete operation, instead, a publisher can simply stop asking for extensions. In this area, as in its replication and caching policies, CFS relies on the assumption that large amounts of spare disk space are available.

If a system offers the persistent storage semantics typical of traditional file systems, the system will eventually fill up with orphaned data. OpenDHT [59] offers storage with a definite time-to-live (TTL) which is specified during the *put* procedure. The system discards the data after TTL is expired. OpenDHT uses a storage allocation policy which ensures fairness in the sense that (i) upon overload, each client has equal access to storage; (ii) it prevents starvation by ensuring a minimal rate at which puts can be accepted at all times. The fairness is ensured at individual nodes instead of globally in the whole system. This means a node can take advantage of storing data under many keys (nodes) instead of under one key. In order to prevent starvation, the system always reserves space for future *puts* which is estimated based on the minimal rate. A proposed *put* is rejected if the reservation condition is violated. Figure 1 shows two proposed *puts*, a large one (in terms of the number of bytes) with a short TTL in (a) and a small one with a long TTL in (b). The large-but-short proposed *put* violates the condition, whereas the small-but-long proposed *put* does not.

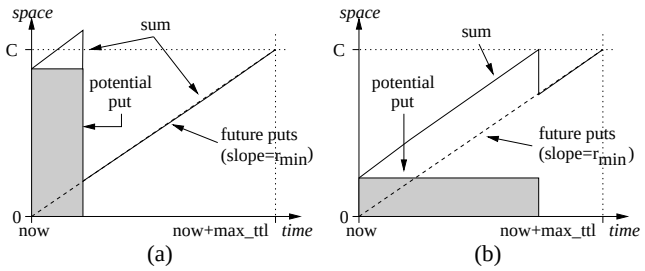

Figure 1: Storage allocation policy used by OpenDHT [59]

Ngan et al. [51] proposed a random peer auditing method to enforce storage fairness. In their model, each node puts a list (called the *remote list*) of files in other remote nodes. From the other node's point of view, each node stores a list (called the *local list*) of file for other nodes. Each node advertises its storage capacity (called the *advertised capacity*) provided to the system. The rule is that a node can put new files into the system only if its *advertised capacity* is larger than the size of its *remote list*. Intuitively, this means if a node provides k bytes to the system, it can put at most k bytes into the system. In order to ensure this, every node X audits the nodes Y in its *local list* by asking them their *remote list* periodically. The file which X stores for Y must appear in Y's *remote list*, otherwise, Y is cheating, and X can delete the file. Note that the auditing must be performed anonymously, i.e. Y does not know it is X who is auditing.

6.3.2 Bandwidth Fairness & Free Riding

In the ideal case, a file that is downloaded by a peer is automatically opened for sharing with other peers. However, peers can, and frequently do, turn off this property and stop sharing a downloaded file to economize on their own resources such as bandwidth. Therefore, the primary advantage of P2P file sharing systems, the implicit or explicit functional cooperation and resource contribution of peers, may be negated. Such selfish behavior leads to a situation called *free riding*.

An analysis [8] of user traffic on Gnutella shows a significant amount of free riding in the system. By sampling messages on the Gnutella network over a 24-hour period, Adar et al. established that almost 70% of Gnutella users share no files, and nearly 50% of all responses are returned by the top 1% of sharing hosts. Furthermore, they found out that free riding is distributed evenly between domains, so that no one group contributes significantly more than others, and that peers that volunteer to share files are not necessarily those who have desirable ones. They argue that free riding leads to degradation of the system performance and adds vulnerability to the system.

Ramaswamy et al. [55] suggested using a utility function based on contribution and consumption to prevent free riding. The utility function determines whether the P2P network will permit a peer to search and download a file or just reject its request. The function is based on two parameters; the total size of the files downloaded and shared[11] by the peer. The difference of two values determines the utility of the user to the system. If the user requests a file to download with a size less than the utility value, then it is permitted to download. Otherwise, it is refused. There are two ways to increase the utility value, either by sharing new files or by waiting for some time for a bonus utility value. When a peer downloads a file, its utility is decreased by the amount of the size of the downloaded file.

