Peer-to-Peer Systems and the Grid

Jon Crowcroft, Tim Moreton, Ian Pratt, Andrew Twigg
University of Cambridge Computer Laboratory,
JJ Thomson Avenue, Cambridge, UK
firstname.lastname@cl.cam.ac.uk

In this chapter, we survey recent work on peer-to-peer systems, and venture some opinions about its relevance for the Grid. We try to bring some historical perspective and structure to the area, and highlight techniques and open research issues in the peer-to-peer arena which may be relevant to Grid computing.

1 Introduction

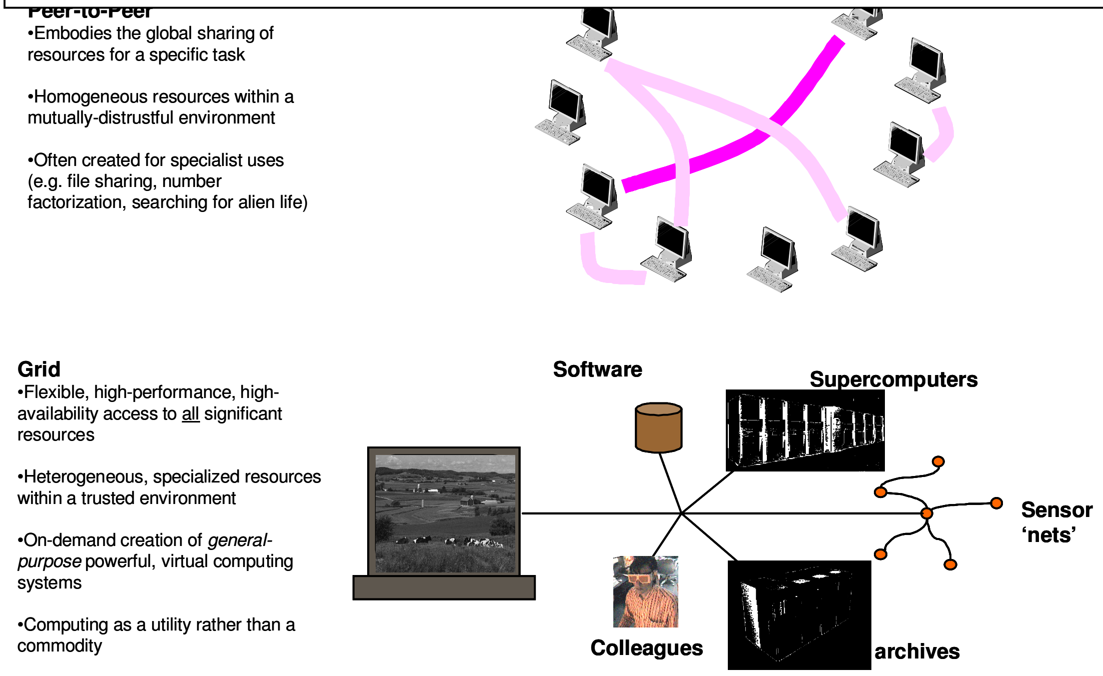
Peer-to-peer systems are distributed Internet applications in which the resources of a large number of autonomous participants are harnessed in order to carry out the system's function. In many cases, peers form self-organising networks that are layered over the top of conventional Internet protocols and have no centralized structure.

The assumptions on which peer-to-peer computing has grown are wildly different than those underlying Grid computing. Peer-to-peer systems have evolved to support resource sharing in an environment characterised by users potentially numbering millions, most with homogenous desktop systems and low bandwidth, intermittent connections to the Internet. As such, the emphasis has been on global fault-tolerance and massive scalability. On the other hand, Grid systems have arisen from collaboration between generally smaller, better-connected groups of users with a more diverse set of resources to share.

Though these differences have led to distinct sets of requirements and applications, the long-term objectives of peer-to-peer and Grid systems may not be disjoint. In the short term, research emerging from this field will be applicable to many of the challenges faced by the next-generation Open Grid Services Architecture (OGSA) [50], in areas as diverse as resource discovery, scalable load balancing of computation, and highly available storage and data distribution systems.

This chapter is arranged as follows. After investigating the history of peer-to-peer computing and putting the topic in context, we look at the notion of peer-to-peer middleware along with several broad application areas: storage, computation, and searching. We proceed by investigating the relationship between peer-to-peer and Grid computing, and conclude by looking at possible future developments which build on peer-to-peer and Grid computing. Our view is that the future of peer-to-peer and Grid is exciting, but choices need to be made carefully to avoid the pitfalls of the past.

Figure 1: Comparing peer-to-peer and grid computing at a high level


Peer-to-Peer
- Embodies the global sharing of resources for a specific task
- Homogeneous resources within a mutually-distrustful environment
- Often created for specialist uses (e.g. file sharing, number factorization, searching for alien life)

Grid
- Flexible, high-performance, high-availability access to all significant resources
- Heterogeneous, specialized resources within a trusted environment
- On-demand creation of *general-purpose* powerful, virtual computing systems
- Computing as a utility rather than a commodity

2 A Brief History

The topic of peer-to-peer networking has divided research circles in two: on the one hand there is the traditional distributed computing community, who tend to view the plethora of young technologies as *upstarts with little regard for, or memory of the past* – we will see that there is evidence to support this view in some cases. On the other hand, there is an emergent community of people who regard the interest as an opportunity to revisit the results from the past, with the chance of gaining widespread practical experience with very large scale distributed algorithms. One of the oldest uses of the term "peer-to-peer computing" is in IBM's Systems Network Architecture documents on LU6.2 Transactions, over 25 years ago. The term, which we shall use interchangeably with p2p, came to the fore very publicly with the rise and fall of Napster [29]. Although there are prior systems in this evolutionary phase of distributed computing (e.g. Eternity [4]), we choose to limit the scope of this survey to the period from "Napster 'til Now"[1].

2.1 Beyond the Client-Server Model

The consensus seems to be that a peer-to-peer system can be contrasted with the traditional twenty-five or more year old *client-server* systems[2]. Client-server systems are asymmetric; we usually assume that the server is a much more powerful, better connected machine. The server

[1] i.e. 1998 – 2003.

[2] It can be argued that transient Grid services and their orchestration blur this divide, and that the fragmentation and cross-integration of enterprises' infrastructures is one motivation for OGSA. The symmetry of the peer-to-peer model, however, contrasts more strongly.

is distinguished as running over some longer period of time and looking after storage and computational resources for some number of clients. As such, the server emerges as a single bottleneck for performance and reliability. Server sites may make use of a number of techniques to mitigate these problems, such as replication, load balancing and request routing, so that one conceptual server is made up of many distinct machines. At some point along the evolution of this thinking, it is a natural step to include the clients' resources in the system - the mutual benefits in a large system are clear. The performance gap between desktop and server machines is narrowing, and the spread of broadband is dramatically improving end clients' connectivity.

Thus peer-to-peer systems emerge out of client-server systems by removing the asymmetry in rôles: a client is also a server, and allows access to its resources by other systems. Clients, now really *peers*, contribute their own resources in exchange for their own use of the service. Work (be it message passing, computation, storage, or searching) is partitioned in some (usually distributed) means between all peers, so that each peer consumes its own resources on behalf of others (acting as a server), but so that it may ask other peers to do the same for it (acting as a client). Just as in the real world, a fully cooperative model such as this may break down if peers are not provided with incentives to participate. We look into trust, reputation, and work into economically grounded approaches in Section 4.3.

A claim sometimes made about peer-to-peer systems is that they no longer have *any* distinguished node, and thus are highly fault tolerant and have very good performance and scaling properties. We will see that this claim has some truth to it, although there are plenty of peer-to-peer systems that have some level of distinguished nodes, and also plenty that have performance limitations. In fact, the fault tolerance claims are hardly born out at all in the early instantiations of the peer-to-peer movement. Initial availability figures in Napster, Gnutella [37] and Freenet [12] do not compare favourably with even the most humble of web sites!

However, second and later generation systems may indeed provide the claimed functionality and performance gains, and we will see in Pastry [39], Chord [42] and CAN [35] very promising results, and even more recent work building applications and services over these systems shows great potential gains.

One can look at differences and similarities between classical client-server and modern peer-to-peer systems on another axis: statefulness. Despite successes with stateless servers, many Web servers offer wider functionality by using cookies, script-driven repositories and Web Services to keep state over various transactions with a client. In a peer-to-peer system, since a peer rarely knows directly which node is able to fulfil its request, each keeps track of a soft state set of neighbours (in some sense) in order to pass requests, messages or state around the network. While such use of soft state is a long-recognised technique of Grid computing infrastructure (e.g. [50]), Grid services themselves are inherently stateful during their (explicitly-managed) lifetimes.

Yet another viewpoint from which one can dissect these systems is that of the use of *intermediaries*. In the Web (and client-server file systems such as NFS and AFS) we use caches to improve average latency and to reduce networking load, but typically these are arranged statically. We will see in peer-to-peer systems how a dynamic partitioning of work between cooperative peers allows excellent locality-oriented load balancing. Taking the example of caching, content distribution systems such as PAST [40] and Pasta [28] aim to distribute data in proportion to demand for it, on peers close in the network to that demand. Systems for Grid services in time may extend the idea to seamlessly and dynamically provisioning distributed computation close to a data source, in proportion to the time and parallelization demands of the job.

The classical distributed systems community would claim that many of these ideas were present in the early work on fault tolerant systems in the 1970s. For example the Xerox Network System's name service, *Grapevine* [27] included many of the same traits as the systems mentioned here. Other systems that could easily be construed as architecturally true peer-to-peer systems include Net News (NNTP is certainly not client-server) and the Web's Inter-cache protocol, ICP. The Domain Name System also includes Zone Transfers and other mechanisms which are not part of its normal client-server resolver behaviour. IP itself is built out of a set of autonomous routers running a peer-to-peer protocol. For example, the routing protocols OSPF and BGP are peer-to-peer, and certainly not client-server, and are designed that way for exactly the reasons given above for peer-to-peer.

2.2 Deploying Internet Services by Overlaying

At the same time as peer-to-peer, a number of network researchers have been frustrated in their attempts to deliver new network services within the context of traditional telecommunications or Internet networks[3]. Instead, researchers have built experimental infrastructures by constructing *overlay* systems, developing new infrastructures using services of and layering on the existing infrastructure, rather than by complementing or replacing it.

An overlay may be a simple as a collection of static IP in IP tunnels, or as complex as a full dynamic VPN ("virtual private network"). Some of these systems are in use in the active networks research community. Others are more straightforward in their motivation, such as the Grid communities' requirements for more robust Internet connectivity!

Clearly, overlaying is a relative term - what the overlay and the underlay are depend on the infrastructure being developed. An essential aspect to most peer-to-peer applications is a distributed object location service used to communicate or pass messages between peers; requiring peers to perform routing in this way is effectively an infrastructure function. Rather than supplant existing Internet services, most of the peer-to-peer work that we discuss in this document is positioned at the Application Layer in the ISO model and uses overlaying to achieve its infrastructure goals: each peer runs its software over the existing operating system and uses the existing TCP/IP architecture.

The architecture of Grid systems uses overlaying to provide unified (virtualised) interfaces to all aspects of service management, and aims to integrate underlying native platforms and protocols. In contrast, peer-to-peer systems leverage overlaying to provide an abstraction for *addressing* between a set of peers spread throughout the Internet.

IP was originally an overlay service, implemented above other layered communications system: the PSTN, ARPANET and X.25 circuit switched networks. Indeed, this overlay

[3] New Internet network level services such as IP QoS in the form of integrated services, and differentiated services, as well as novel service models such as multicast and mobility have proved notoriously hard to build and deploy in their native forms.

model keeps re-emerging as network operators deploy faster switched infrastructures such as Frame Relay, ATM and WDM and PONS (Pure Optical Networked Systems) core networks.

Balakrishnan *et. al.* at MIT have developed the Resilient Overlay Network system [2,3], where a number of sites collaborate to find a longer path at the IP ``level'', which has better properties (such as throughput or loss) at the application level, by dynamically routing via a set of dynamically created tunnels. In parallel, Turner *et. al.* at Washington developed a similar approach for multicast [41].

The difficulties in deployment of native IP multicast have led several groups to develop schemes that use overlaying, including 'End System only Multicast' [11,10]. Further work used this to implement multimedia conferencing systems and demonstrated the workability and usability of the scheme [23]. Other researchers have used the approach for streaming media [16], *anycast* [46], and server selection [21].

2.3 Napster

Peer-to-peer "year zero" can effectively be set to Napster [29]. Napster was used heavily to share music content. This is a political hot-potato - the music industry was (and still is at the time of writing) very slow to make content available through the Internet, perhaps due to perception of the lack of market, perhaps due to lack of efficient rights management, and content tracking and protection mechanisms.

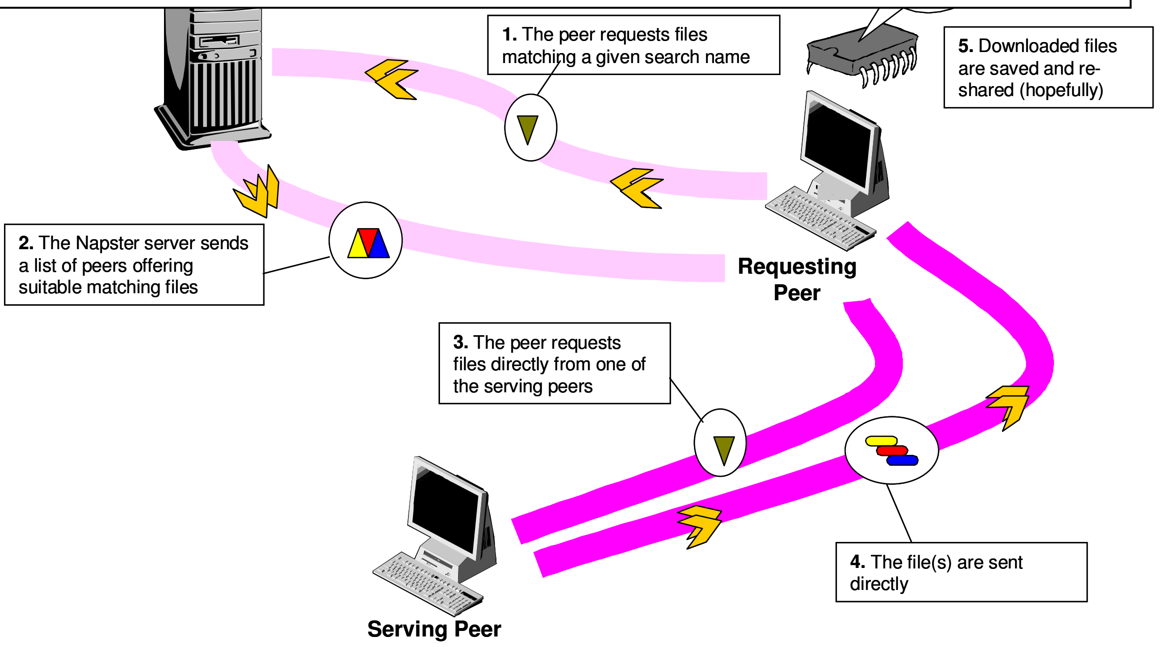
Napster is a *file sharing system*, in that it allows users to search for and download music files held on other Napster users' hard drives. When the application is first started, metadata for all of a user's shared songs is transferred into a global directory. When other users search for a song using keywords in this metadata, the directory returns a list of clients sharing songs matching the query. The end machines (peers, in this sense) cooperate to transfer the song directly between themselves. Each takes on the role of a client and a server, alternating as to whether they are uploading a song to another user or obtaining a new one for themselves.

Opinions differ as to whether Napster itself can be described as truly peer-to-peer, since its directory is stored on central servers, and clients never participate in processing search queries. However, by distributing the bandwidth and storage requirements of maintaining the music files themselves, the system succeeded in ameliorating its initial perceived scalability and performance bottlenecks. Further, the real utility of the network - the diversity of music that was available - was certainly a property of its constituent peers.

Technically, the program suffered a simple interface, and the poor reliability and bandwidth of other clients' connections often hampered users' attempts to retrieve songs. However, since it dramatically simplified the task of obtaining music on the Internet (albeit mostly illegally), it became immensely popular, drawing the close attention of music industry executive - at its peak, Napster could boast around 1.6 million simultaneous users[4].

[4] Source: Webnoize, February 2001.

Figure 2: Napster - an example of a centralized peer-to-peer system


1. The peer requests files matching a given search name
5. Downloaded files are saved and re-shared (hopefully)
2. The Napster server sends a list of peers offering suitable matching files
Requesting Peer
3. The peer requests files directly from one of the serving peers
4. The file(s) are sent directly
Serving Peer

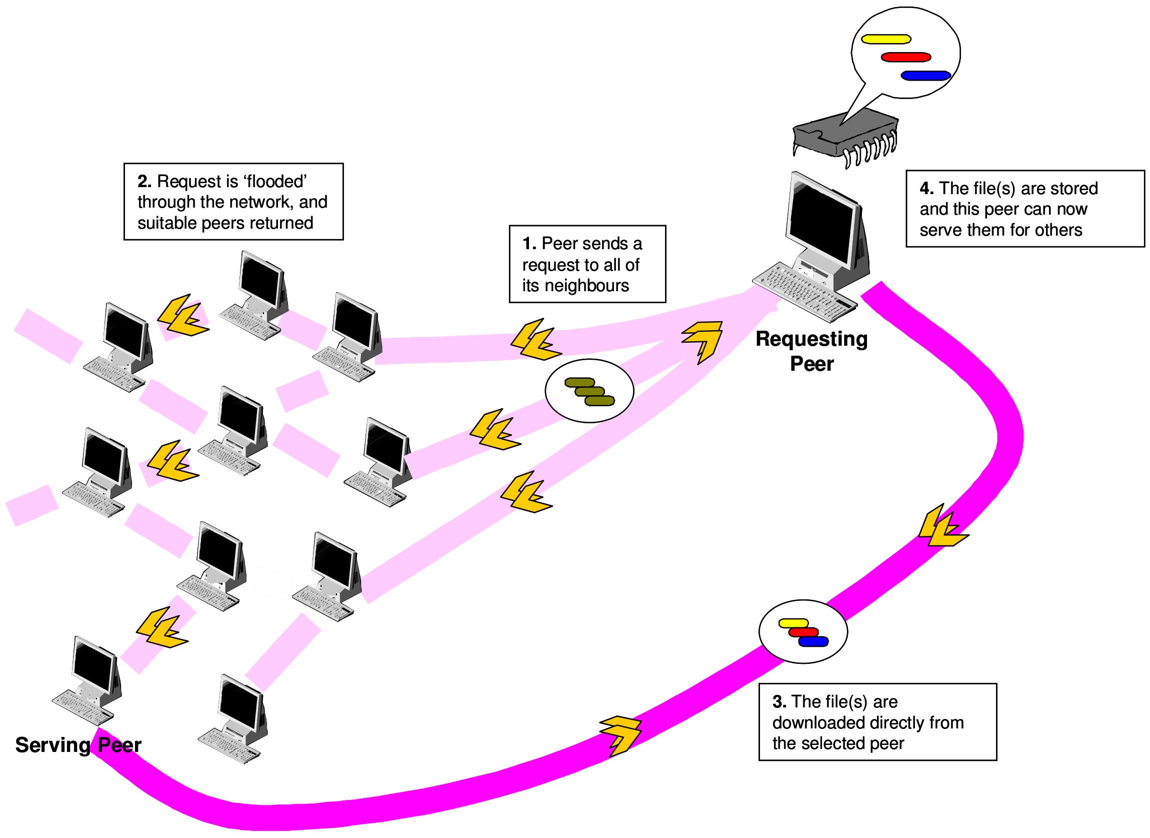
The centralised directory was a single point of failure for legal, economic and political attacks, and allowed Napster to be shut down by court order for facilitating its users to infringe copyright. While the service was active, the directory servers became a severe bottleneck, costing the company increasing amounts in machines and network connectivity. An effective fully distributed searching solution would have solved these issues; though not only did Napster underestimate how popular their product would become, but also, in technical terms, such a facility is the matter of ongoing research.