In [70], Vishnumurthy et al. suggested using a single scalar value, called *Karma*, to evaluate a peer's utility to a system like in [55]. Each peer has an account with some Karma. When a peer uploads a file to a requesting peer, it gets some amount of *Karma* from the requesting node. On the other hand, if it downloads, it gives some amount of its *Karma* to the source peer. The account of a peer is replicated by a group of peers, called the *bank-set*, in order to ensure the Karma against loss and tampering. The transfer of Karma between peers is executed through the bank-set of each peer.

Ledlie et al. [47] proposed that node should select their logical identifier so that the fraction of the ID space for which they are responsible more closely matches their fraction of the total system bandwidth. Their experiment shows that low-bandwidth nodes obtain a 35% improvement in productivity when nodes perform random block downloads on a 256-node network.

[11]The utility of a shared file is moderated by the popularity which is measured by the number of times it is downloaded.

6.4 Data Integrity

In some systems, such as BitTorrent, the downloader knows the cryptographic hash of the data. It is easy for the reader to know whether the received data is correct. However, in many distributed systems, it is more complicated for the reader to ensure the authenticity of the data. We will look at a few systems on how they provide data authenticity.

In CFS [20], a file can only be updated by its publisher. CFS authenticates updates by checking that the new file is signed by the same key as the old one. A time-stamp prevents replays of old updates.

In OpenDHT [59], a hash value of a secret is associated with each key-value pair. To remove a key-value pair in the DHT, a client must supply the secret associated with the key-value pair. To change a value, a client simply removes the old value and puts a new one.

6.5 Anonymity & Censorship Resilience

It is not uncommon for file sharing P2P networks to contain copies of copyrighted material. There are several companies such as NetPD, BayTSP and Cyveillance which provide monitoring on the sharing copyrighted materials. On the other hand, there are also systems which provide anonymity or censorship-resistance. Anonymity plays a central role in private communication. Its applications range from file sharing to military communication, and include anonymous email, private web browsing and online voting. We will look at some of these systems.

The onion routing [31] provides anonymous and private Internet connections through onion-routers. When a node X wants to talk a node Y, but does not want Y to know X's identity, X creates a route through a list of onion-routers $R_1, R_2, ..., R_n$. Messages are passed from X through $R_1, R_2, ..., R_n$ to Y. Replies are passed back from Y through $R_n, R_{n-1}, ..., R_1$ to X. For performance reasons, the same route is used throughout the session. All packets are recursively encrypted using intermediate onion-routers' public keys so that none of the onion-routers know both X and Y's identity. In other words, if some of the onion-routers are compromised, the identity of X and Y are not revealed. Note that onion routing relies on PKI, which is a centralized system. If the CA, in this case the *directory server*, is compromised, the whole system is compromised. It has also been shown that if the first or last onion-router is compromised the source or destination is revealed [69].

Tarzan [28, 30], also uses layered encryption and multi-hop routing. The source chooses a set of relays to act as a path and iteratively establishes a tunnel through these relays with symmetric keys between them. Tarzan is different from onion routing in the sense that Tarzan use a distributed peer discovery protocol instead of a centralized directory server.

In onion routing, a static route is selected to be used throughout a communication session. Wright et al. [72] shows that during long-period communication sessions, intermediate onion-routers are likely to fail, thus routes are reconstructed

using a different list of onion-routers. After a large number of resets, the sender has much higher probability of being a path member than other nodes. Assume that the "first" attacker on each path (of the same session) logs its predecessor. After a number of path resets, the identity of the sender can be guessed with increasing probability. Cashmere [76] addresses the problem by replacing each onion-router with a *relay group*. Thus a route consists of a list of *relay group*. A relay group is a set of nodes which share a public/private key pair. Packets are forwarded by any of the nodes in the relay group. It is shown that Cashmere reduces route reconstruction and improves anonymity over onion routing.

Instead of PKI, Katti et al. [42] takes another approach by splitting message into many small pieces. Recall that PKI is used for hiding the route from intermediate routers. Without PKI encryption, the first router R_1 knows whole route including the sender X and the receiver Y. With PKI encryption, the first router R_1 only knows X and R_2. During the path construction in [42], the sender X, instead of encrypting the remaining route after R_2, splits the remaining route after R_2 into pieces and send each piece to R_2 through different routers.