2.4 In Napster's Wake

Although Napster's success was attributable to online music sharing being a ``killer application'' at the time, nevertheless it demonstrated the potential in harnessing the resources of clients to satisfy their own need for a service. Further, the demise of Napster meant there became a requirement within the music sharing community for a fully decentralized service that would not be susceptible to a similar legal attack. Those projects that rose to the challenge contributed to the start of a process of important technical developments due to academic and industrial research in distributed object location and routing, distributed searching, and content dissemination.

2.4.1 The second generation: full decentralization

The first of these was Gnutella [37], a distributed search protocol adopted by several file sharing applications that dispensed with the centralised directory and instead broadcast search queries between a peer's neighbours. Despite several measures to limit and restrict queries, several studies and user experience found that both the queries and the volume of


2. Request is 'flooded' through the network, and suitable peers returned
1. Peer sends a request to all of its neighbours
4. The file(s) are stored and this peer can now serve them for others
Requesting Peer
3. The file(s) are downloaded directly from the selected peer
Serving Peer

'ping' messages used to maintain neighbour lists caused excessive network load, limiting the size of Gnutella networks, the chance of satisfying a given query and the amount of a client's bandwidth left for actual file transfers.

Next, systems for locating content including Freenet [12] added mechanisms to *route* requests to an appropriate node where the content is *likely* to be, in a best-effort partial partitioning of the networks' sum content.

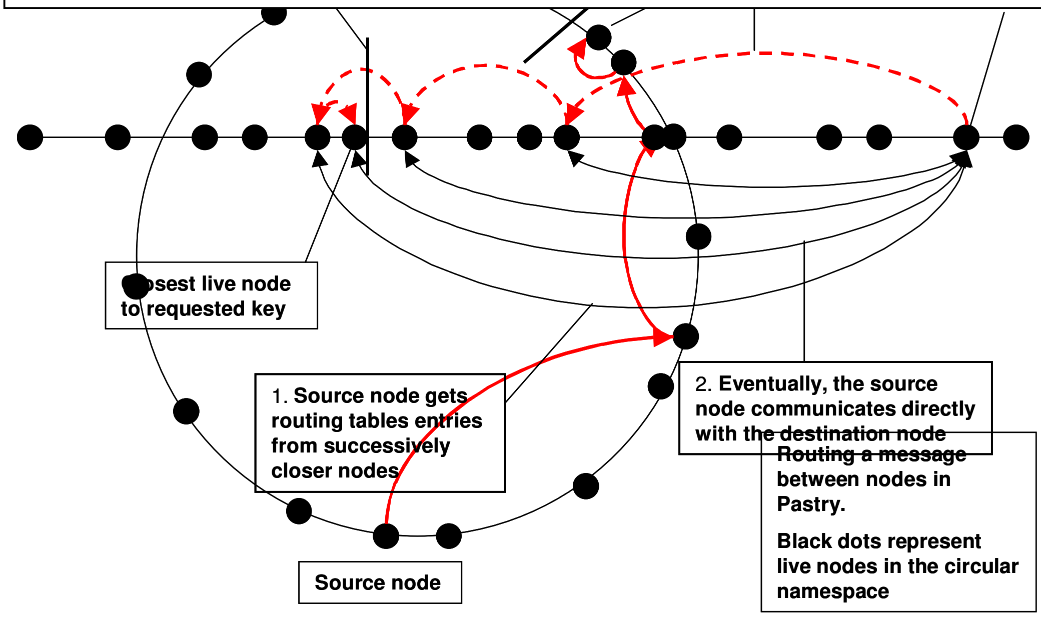
Figure 3: Gnutella - an example of a fully decentralized peer-to-peer system

ded
ome
super-peers, caching and serving common queries or content; child peers perform queries and provide content through their allocated super-peer. These schemes take advantage of the observed Zipf-like distribution of requests for content, and mitigate the difficulties of passing queries through hosts on high latency, low bandwidth dialup connections.

2.4.2 The third generation: efficient routing substrates

Although the range of problems to which peer-to-peer techniques had been applied was still limited towards the end of 2001, a common requirement had emerged. In order for each peer to make a useful contribution to the global service, a reliable way of partitioning the workload and addressing the node responsible was needed. Further, the emphasis on scalability, and the corresponding observation that in a global-scale system peers will be joining, failing and leaving continually, requires this to be done with knowledge of only a fraction of the global state on each peer, maintained with only a low communication overhead in the underlying network.

Figure 5: Routing a message between nodes in Kademlia, a distributed hash table (DHT). The key space is acyclic and the source node locates the node closest to the requested key by successively learning about and querying nodes closer to it. The dashed line represents the route that Pastry would have taken.



osest live node to requested key
1. Source node gets routing tables entries from successively closer nodes
2. Eventually, the source node communicates directly with the destination node
Routing a message between nodes in Pastry.
Black dots represent live nodes in the circular namespace
Source node

Figure 4: Routing a message between nodes in Pastry, a distributed hash table.

tes,
hain
approach centres around the Distributed Hash Table (DHT), where nodes are assigned a unique pseudo-random identifier which determines their position in a key space. Messages are routed to points within the same key space, and are delivered eventually to the *closest* node. According to the way in which applications use this service, a message destined for a given key represents a request to provide a given service with respect to that key. As requests' keys must be mapped on to the key space pseudo-randomly too, usually using some secure hash function such as SHA-1 [53], then these DHTs offer effective partitioning of the work between peers. The schemes differ as to the structure of information on nodes, and how messages (or sometimes requests for routing information) are passed between peers. Other variants are being developed, and are discussed below.

The presence of DHT substrates offering routing services, node management and a simple interface led to a rise in the number and variety of peer-to-peer applications being developed. Systems for event dissemination and overlay multicast, file archival, file systems, and replacements for DNS have emerged; we survey these areas in Section 3.

2.5 Future Directions for Peer-to-Peer

Peer-to-peer is still a `hot' topic and progress is steady. We outline below technical issues facing the research community, before describing them and their application to Grid architectures in more detail in Section 5.

While DHTs introduced an essential split between peer-to-peer middleware and applications, they have limitations that are providing an impetus for more flexible schemes. Further, each proposal for a new routing substrate contains convincing evaluation results from large-scale simulations, but no Internet deployment has tested their properties under real-world conditions - with respect to failure and latency, in particular. Such analysis will play an important part in directing research.

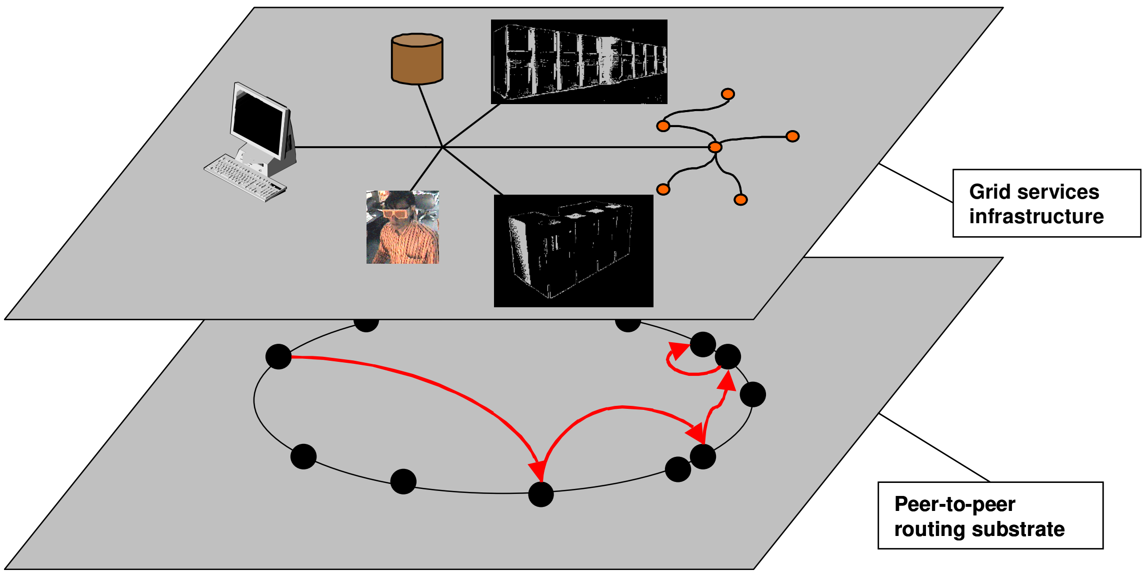
The scale of peer-to-peer systems means that participants are typically individuals and organizations without an out-of-band trust relationship. This key characteristic is not currently shared by Grid computing, but takes on increasing significance as Grid architectures scale up. This property generates interesting work in the area of trust, reputation systems, economic incentives, and detecting malicious participants.

In on
de on
a in
re ct
be al
benefits of cooperation will not stop people defecting. Work on providing incentives to participants through economic [54] and trust [51] models forms an important part of ongoing research.

Figure 6: A peer-to-peer grid computer? One seeks to combine the varied resources, services and power of grid computing with the global-scale, resilient and self-organizing properties of large peer-to-peer systems. A peer-to-peer substrate provides lower-level services on which to build a *globally-distributed* grid services infrastructure. Issues such as trust which grid computing assumes but are lacking in peer-to-peer systems need to managed *between* the layers.

Much progress has been made in the security and censor-resistant aspects of some applications [55,8], including an important general result in the impossibility of preventing *pseudospoofing* [56] (owning more than one virtual identity) without a trusted link to a real world identity.

Finally, as peer-to-peer computing matures, we will see a diversification in its applications. As Grid systems scale up and peer-to-peer techniques begin to capture shared use of more specialised resources, and as users are able to specify location, performance, availability and consistency requirements more finely, we may see a convergence between techniques in the two areas. We describe this view further in Section 6.



3 Applications

This section roughly divides existing peer-to-peer projects up into routing substrates, and the main classes of applications that run on them: systems for storage, computation, and searching.

3.1 Routing Substrates

As mentioned above, routing substrates are peer-to-peer middleware that facilitate communication between and management of a network's constituent nodes. We categorise them as *unstructured* or *structured* - the essential difference being whether the neighbours that each peer maintains are organised in such a way as to allow deterministic location of a piece of content or a node.

3.1.1 Unstructured routing

When joining a peer-to-peer network, a new node needs knowledge of at least one peer already present in the network from which to obtain its initial `routing table entries'. Nodes in unstructured systems tend to maintain these same neighbours, only replacing their entries if it detects that the node in question has failed. This means the topology of the network grows in an arbitrary, unstructured manner; it becomes difficult to bound the maximum path length and guarantee even connectivity between groups of nodes. This impacts on performance and reliability - unintentionally, some nodes may become bottlenecks.

So far such systems have not allowed efficient searching (either for keys, or more complicated metadata queries). Gnutella [37] used a flooding-based search where a query is broadcast to each of its neighbours, which in turn pass it on to each of their neighbours; each peer tracks the queries that it has seen to prevent routing loops. Unfortunately, the build-up of traffic from each query is exponential – to such an extent that unless the search breadth and depth are very low (so that only queries for very popular content are likely to succeed), the system will not scale.

Other more efficient schemes have been informed by borrowing from conventional data structures, including iterative deepening techniques [57] (incrementally considering the nodes at a given number of hops from the requester), or using random walks [58].

However, a concept of *direction* seems essential in pruning the potential search space and routing efficiently: it is for this reason DHTs organise nodes on numeric keys. In Freenet [12], a publishing network where peers cooperatively participate in caching and retrieving documents, each node maintains a data store that locally caches the data and key associated with a document, and also the node from which it was originally obtained. Entries for evicted documents are maintained, but without the attached data. On receiving a request for a key where no cache entry exists, a node finds the entry for the document with key *numerically closest* to that sought, and forwards the request to the node from which it was obtained. In this way, Freenet nodes over time may come to specialize in portions of the keyspace, and other nodes' knowledge of this gives searches direction. However, because this scheme relies on uniform local knowledge of the keyspace, it suffers poor worst-case lookup performance [59] and cannot guarantee success, even when the matching data exists in the network.

3.1.2 DHTs and structured routing substrates

Structured routing substrates (approximately synonymous with DHTs at present) organise their peers so that any node can be reached in a bounded number of hops, typically logarithmic in the size of the network.

Although subtly different, all of the main schemes operate similarly. Pastry maintains per-node routing tables organised by the length of the entries' shared address prefix. Tapestry [48] nodes each maintain an inverted index organised, again, by prefix. Kademlia [26] routes according to distance between IDs using the XOR metric. In Chord [42], each peer is arranged in a circular ID space and maintains links with its immediate predecessor and successor, and a number of 'chords' to other nodes whose distances are arranged in an exponential fashion. CAN [35] uses several hashes, to map into a multi-dimensional ID space; queries are passed along the axes of this space.

The class of DHTs derive from Karger's work on consistent hashing [60] and Plaxton's distributed data structure [33]. They are mathematically described in [7], which gives upper and lower bounds on the query time in DHTs - the results are similar to the empirical results from CAN and Chord.

As an alternative underlying technique, Distributed Tries [18] use a trie, and so the same `lookup(key)->value` as DHTs, but they may offer lower average-case message overhead. Each node holds a trie: by using a backtracking search they query nodes that are known to contain other parts of the trie, which in turn may return the object or more up-to-date or detailed parts of the trie. In the worst case this scheme degenerates to broadcast search.

Many recent schemes focus on minimising lookup latency. Pastry fills routing table entries with the closest nodes that it can, based on a local proximity metric [39]. The Oceanstore project [24] uses Bloom filters [9] to probabilistically locate content on nearby nodes. Brocade [61] employs landmark routing over Tapestry [48] to reduce cross-AS traffic.

An inherent difficulty with DHTs concerns its uniform partitioning of work. Since data (be it content, blocks to store or multicast topics) is associated with pseudorandom keys, a user cannot control which peer is responsible for it[5]. Locality of reference is broken: an essential property if peer-to-peer computing is to offer the performance seen by conventional client-server models. Significantly, though, SkipNet [52] offers a hybrid architecture based on skip lists that can route and store within explicit administrative domains.

3.2 Content Distribution and Storage Systems

Peer-to-peer techniques first found their niche in *file sharing systems*. We distinguish such applications from *distributed file systems* such as NFS [62]: the former allow users to obtain specific well-defined content, usually based on a metadata search; the latter expose local file system hierarchies to remote users, may be writeable, and may implement access control or consistency semantics. We also describe *distributed archival storage systems*, in which insertion and retrieval operations are coarse-grained (i.e. documents at a time) and storage is durable, long-term, and often focused on censor-resistance or anonymity. Finally, we consider storage requirements in Grid computing applications.

3.2.1 File sharing

Recall that Napster, while partly centralized, applied peer-to-peer techniques to file sharing by distributing the high bandwidth requirement of transfers: files were passed directly between peers. However, in this set up the performance and reliability of file retrieval is dependent on the peer with which a user is transacting, preventing the system implementing any QoS guarantees. Frequently, transfers may be aborted as the sender cancels, disconnects, or by network partition. The rate at which transfers proceed depend on the relative positions of the endpoints, their latency and their bandwidths.

Recent file sharing applications have used various techniques to mitigate these difficulties. Reliable, well-connected nodes dynamically become s*uper-peers*, as described above, through which queries and replies to other nearby users are routed. Caching on super-peers improves the availability and retrieval time of commonly-sought files, even in the face of partitions. By storing partial directory information they also reduce the network load of low-bandwidth endpoints, allowing their transfers to complete more quickly.

Swarm distribution in systems such as Kazaa [63] improves load balancing and reduces a transfer's dependence on individual nodes by naming the file by the secure hash of its contents. If a transfer aborts, another peer sharing the same file may be identified and the transfer resumed. Further, subdividing the file into portions and naming these allows different parts of the file to be transferred from multiple sources at once, improving performance for well-connected machines. By allocating small portions to dial-up peers and larger portions to others, each node contributes according to its ability.

Recent peer-to-peer applications based on DHTs offer content streaming and effective content dissemination by replicating data in proportion to demand for it, close in the network to that demand: these include CFS [15] and Pasta [28], file systems, and SplitStream [82], for streaming media.

[5] Note that this means current DHTs are not suitable, for example, for use as RLIs in Giggle's Replica Location Service[], where it is required that "failure of a remote RLS component does not affect local access to local replicas".

The difficulty of performing arbitrary metadata searches and obtaining deterministic results in a fully decentralized environment has limited file sharing; many existing systems are restricted to specific areas (in particular media and software distribution) where the search-space for users is well-defined; typically, users ‘discover’ content by out-of-band means. Certainly, uptake of a consistent metadata framework such as Dublin-Core [64] is important for progress.

3.2.2 Archival storage systems

The Eternity service [4] proposed a design for a system that offers censor-resistant durable storage to users for a specific time period by replicating and scattering data across servers in a distributed system. While Eternity is a design, and does not specify any of the mechanisms by which peer-to-peer applications are now characterised, its ambitions were reflected in many early peer-to-peer systems.

Free Haven [65], a document storage system, implements a variation on Eternity. Its primary aim is to offer anonymity to publishers of documents, and to provide plausible deniability (see Section 4.8) to server owners. Documents are stored on a peer-to-peer network of mutually-distrustful servers, called a *servnet*: queries and publication requests are broadcast to the whole group.

Much early work on the nature of peer behaviour is present in Free Haven’s design. It makes pairs of servers mutually accountable for files that they store using a buddy system, and uses a reputation system to report complaints from ‘buddies’: servers over time develop trust for other servers according to their reputation. An economic system of trading reputation capital for resources on other servers provides an incentive to participate and minimises the damage caused by individual malicious entities.

PAST [40] is an experimental archival utility built over the Pastry DHT. Storage and retrieval operations are performed at the granularity of whole files. No human-readable naming scheme is supported: rather, a fileID associated with the insertion must be passed by other means. Inserted files are immutable until withdrawn by their owner. The system can offer strong data persistence by enforcing storage quotas through a scheme of smartcards allocated out-of-band.

3.2.3 Global-scale file systems

Network file systems were one of the first great successes of client-server distributed systems – Sun RPC and NFS were ubiquitous from the mid 1980s in research and education labs and many small organisations use Samba to share storage resources.

Recent designs for distributed file systems aim more ambitiously to present a unified view of storage resources of any Internet-connected system, while offering secure reliable storage, better-defined, application-variable concurrency guarantees, and efficient content-distribution. Nowadays, the cost of management and organisation of storage tends to exceed the cost of the physical media – this has led to the adoption of peer-to-peer techniques for managing a large, dynamic set of unreliable hosts, proposed as a replacement to brittle location-dependent mutual client-server systems such as NFS, and high-maintenance organisation-centric client-server systems such as AFS [66].

The Cooperative File System (CFS) [15] is implemented over Chord. Files are split into fixed-size blocks, which are identified by their secure hash, then distributed to nodes. Storage can be guaranteed for a set time period enabled by per-node storage limits based on IP addresses. Users arrange files hierarchically in a 'file system', which forms a per-publisher decentralised namespace. CFS offers coarse-grain file mutability, but since each publisher manages their own file system, collaborative file manipulation is not possible. No cache consistency or concurrent update control scheme is proposed.

Pasta [28] is a prototype peer-to-peer file system operating over Pastry [39]. Its design offers a persistent, mutable shared storage and content distribution service to a potentially very large collection of mutually distrustful users. It integrates closely with local file systems through a loopback NFS interface and presents a decentralized, hierarchical namespace.