While previous systems provide anonymous communication channels, Freenet [17] provides anonymous storage. In Freenet, both the publisher and retriever's identity are protected from censorship. It consists of a network of peers that host encrypted documents. Like previous anonymous communication channels, files in Freenet are passed through a chain where each peer knows only the adjacent peers. Peers use keys to locate and access the encrypted documents on the network. There are several types of keys. One of them is just a hash of the document itself. Another one is keyword strings describing the document. As all documents in the system are encrypted with some type of key, the host node does not know what documents it is serving at any point. This allows the use of plausible deniability as a defence against legal attacks against the owner of a node.

6.6 Trust and Reputation

Reputation is often used as a proxy for establishing trust. For example, transactions in the real world are based on personal or corporate reputations. The higher the reputation of an entity, the more trustworthy and reliable it is believed to be. Similarly, in distributed systems, the higher the online reputation of a peer, the more trustworthy it might be believed to be. By employing a reputation-based trust mechanism, a decentralized network attempts to motivate honest participation and promote cooperation in the system.

XenoTrust [26] is a centralized reputation based system. In XenoTrust, clients can submit the assessment of the performance of other clients to the server. For example, "Client X says client Y's honesty is 0.6." Clients can, later on, query the server for the performance of other clients. For example, "Tell me (X) what is the average honesty of Y." A more complicated example, "Tell me (X) what is the average honesty of Y, computed based on statements made by clients whose honesty I (X) valued to be larger than 0.5". In this case, only the opinions of the clients whom X trusted are averaged. In all the examples, X is known as the

advertiser and Y is known as the *subject*.

BambooTrust [43] implements XenoTrust as a distributed system. In BambooTrust, the server is replaced by the OpenDHT [59]. All the assessments with the same *subject* are stored under one OpenDHT node. All the queries requesting the *subject*'s honesty are routed to the node.

PeerTrust [73] is a reputation-based trust supporting framework, which includes a coherent adaptive trust model for quantifying and comparing the trustworthiness of peers based on a transaction-based feedback system. It is decentralized and implemented over a structured P2P network.

In PeerTrust, a peer's trustworthiness is defined by an evaluation of the peer it receives in providing service to other peers in the past. The reputation reflects the degree of trust that other peers in the community have on the given peer based on their past experience. There are five important factors in this evaluation:

1. the feedback a peer obtains from other peers,
2. the feedback scope, such as the total number of transactions that a peer has with other peers,
3. the credibility factor for the feedback source,
4. the transaction context factor for discriminating mission-critical transactions from less or noncritical ones, and
5. the community context factor for addressing community-related characteristics and vulnerabilities.

EigenTrust [40] is a reputation management system proposed for P2P file sharing networks such as Gnutella to identify inauthentic files (or poisoned content) in the network. In a file sharing network, attacker can publish inauthentic files under attractive descriptions. These files waste bandwidth and storage. EigenTrust provides an algorithm to decrease the number of downloads of inauthentic files in a peer-to-peer file-sharing network that assigns each peer a unique global trust value, based on the peer's history of uploads. A distributed and secure method is used compute global trust values, based on Power iteration.

PRIDE [24] is a peer-to-peer reputation infrastructure that uses an elicitation-storage protocol for exchange of recommendations. Each peer runs its own certificate authority which signs the identity certificate of the peer. In order to prevent Sybil attack where an attacker creates a "liar farm", IP Based Safeguard (IBS) is used to ensure that a single IP address cannot act as two nodes.

Similar to EigenTrust, XRep [21] is proposed for preventing inauthentic files from spreading in file sharing networks such as Gnutella. In XRep, reputation sharing is realized through a distributed polling algorithm by which resource requestors can assess the reliability of a resource owned by a participant before initiating the download.

In P-Grid [7], reputations are expressed in the form of complaints. The more the complaints a peer gets, the less trustworthy it could be. This assumes that most of the peers in the network are honest.