Users store data as a series of immutable blocks, referenced through mutable index blocks that store fragments of decentralized namespace. By replicating and maintaining blocks across peers, Pasta offers persistent and reliable storage. Widespread localised caching in proportion to demand provides an efficient content distribution system by preventing hot spots and migrating data towards the source of requests. Files are split into blocks in terms of their contents, to exploit commonality between sections of files to improve storage efficiency and enhance caching performance; in particular this allows Pasta to store different versions of a file efficiently and modify them on a copy-on-write basis. A scheme of storage quotas, enforced in a distributed fashion, regulate consumption.

Ongoing work on Pasta focuses on a scheme whereby privately-owned data may be collaboratively modified by untrusted users. Through namespace overlays, users may appear to share and organize each other's stored data; modifications to files or namespace are seen as in a versioning file system, and third parties form a view by choosing which modifications to trust.

OceanStore [24] is an infrastructure for providing persistent data storage in a global-scale ubiquitous computing environment, although its current prototype, Pond [67] shares more with the above systems. It uses a variant of Plaxton's distributed hierarchical data structure [33] to locate stored data efficiently. The system caches data widely for performance and availability, and performs distributed cache consistency management. For each file, a primary tier of replica-carrying nodes use a Byzantine agreement protocol [68] to commit modifications: a conflict resolution scheme resolves concurrent updates as Bayou does [69].

3.2.4 Data access in computational Grids

Requirements for data access and movement in Grid computing, where computation is performed at a remote site, may see applications built that combine techniques from both content distribution and peer-to-peer file systems.

Several established designs for peer-to-peer file systems including Oceanstore[24] and Pasta are suitable to many aspects of the task, and offer fault-tolerant, highly available long-term storage, too. They provide schemes to support true location-independence of data on a global-scale, allowing data to be gathered from a variety of sites and sensors, or from dynamically-established caches, while being named in a unified way. While many schemes offer limited concurrency semantics, this is all that most Grid file access patterns require. Such systems also use conventional file system interfaces – necessary to minimise rewriting of applications for a Grid setting.

Grid applications require flexible caching of input files and output files for rerunning computational tasks with different parameters or for comparing results. Systems such as CFS and Pasta incorporate caching schemes suitable for this rôle. However, the size of some data sets may necessitate them being stored across different nearby sites, then streamed; further, streaming of diagnostic data back to the client site is essential for tracking progress. Data movement problems such as this may benefit from swarm distribution and other techniques developed for peer-to-peer file sharing applications.

3.3 Distributed Computation

Processor cycle sharing is familiar through remote shell and other applications. Since 'Moore's Law' appears still to describe the growth in new CPU speed over time, we have several hundred million personal computing and workstation machines now on the public Internet, which have hundreds of MIPS going unused every second.

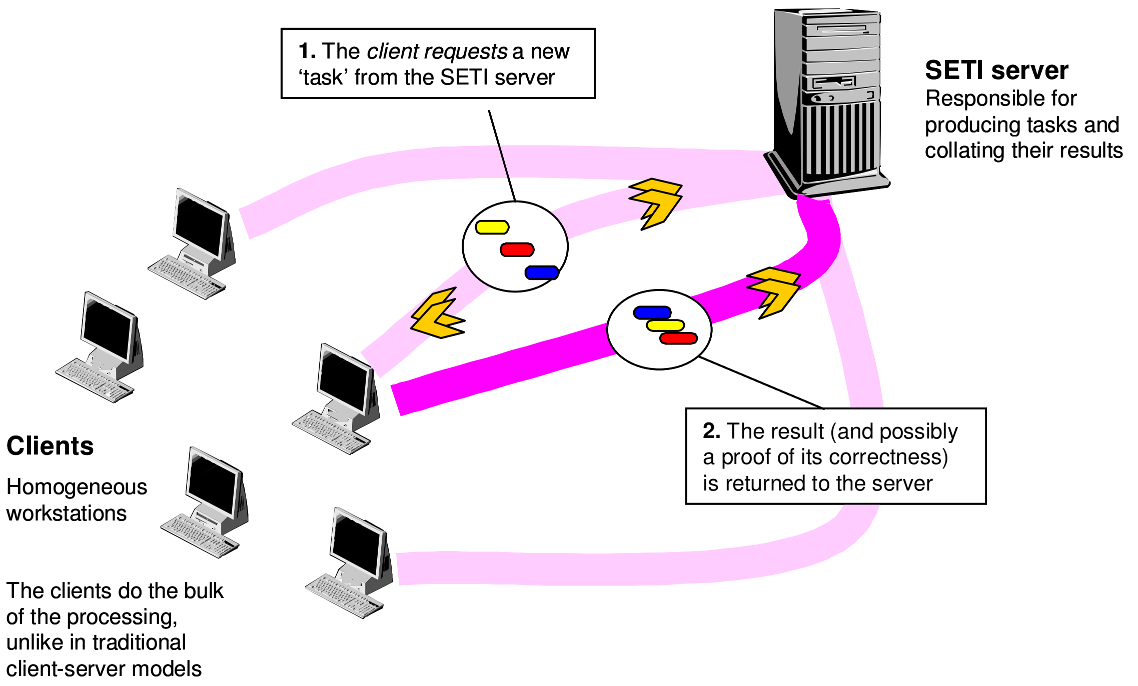
Attempts to exploit this vast processing resource have been limited: the granularity of computations required in many applications is quite small, individual nodes are unreliable, and external code and data distribution is hampered by relatively poor latency and bandwidth. The class of computation for which it makes sense are those that can be broken into many relatively small pieces, which require long individual processing times, but perform little communication.

The sort of computation to which peer-to-peer techniques have been applied are those that can take advantage of a very large set of processors that may vary in job throughput, but that are homogenous in the sense that they offer no specialised functionality. Problems in large-scale signal processing (SETI@Home [70], Genome searching [72], Astrogrid, and code cracking) have all been successfully tried, and continue to thrive[6].

Since peers in such systems have typically been home users with slow and intermittent connections, highly parallelisable tasks have been better suited. Recent consideration of inter-node proximity in peer-to-peer systems might lead to the use of clustering, allowing computations of a less constrained style (e.g. in Computational Fluid Dynamics: in turbine design, and atmospheric modelling).

Condor [49] is a system for utilizing idle workstations for user-submitted computation. Multiple computers form a *pool*, managed by a single computer that matches waiting jobs with available machines according to resource specifications. Condor features transparent checkpointing, where a job's state is serialized periodically to the pool's central repository: this allows a job to be migrated when a workstation becomes unavailable (e.g. its user returns), and to mitigate against loss of computation due to node failures. Its GlideIn mechanism allows Globus [71] resources to join a condor pool.

[6] SETI is the Search for Extra Terrestrial Intelligence, which takes the signal from a radio telescope, and attempts to find low entropy strings in it which may or may not represent *artificial* signals from remote intelligent life. Different peers process different portions of the recording, divided up in terms of time, direction and position in the sky.



1. The *client requests* a new 'task' from the SETI server
SETI server
Responsible for producing tasks and collating their results
Clients
Homogeneous workstations
The clients do the bulk of the processing, unlike in traditional client-server models
2. The result (and possibly a proof of its correctness) is returned to the server

While Condor's harnessing of nodes' own resources for their mutual benefit (i.e. higher job throughput when a user needs it) is characteristic of systems such as Seti@Home and Napster, it does not share peer-to-peer's assumption that participants may be very numerous, mutually distrustful, widely-dispersed and pseudonymous. At best, Condor requires a relatively static localised user community, as jobs inherit permissions from user identities in underlying operating systems and shared file systems. Worse, some configurations require out-of-band trust relationships between users. Limited restrictions on a process' activity may expose machines to malicious code.

Condor's centralised repositories design limits its inherent scalability. A 'flocking' mechanism exists to spill-over jobs to pools, but this is limited: joins are by coordination of central managers, must be statically-configured, and are suitable only for tasks requiring few privileges. Further, as a job is only 'flocked' if no free host exists in a pool, Condor may not make the best match for a job's requirements.

Figure 7: SETI@home is an efficient peer-to-peer use of computational resources. The communication : computation ratio is particularly low, so the clients do the bulk of the processing and the server merely checks and collates the results, unlike in traditional client-server systems where the server often does most of the work.

ited
ered
ion.
ring
the course of execution. The XenoServers architecture will use peer-to-peer techniques to provide resource discovery, a shared file system, and job migration; it will itself form a platform on which users may carry out accountable distributed execution.

3.4 Distributed Searching

Further to distributed file sharing systems that search using filename metadata, and DHTs with key-based searching, there has been work in applying generalised metadata search functionality over DHTs. These systems operate within the context of resource discovery, and may impact on similar service discovery mechanisms in Grid computing.

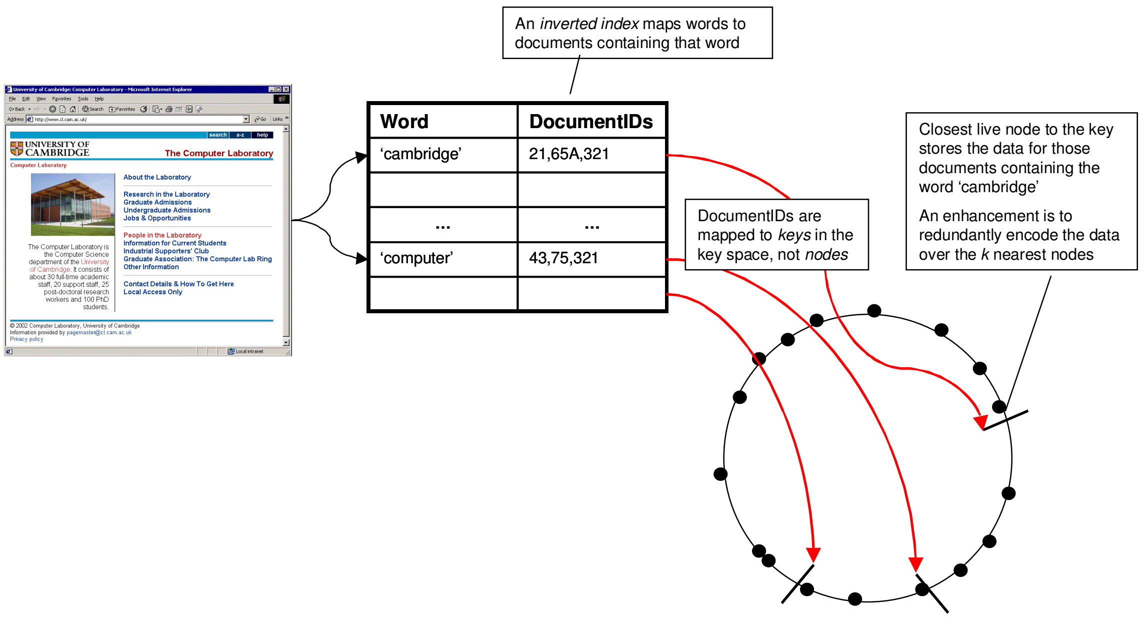

An *inverted index* maps words to documents containing that word
Word
DocumentIDs
'cambridge'
21,65A,321
...
...
'computer'
43,75,321
DocumentIDs are mapped to *keys* in the key space, not *nodes*
Closest live node to the key stores the data for those documents containing the word 'cambridge'
An enhancement is to redundantly encode the data over the *k* nearest nodes
UNIVERSITY OF CAMBRIDGE
The Computer Laboratory

Figure 8: Distributing an inverted index over a distributed hash table, such as Pastry's circular key space. An inverted index maps keywords to documents containing those words. A web page containing certain keywords is found by intersecting (using bloom filters passed between peers) the sets of possible documentIDs.