OpenPrivacy Distributed Reputation System [45] is based on a web-of-trust style network of peer certifications. Every certificate stores the value of the target's reputation and the confidence of the certificate creator. The reputation network is composed of identities (nodes) and evaluation certificates (edges). The trustworthiness of the nodes can be estimated from a visible sub-graph of the reputation network.

6.7 DoS

Traditional denial of service (DoS) attacks can be targeted to distributed systems. An adversary can generate a large amount of traffic to overload targeted nodes. This will cause the node to appear to fail and the system will be able to adapt to this as if the node had failed in some normal manner. A system must use some degree of data replication to handle even the normal case of node failure. This attack may be effective if the replication is weak (i.e. the malicious nodes can target all replicas easily) or if the malicious node is one of the replicas or colluding with some of the replicas. Sit and Morris [65] suggested that, in order to prevent attacks on replicas, node identifiers must be uniformly assigned and replicas should be located in physically disparate locations. These would prevent a localized attack from preventing access to an entire portion of the key space. If an adversary did wish to shut out an entire portion of the key space, it would have to flood packets all over the Internet.

Another type of denial of service attack called *rapid joins and leaves* is discussed in [65]. As nodes join and leave the system, the rules for associating keys to nodes imply that new nodes must obtain data (from replicas) that was stored by nodes that have left the system. This re-balancing is required in order for the lookup procedures to work correctly. A malicious node could trick the system into re-balancing unnecessarily causing excess data transfers and control traffic. This will reduce the efficiency and performance of the system; it may even be possible to overload network segments. This attack would work best if the attacker could avoid being involved in data movement since that would consume the bulk of the bandwidth. Therefore, the system should force the newly joined node (attacker) to be involved in data movement thus consuming the bandwidth of the attacker.

References

[1] The circle homepage. `http://thecircle.org.au/`.

[2] Codons homepage: Cooperative domain name system. `http://www.cs.cornell.edu/people/egs/beehive/codons.php`.

[3] Faroo homepage: Peer-to-peer web search engine. `http://www.faroo.com/`.

[4] Feedtree homepage: collaborative rss and atom delivery. `http://feedtree.net/`.

[5] Retroshare homepage. `http://retroshare.sourceforge.net/`.

[6] Yacy homepage: P2p web search engine. `http://www.yacy.net/yacy/`.

[7] Karl Aberer, Philippe Cudre-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: A self-organizing structured p2p system. *ACM SIGMOD Record*, 2003.

[8] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 2002.

[9] Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi. Dks: A family of low communication, scalable and fault-tolerant infrastructures for p2p applications. *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, 2002.

[10] Mattias Amnefelt and Johanna Svenningsson. Keso - a scalable, reliable and secure read/write peer-to-peer file system. *Master Thesis, KTH/Royal Institute of Technology*, 2004.

[11] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged byzantine impostors. *Technique Report, YALEU/DCS/TR-1332, Yale University Department of Computer Science*, 2004.

[12] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS), Volume 17 , Issue 2 (May 1999) Pages: 41 - 88*, 1999.

[13] M Castro, P Druschel, A Ganesh, A Rowstron, and DS Wallach. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*, 2002.

[14] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth content distribution in cooperative environments. *International Workshop on Peer-to-Peer Systems*, 2003.

[15] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.

[16] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.

[17] Ian Clarke, Theodore W. Hong, Scott G. Miller, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 2002.

[18] Tyson Condie, Varun Kacholia, Sriram Sankararaman, Joseph M. Hellerstein, and Petros Maniatis. Induced churn as shelter from routing-table poisoning. *Network and Distributed System Security*, 2006.

[19] Tyson Condie, Varun Kacholia, Sriram Sankararaman, Petros Maniatis, and Joseph M. Hellerstein. Maelstrom: Churn as shelter. *13th Annual Network and Distributed System*, 2005.

[20] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. *18th ACM Symposium on Operating Systems Principles*, 2001.

[21] Ernesto Damiani, Sabrina De Capitani di Vimercati, and Stefano Paraboschi. A reputation-based approach for choosing reliable resources in peer-to-peer networks. *9th ACM conference on Computer and communications security*, 2002.

[22] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, , and Ross Anderson. Sybil-resistant dht routing. *European Symposium On Research In Computer Security*, 2005.

[23] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987.

[24] Prashant Dewan and Partha Dasgupta. Pride: Peertopeer reputation infrastructure for decentralized environments. *13th international World Wide Web conference on Alternate track papers and posters*, 2004.

[25] John R. Douceur. The sybil attack. *1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[26] Boris Dragovic, Evangelos Kotsovinos, Steven Hand, and Peter R. Pietzuch. Xenotrust: Event-based distributed trust management. *14th International Workshop on Database and Expert Systems Applications*, 2003.

[27] Amos Fiat, Jared Saia, and Maxwell Young. Making chord robust to byzantine attacks. *European Symposium on Algorithms (ESA)*, 2005.

[28] Michael J. Freedman. Tarzan: A peer-to-peer anonymizing network layer. *First International Workshop on Peer-to-Peer Systems*, 2002.

[29] Michael J. Freedman, Eric Freudenthal, and David Mazieres. Democratizing content publication with coral. *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[30] Michael J. Freedman, Emil Sit, Josh Cates, and Robert Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. *First International Workshop on Peer-to-Peer Systems (IPTPS 02)*, 2004.

[31] David Goldschlag, Michael Reedy, and Paul Syversony. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 1999.

[32] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead (2003). *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[33] Steven Hand and Timothy Roscoe. Mnemosyne: Peer-to-peer steganographic storage. *First International Workshop on Peer-to-Peer Systems (IPTPS 02)*, 2002.

[34] Cyrus Harvesf and Douglas M. Blough. The effect of replica placement on routing robustness in distributed hash tables. *6th IEEE International Conference on Peer-to-Peer Computing*, 2006.

[35] Nicholas J.A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. *USENIX Symposium on Internet Technologies and Systems*, 2003.

[36] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with pier. *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, 2003.

[37] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A decentralized peertopeer web cache. *21th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, 2002.

[38] Elliot Jaffe, Dahlia Malkhi, and Elan Pavlov. Limiting duplicate identities in distributed systems. *2nd Workshop on Future Directions in Distributed Computing*, 2004.

[39] Petri Jokela, Pekka Nikander, Jan Melen, Jukka Ylitalo, and Jorma Wall. Host identity protocol: Achieving ipv4 - ipv6 handovers without tunneling. *Wireless World Research Forum (WWRF8bis)*, 2004.

[40] Sepandar D. Kamvar, Mario T. Schlosser, and Hector GarciaMolina. The eigentrust algorithm for reputation management in p2p networks. *12th international conference on World Wide Web*, 2003.

[41] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *ACM Symposium on Theory of Computing*, 1997.

[42] Sachin Katti, Dina Katabi, and Katarzyna Puchala. Slicing the onion: Anonymous routing without pki. *HotNets IV*, 2005.

[43] Evangelos Kotsovinos and Aled Williams. Bambootrust: Practical scalable trust management for global public computing. *ACM symposium on Applied computing*, 2006.

[44] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*, 2000.

[45] Fen Labalme and Kevin Burton. Enhancing the internet with reputations, an openprivacy white paper. `http://www.openprivacy.org/papers/200103-white.html`.

[46] Martin Landers, Han Zhang, and Kian-Lee Tan. Peerstore: Better performance by relaxing in peer-to-peer backup. *Fourth International Conference on Peer-to-Peer Computing*, 2004.

[47] Jonathan Ledlie, Jeff Shneidman, Matt Amis, and Margo Seltzer. Reliability- and capacity-based selection in distributed hash tables. *Technical report, Harvard University Computer Science*, 2004.

[48] BN Levine, C Shields, and NB Margolin. A survey of solutions to the sybil attack. *Tech report 2006-052, University of Massachusetts Amherst*, 2006.

[49] Alan Mislove, Ansley Post, Charles Reis, Paul Willmann, Peter Druschel, Dan S. Wallach, Xavier Bonnaire, Pierre Sens, Jean-Michel Busca, and Luciana Arantes-Bezerra. Post: A secure, resilient, cooperative messaging system. *9th IEEE Workshop on Hot Topics in Operating Systems (HotOS-IX)*, 2003.