Research in the area of information retrieval has led to two main systems PlanetP [14] and pSearch [44]/Sedar [25]. These use the Vector Space Model (VSM) to represent documents and therefore these complex searches become similarity searches in a Vector space. These searches are carried out in pSearch by using a CAN [35] network to route requests and in PlanetP by summarizing using Bloom filters [9] and `gossiping' using the `Name dropper' algorithm [22].

Another vector space searching system is Multi-Dimensional Pastry [73], an extension to Pastry. It represents each dimension of the query as a Pastry ring, then uses Bloom Filters to summarize each dimension at various levels: these allow a query to combine *ranges* of each dimension by union and intersection, instead of being a simple hypersphere search. CAN has also been extended to support resource discovery within Grid systems [5].

3.5 Developing Peer-to-Peer Software

Software for peer-to-peer systems is symmetric; such software systems are harder to write than client-server systems. However, the commonality of the use of Java in some research projects, combined with the observation that there are enough common components in peer-to-peer development activities, has led to some common toolkits for future work - one such is JXTA [43] from Sun.

The peer-to-peer programmer needs to build event driven systems, which in general have proved harder to write correctly than client/server. SEDA [74], a framework to support event-driven programming in Java, has been utilised to this end by the Oceanstore [24] project.

One risk in such systems is that one has to pay great attention to synchronisation effects. Another problem is that the programmer must cope with the type of erroneous requests that only server (rather than client) application programmers have to deal with. This makes peer-to-peer programming currently an expert systems programmer task, rather than the separation of concerns that client-server architecture achieves. The separation in many areas of a peer-to-peer system into routing substrate or application can serve to reduce development complexity greatly, however.

4 Properties and Issues

In this section we explore various properties of peer-to-peer computing: some aspects to which are present in current systems, some still the subject of on-going work. We draw out the relation and relevance to Grid computing in each section.

4.1 Harnessing Resources

In general, peer-to-peer systems treat resources as *homogenous*, and peers as *individually dispensable*, and therein lies many of their strengths and weaknesses. Simplifying assumptions derive these conclusions naturally from the peer-to-peer fault model: one of an unreliable infrastructure and mutually distrustful participants. The design of DHTs embodies these assumptions. Any node is equally likely to be responsible for one particular key: so is assumed to be equally suitable to carry out a task related to it. Nodes carry similar numbers of keys: so it is assumed that their resources to store or manage them are assumed equal.

However, peers are unlikely to have similar resources – either in quantity or quality – or reliability characteristics. Systems that recognise this benefit in terms of performance and availability. For example, super-peering in file sharing systems takes advantage of well-connected nodes to implement distributed caching and indexing schemes.

Sometimes, use of a DHT routing substrate hinders an application’s ability to recognise heterogeneity of resources. In CFS [15], each node offers a globally-fixed amount of disk space as storage for blocks inserted by other nodes. Nodes with substantially more spare capacity can run separate *virtual nodes* from the same physical machine each offering this fixed unit. Even though CFS employs routing table ‘snooping’ to avoid increased lookup path lengths (because virtual nodes effectively increase the size of the network), it weakens assumptions about independent failure of nodes.

Grid systems, on the other hand, tend to comprise fewer, more varied, more specialised resources; each resource’s properties are described and published, and individual work units matched to a provider. A DHT may be viewed as a scalable property-blind resource allocator; the challenge is to evolve scalable node management and request routing to perform matching for complicated work unit and resource properties.

4.2 User Connectivity

The nature of peer’s network connections is an essential consideration when designing peer-to-peer systems that have practical use in established user communities. The problem is two-fold: mean connection quality is low, but at the same time it has a high variance: represented by the differences between dial-up, broadband, and connections from academic or corporate networks. This both severely limits the scope for generic inter-node communication, and requires applications consider the heterogeneity of peers' connections.

Other difficulties are encountered with peers that cannot accept incoming connections, either because their ISP uses NAT and as such have no externally-recognised IP address, or because they are behind an separately administered firewall. Several peer-to-peer projects exploit techniques to bypass firewalls[7], but these rely on institution-specific configurations. There is also unequal provisioning of upstream and downstream bandwidth in most broadband connections, and by 'capping' on permanent connections. The asymmetry of possible connections (and their bandwidths) between nodes complicates attempts to understand routing behaviour in real deployments.

Existing computational Grids tend to be comprised of participants connected by well-administered reliable academic networks. However, as generic Grid services begin to incorporate more diverse peers, these issues may become more important.

Studies of the deployment of Mojo Nation in [54] note an interesting pattern of user connectivity. It notes that established users rarely connected for more than a few hours at a time; indeed, ISPs offering 'unlimited' dial-up access for a lump payment often have a policy of disconnecting users after a set period. Further, in a peer-to-peer system, users will often join to test whether it offers a desirable service: in Mojo Nation, around 80% of peers left within an hour of connecting for the first time, and did not join again.

The Kademlia [26] routing substrate tries to take heed of this evidence; by only removing routing table entries for nodes that have failed, it favours using long-lived nodes for routing. However, it is not clear that this strategy is effective – routing table entries are passed rapidly, and tables are limited in size, so most nodes perform it adequately. Instead, it may just adversely load long-lived nodes. Instead, it is storage at a node that should favour longevity.

4.3 Collaboration and Trust

We have seen that resources within a GRID often form part of some trusted environment, often within physical organizations (or virtual organizations, VOs). In contrast, peer-to-peer systems have a number of characteristics that force us to consider the issue of collaboration. Peer-to-peer systems are often:

- *Globally-distributed.* Often, the owner of one `peer' has no real-world knowledge of the owner of another `peer'. So why should we interact with them?

- *Composed of peers are run by individuals (rather than corporations).* This has implications for uptime and reliability, leading to the observed power law uptime distributions for Gnutella, and the relatively small number of hosts that are reliably available.

- *Operated with no prior agreement between peers.* Consider a typical GRID computing task, such as processing a large amount of (possibly confidential) data. Before submitting the task to a GRID compute service, the submitter (consumer) often agrees on some terms and conditions with the service provider.

[7] Such as by using port 80, reserved for HTTP traffic, or 'forcing' incoming UDP by sending traffic to the equivalent port.

- *Composed of nodes which act hedonistically, in the interest of their own good.* This means we have to assume a mutually-distrustful environment, and that *without incentive to participate, a node will free-ride*, i.e. use the service without returning anything to it. In fact, free-riding is the norm rather than the exception in Gnutella.

Many peer-to-peer services rely on a cooperative model of interaction among nodes, yet actually provide little incentive for nodes to collaborate. We now consider some approaches that aim to address this.

4.3.1 Economic and game-theoretic approaches

Work on using economic models to realign nodes' incentives to participate have presented schemes that assume variable demand for services. Geels and Kubiatowicz argue in [75] that replica management in global-scale storage systems should be conducted as an economy. Nodes trade off the cost of storing data blocks with the value they are able to charge for access to them - in this context, a variable demand for blocks is essential.

However, variable demand properties may hold human-valued commodities, such as the information stored or shared in a peer-to-peer system, but not for routing table entries. Since DHTs typically determine the allocation of items to nodes pseudo-randomly, requests for keys will also be distributed evenly, so no particular value can be conferred on any particular destination.

Currently, the lack of a scalable, low-overhead digital cash system that may be fully decentralized hampers uptake of economic models. Mojo Nation [54], a distributed file storage and trading system, used a currency `mojo' to discourage free-riding and to obtain load balancing at servers by congestion charging, but relied on a centralized trusted third party to prevent double-spending.

Acqusti et al. [76] develop an incentive model for a distributed application that offers anonymity to its participants. They take a game-theoretic approach to analysing node behaviour and attacks in various system models. Trust is only considered as a means to ameliorate pseudospoofing attacks, rather than as a means to provide incentives to peers.

4.3.2 Trust and reputation frameworks

Aberer et al. [77] present a system for 'managing trust' in a peer-to-peer system using a complaint-based metric; nodes can make 'complaints' regarding interactions they have had with other nodes. They present a threshold technique for checking whether a node is untrustworthy, based on the difference between its recommendations and the average view. However, this presents a rather brittle and almost binary view of trust, which is often difficult to reason about explicitly.

The NICE system [78] aims to identify rather than enforce the existence of cooperative peers. It claims to "efficiently locate the generous minority [of cooperating users], and form a clique of users all of whom offer local services to the community". The system takes a novel approach to economics and trust; rather than using economics to model trust, it proposes using trust to model expected prices of services.

4.3.3 Explicit modelling of trust

Many approaches to enforcing or ‘incentivizing’ collaboration have been based on rather arbitrary measures. We developed a trust and security architecture [51] for a routing and node location service based on Kademlia, a third-generation peer-to-peer routing substrate based on a distributed hash table. Crucially, rather than ‘routing round’ defective or malicious nodes, the service discourages free-riding by requiring a node to contribute honestly in order to obtain routing service in return.