[50] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. *USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[51] Tsuen-Wan Johnny Ngan, Dan S. Wallach, and Peter Druschel. Enforcing fair sharing of peer-to-peer resources. *2nd International Workshop of Peer-To-Peer Systems (IPTPS03)*, 2003.

[52] Hoaison NGUYEN, Toshio OKA, Hiroyuki MORIKAWA, and Tomonori AOYAMA. Sens: A scalable and expressive naming system using can routing algorithm. *Advanced Information Networking and Applications (AINA)*, 2006.

[53] Pekka Nikander, Jari Arkko, and Borje Ohlman. Host identity indirection infrastructure (hi3). *Second Swedish National Computer Networking Workshop (SNCNW)*, 2004.

[54] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Workshop on Algorithms and Data Structures, pages 437449, 1989*, 1990.

[55] Lakshmish Ramaswamy and Ling Liu. Free riding: A new challenge to peer-to-peer file sharing systems. *Multimedia Computing and Networking (MMCN)*, 2002.

[56] Weixiong Rao, Hui Song, and Fanyuan Ma. Querying xml data over dht system using xpeer. *Third International Conference on Grid and Cooperative Computing*, 2004.

[57] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. *Special Interest Group on Data Communication (SIGCOMM)*, 2001.

[58] Thomas Reidemeister, Klemens Bohm, Erik Buchmann, and Paul A.S. Ward. Man-in-the-middle attacks in distributed hash-tables. *IEEE Journal on Selected Areas in Communication*, 2006.

[59] S Rhea, B Godfrey, B Karp, J Kubiatowicz, S Ratnasamy, S Shenker, I Stoica, and H Yu. Opendht: A public dht service and its uses. *conference on Applications, technologies, architectures, and protocols for computer communications*, 2005.

[60] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. *Technical Report MIT/LCS/TR-684, MIT LCS*, 1996.

[61] Hosam Rowaihy, William Enck, Patrick McDaniel, , and Thomas La Porta. Limiting sybil attacks in structured peer-to-peer networks. *IEEE Infocom Mini-Symposium*, 2005.

[62] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. *18th ACM Symposium on Operating Systems Principles*, 2001.

[63] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. Defending against eclipse attacks on overlay networks. *the 11th workshop on ACM SIGOPS European workshop*, 2004.

[64] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using sip. *Columbia University Technical Report CUCS-044-04*, 2004.

[65] E Sit and R Morris. Security considerations for peer-to-peer distributed hash tables. *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[66] I Stoica, R Morris, D Karger, MF Kaashoek, and H Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Special Interest Group on Data Communications (SIGCOMM)*, 2001.

[67] I Stoica, R Morris, D Liben-Nowell, DR Karger, MF Kaashoek, F Dabek, and H Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 2003.

[68] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *Special Interest Group on Data Communications (SIGCOMM)*, 2002.

[69] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. *Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[70] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. Karma : A secure economic framework for peer-to-peer resource sharing. *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[71] Michael Walfisha, Hari Balakrishnana, and Scott Shenkerb. Untangling the web from dns. *Networked System Design and Implementation (NSDI)*, 2004.

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

[73] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust in peer-to-peer communities. *IEEE Transactions on Knowledge and Data Engineering (TKDE), Special Issue on Peer-to-Peer Based Data Management*, 2004.

[74] H Yu, M Kaminsky, PB Gibbons, and A Flaxman. Sybilguard: Defending against sybil attacks via social networks. *conference on Applications, technologies, architectures, and protocols for computer communications*, 2006.

[75] Lidong Zhou and Robbert van Renesse. P6p: A peer-to-peer approach to internet infrastructure. *3rd Interational Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[76] Li Zhuang, Feng Zhou, Ben Y. Zhao, and Antony Rowstron. Cashmere: Resilient anonymous routing. *USENIX Symposium on Networked Systems Design and Implementation*, 2005.

[77] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An architecture for scalable and fault tolerant widearea data dissemination. *Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2001.