The paper proposes the idea that a `collaborative' service should have two desirable properties: *avoidance*, where dishonest nodes are “routed around” by those using the service (which is often the desired case for a hedonistic node); and *exclusion*, where dishonest nodes are unable to use the service because others refuse to carry out work for them (e.g. not forwarding packets).

We show how services such as Kademlia can be made ‘collaborative’ by using a *trust protocol* which describes how honest nodes perform, and a distributed trust model which allows the explicit modelling and, crucially, reasoning about trust in the network. The paper describes how our modified version of Kademlia is resistant to a number of attacks, including collusion.

4.4 Scalability

Peer-to-peer systems often add resources as they add customers. Thus they should scale (at least at the computing and storage resource level, if not networking) linearly, or better, with number of peers. Of course, many networks are designed assuming client-server traffic, and so it is entirely possible that this performance scaling properties may not be achieved transparently. There are some claims that the `small world' models of human behaviour and interaction, combined with the preferential way in which people link to known locations, lead to power law structures in connectivity.

Scalability is not a trivial consideration. While, for example, hops in a DHT may vary as $log(n)$ with the number n of nodes in the network, in a file system what is `acceptable' for access latency is constant and bounded by the user. It requires peer-to-peer designs to make good use of proximity information, to balance load well between nodes, and efficiently maintain state as nodes join and fail. For a file system, this means exploiting caching, predictive pre-fetching, and the apparent Zipfian power law distribution in the popularity of content.

Forming a structured topology is important in most peer-to-peer systems to provide bounds on latency and performance of inter-node communication. Approaches that structure and partition the keyspace tend to allow deterministic node location and better load balancing.

4.5 Proximity

Latency is an important consideration when routing in peer-to-peer systems. Poor proximity approximation in DHTs can involve a message travelling around the globe many times to reach its final destination. Several distributed applications aim to automate in the gathering of relevant proximity information; [47] assures us that point-to-point latency is constant enough over time to allow such systems to provide good approximations. Currently, these systems are IDMaps [17], GNP [30], Lighthouse [32], King [20], and for geographical position estimates [31].

Issues of “stretch” (distance travelled relative to the underlying network) become increasingly important in file sharing or storage systems where large quantities of data must be regularly transferred between peers. Further, given the observation that data in conventional file systems tends to be accessed most from where it was inserted, it becomes essential to store or cache data near to where it is accessed. This minimises load on the network and increases the rate at which data may be obtained.

A system simply storing blocks across a DHT is at odds with this – since nodes and keys tend to have pseudorandom identifiers, blocks will be assigned to a node regardless of that node’s position. However, when replicating data across *k* neighbours, which are likely to diverse in location, a DHT that takes into account locality (and so is likely to route through the *nearest* of these *k* nodes) provides some means of obtaining content from nearby peers.

Pond [67], the Oceanstore prototype, uses the Tapestry DHT to locate blocks, but supplements this with a probabilistic approach using Bloom Filters [9] that attempts to simultaneously obtain the same data from *nearby* peers, regardless of their location in the DHT.

4.6 Load Balancing

When nodes and keys are assigned psuedorandomly an imbalance in the allocation of keys to nodes will occur: the maximum load at any node will be *log(n)* the mean load, given an *n* node network. Of course, this also expects that any resources associated with keys are homogenous – making the same requirements of their destination node.

In PAST, an archival storage system, whole files are associated with a key and inserted into an underlying DHT. Because the size distribution of files is heavily skewed, the above imbalance is exasperated and a complicated scheme of storage management is required, where replicas are diverted to nodes with more storage space: one net effect is an increase in the average hop count (and so latency) required to retrieve files.

Performing load balancing in an environment of heterogeneous resources and competing job requirements is difficult, and requires a trade-off to be made between best allocation of job to resource and the rate at which job and resource properties are distributed.

4.7 Availability

Peer-to-peer networks experience a high rate of nodes joining and leaving both on account of their scale and the nature of their user communities. It follows that individual peers cannot be relied upon to maintain any essential state on their own.

As mentioned above, for purposes of redundancy most DHT-based applications store state at the *k* nodes with IDs numerically closest to the associated key. This replication invariant is maintained by local node cooperation, despite nodes joining or failing. If surrounding nodes maintain their routing tables correctly, this offers automatic fail-over: if the nearest node to a key fails, requests are automatically rerouted to the next closest node, which will also maintain state for that node.

Erasure coding schemes have been shown to offer the same availability for an order of magnitude lower overhead compared to deployed replication schemes. Data is encoded into *n* unique fragments; the coding ratio determines the proportion *m*/*n* of unique fragments that are required to recover the original data. However, since each fragment is unique, a simple local maintenance scheme does not suffice to maintain a data item's availability as different nodes fail.

The nature of network failures in the Internet is an important consideration for any practical system. Networks tend to fail along administrative boundaries, close to users, because of individual router or link failures. Conversely, individual peers are assumed to fail randomly (although we can see other patterns too). While most DHTs perform suitably under the latter failure model, a network failure of the former type tends to render most of the keyspace inaccessible (most likely all of it, unless locality is taken into account when choosing routing table entries).

SkipNet [52], however, is a routing substrate based on a data structure similar to a *skip list* and offers DHT-like data distribution, load balancing and routing at various administrative levels. Keys specify using a reverse-DNS notation the domain of peers over which they may be placed – allowing items to be distributed solely across nodes in the same organization. Further, a request for a key specifying a specific organizational domain will never be routed through nodes outside the same domain, maintaining access to that data even if a network failure separates that organization from the rest of the Internet.

Little attention has been paid to the effect of network partitions on systems in which partially or wholly independent fragments of systems are formed, update their own state, and then later rejoin. Quorum systems such as [68] have been used to enforce state consistency between peers updating replicated data [24], but the overhead of these schemes is prohibitive for use across a whole network. Instead, techniques from the literature on reconciliation of divergent file replicas may inform the semantics of peer-to-peer systems under network partition.

One exception is Ivy [79], a peer-to-peer file system where participants each maintain a log of the updates they make to the global state. Log entries have per-log sequence numbers and are tagged with version vectors, detailing the latest log entry seen for each other log in the system. Later, as each update carries sufficient information to determine the exact view of the global state at that time, conflict resolution can be performed to combine these states.

4.8 Anonymity and Censorship-Resistance

Some peer-to-peer systems offer privacy at the level that users' identities are masked. Some go further and mask content so that neither peer exchanging data knows who delivered or stores which content. True anonymity is typically two layer, requiring some form of anonymized IP routing system (e.g. onion routing) as well as application layer mechanisms to protect privacy.

One of the main novel characteristics of Eternity and subsequent peer-to-peer file systems has been the ability to withstand censorship. This is achieved by several means:

Partition

By splitting a file into component parts we make sure that no single site carries the whole file, and a denial of service attack has to run over multiple sites. Later systems made clever use of techniques such as Rabin fingerprinting or other techniques for finding common elements of objects can also ensure that overlapping content between multiple files is exploited to reduce storage costs.

Replication

By replicating blocks of a file over multiple sites, we also provide higher availability. Combined with locality information, this can be tuned to reduce latency and increase file sharing throughput, at some cost in terms of consistency in update.

Encryption

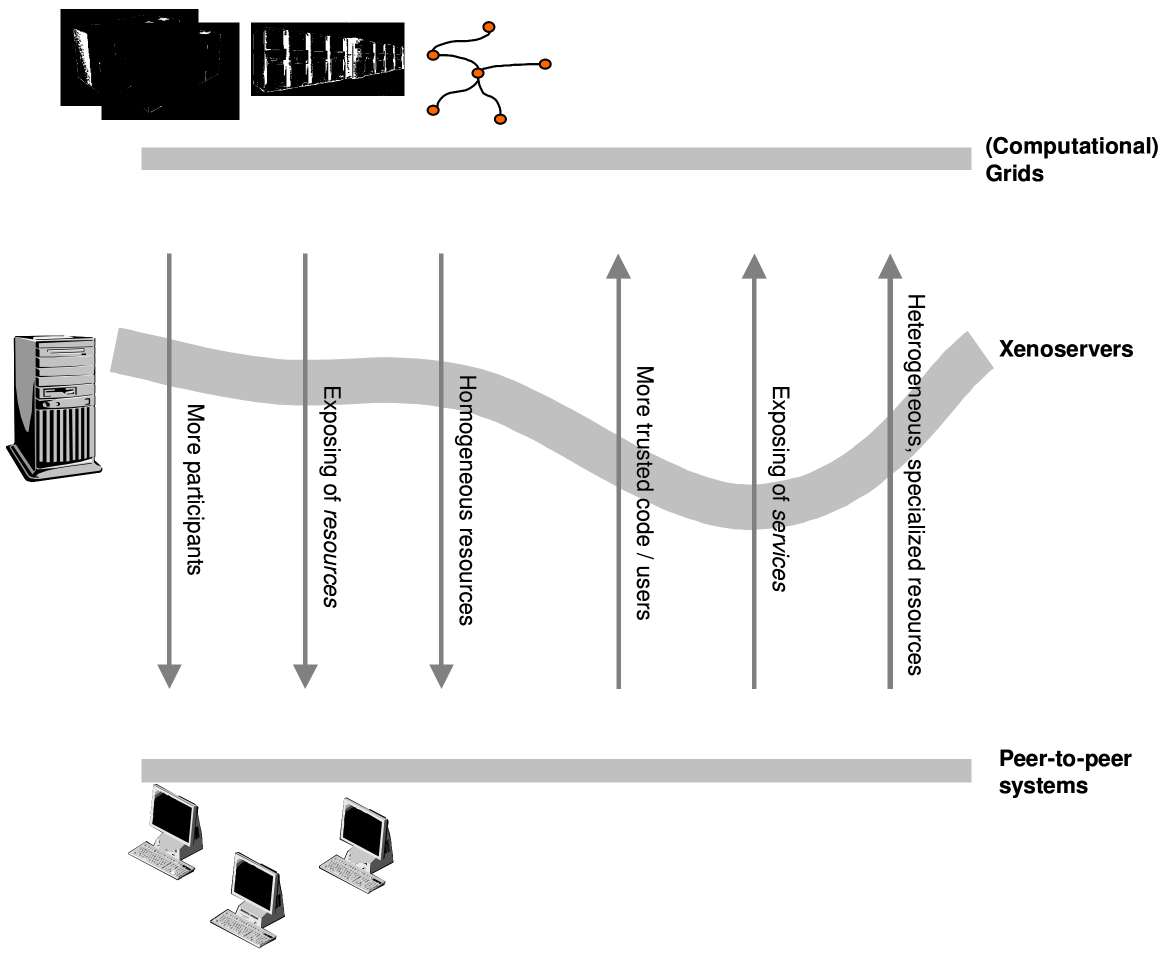
By encrypting blocks of the file, we make sure that disclosure is unlikely, but also that a node can deny knowledge of the actual content it carries - again, P2P exploits mutual benefit: the argument is that ``I might not approve of this content, but I approve of the ability to hide my content in a set of peers, so I will live with what I do not know here!". This is often termed ``plausible deniability" and is used by Service Providers to align themselves with the ``common carrier" defence against legal liability for content, as with telephony and postal providers can do.

Anonymization

By masking the identity of sources and sinks of requests, we can also protect *users* from the potential *censor* or unsavoury *agency*. However we need to mask location information as well as identifiers, as otherwise a traffic analysis may effectively reveal identities, so some form of onion routing is also usually required.

5 Future Directions

Figure 9: The apparent continuum between Grid and P2P computing styles



In this section, we peer into several interesting areas where we believe additional research questions can be identified, based on problems and shortcomings in current peer-to-peer systems. In describing the areas, we try to keep history in mind and how some of the problems relate to those past, present and possibly future, in Grid computing. Undoubtedly there are other ideas that equally belong here, but these are our current favourites.

5.1 Sharing Computation

Two systems epitomize the sum use of peer-to-peer technology thus far – Napster and SETI@home, or file-sharing and coarse-grained distributed computing. Peer-to-peer systems have been successful in supporting the former, but the latter represents the tip of the iceberg in distributed computing.

A critical difference between sharing files (and file transfers) and sharing computation is that the former are static, and so are partitioned easily: transferring one file is independent of another. 'Computation' is notoriously hard to distribute, yet there are some interesting cases that are well understood: these are situations with a low communication overhead, compared to the computation required.

In order to broaden the range of computation tasks admissible to a massive distributed peer-to-peer system, work is required in two areas. First, the infrastructure needs to handle closely-coupled distributed computation better, by exploiting self-organizing properties, staging and timing, and using innovative data transfer schemes to minimize the communication overhead. Additionally, algorithms should be *designed* to exploit peer-to-peer properties, by using, for example, asynchronous schemes to reduce dependency on low latency communication links.

5.2 Auditing and Accounting

SETI@home relies on users simply volunteering their CPU resources. Introducing an economic model where resources are bought and sold adds a new complication: accounting for their use.

The motivations for file and CPU resource sharing are not the same: it is not clear how to apply the mutual benefit arguments that work for file sharing to cycle sharing. Most file sharers' connections have some degree of separation in capacity provisioning and allow them to upload and download independently; to some extent, cooperation does not disadvantage them. However, the bursty nature of interactive use means the same is not true for any third-party sharing of a local CPU. Fine-grain accounting for resource sharing is required to limit, prioritise, and arbitrate between sharers, but how do we measure and enforce relative resource usage between different services?

The Xenoservers project [36] takes one approach. It intends to deploy machines running a *hypervisor* that performs accounting of commodities such as CPU cycles and network bandwidth. Principals may use these machines to deploy high level services, paying server operators for each service's use of low-level resources. In turn, other principals may be charged for accessing the high-level services. Xenoservers might be used a platform for both peer-to-peer and Grid services, but their accounting scheme has direct application in all distributed computation systems.

Peer-to-peer systems make accounting in general difficult, due to the coarse nature of jobs, the mutual distrust assumed between participants (the lack of an out-of-band trust relationship) and the possible short-term network presence of pseudonymous participants. Accounting in a Grid environment is easier – an approach and associated issues are described in a paper about GSAX [81] (Grid Services Accounting eXtensions), an extension to the OGSA standard. Accounting, and ultimately charging for, services at the application level suits the Grid computing paradigm since OGSA embraces a notion of *virtualization*, exposing resources as services.

5.3 Local Solutions to Achieve a Global Optimum?

Recent peer-to-peer systems select preferred neighbours to fill routing table entries by *proximity*. While it has been shown (e.g. in Pastry [39]) that making peering decisions locally can improve global routing properties (i.e. it minimises "stretch", routing distance relative to IP), such an approach is error prone: more importantly, it leads to a succession of local optimisations, rather than a solution *optimised* for global performance. In future, results from location services, such as those mentioned in Section 4.5, may be obtained and cached locally to improve proximity estimates and inform routing decisions.

We have a long way to go in providing performance guarantees of peer-to-peer systems: regardless of whether we want an adaptive system that responds to congestion (e.g. via explicit congestion notification or a pricing mechanism) or an engineered solution based on some type of signalling. In particular, high-level composite interactions in a peer-to-peer system often rely on the coordination of many peers – so it is difficult to base solutions purely on local decisions. However, as measurement projects produce more results to characterise peers' behaviour, we may be able to obtain good solutions through localised traffic engineering, admission control and scheduling algorithms.

The problem is related to providing incentives to participants. If we were to apply a market economy model, in which each peer is free to set its own prices for resources, and a stable global equilibrium is reached based on supply and demand, will an "invisible hand" mechanism globally optimise resource supply and utilisation? How do we ensure fairness? Lessons from economics will play an increasingly large part in the design of such systems.

5.4 Locality versus Anonymity

Peer-to-peer networks with potentially very large sets of participants offer an opportunity for obfuscating the activities of individual nodes – as described above, many early projects capitalised on this. Yet practical peer-to-peer systems struggle with the apparently inherent contradiction between offering anonymous sharing of resources, and the localisation of service offers. A number of factors are reducing the anonymity characteristic of peer-to-peer systems; at the very least, their immunity to traffic analysis is being lost as such techniques become more sophisticated.

Increasingly, anonymity-preserving features may be implemented as an overlay on top of peer-to-peer applications or middleware – Crowds [80] and 'onion routing' already take this approach to the web and email, respectively. Of course, the extent of the trade-off with performance lies in different users' requirements, and the degree to which particular applications need to exploit locality to obtain this.

5.5 From Overlay to Infrastructure

We have observed that many successful overlay systems migrate over time into the infrastructure itself, often to maximize the efficiency of the protocols but also to handle application-level security concerns. Indeed, the US National Academy of Science in their report *"looking over the fence at networking research"* [13] recommended looking at overlay systems as a general approach to building research infrastructures. As the nature of peer-to-peer networks and infrastructures become well understood, we might see the techniques migrate into the infrastructure, just as the handling logic for IP traffic has migrated from overlay networks into native services such as routers.

However, many peer-to-peer systems are complex, and devising a minimal service enhancement in the lower levels that would support their algorithms is an interesting challenge. The IETF FORCES working group has been working on remote IP router control (separating out packet forwarding and packet routing). Yet we need more than this if very general peer-to-peer intermediary functions are to be performed within time insignificant relative to packet transmission time. Furthermore, complex filtering and processing of content keys would be needed for maintaining a global-scale distributed hash table at the infrastructure level – not to mention the many hashes and signatures used in many current schemes. Such hopes show that we do not really understand what peer-to-peer actually means.

5.6 P2P and ad hoc wireless network duality

A defining characteristic of peer-to-peer systems is their ability to provide efficient, reliable and resilient routing between their constituent nodes by forming structured ad hoc topologies. In this respect, we can draw useful parallels with ad hoc wireless networking.

Resilient peer-to-peer mechanisms for content distribution services have been proposed, but the effect of these systems on global network utilisation is not well understood. Studies show that high topology maintenance and message routing overheads prohibit the use of such systems on wireless ad hoc networks, which suffer stringent power requirements and highly transient network connectivity.

An application of peer-to-peer techniques to mobile wireless ad hoc networks would involve making peers "load aware", by distributing load information and implementing a distributed congestion control scheme. To handle mobility, a topographical metric might be used for nodes to specify their own location. An *anycast* routing scheme, allowing a request to specify a set of satisfactory destinations, would be one approach to reducing message-passing overhead.

It is likely that next generation peer-to-peer systems will use actual location and scope information to influence routing functions so that content is initially placed and requests are routed to copies that have proximity on a number of quality of service (QoS) axes – often including delay, throughput and packet loss, but perhaps also battery considerations for wireless users. Thus the distribution of replicas in a content delivery or storage infrastructure would evolve to meet the user demand distribution, optimising use of the scarce wireless resources to better match user concerns.

6 Conclusion

There is no doubting the impact that peer-to-peer computing has had on mainstream computing, even to the extent of blurring the distinctions between computer science, engineering and politics. Unfortunately, the apparent phenomenon of these systems has prompted many attempts to reinvent the wheel without due consideration of much classic research in distributed systems. We hope that this chapter has shed light on the reasoning behind the success of peer-to-peer computing and its defining characteristics. Equally importantly, we hope we have made the reader aware of an interesting relationship between peer-to-peer and Grid computing styles, and what the future may hold for them.

We are entering the age of massively distributed, global-scale computing and storage systems where computing will change from a *commodity* to a *utility*. The peer-to-peer research wheel is now spinning quite rapidly; hence the work being done now will heavily influence this vision of utility computing. Our belief is that neither pure peer-to-peer nor pure Grid systems will be the winner, but rather some hybrid.

With that in mind, our goal in writing this chapter was to stimulate and offer food for thought for the peer-to-peer and Grid community. Peer-to-peer has a lot to offer the goal of utility computing – we just need to make sure we wear the right spectacles with which to see it.

Bibliography

1
ADAR, E., AND HUBERMAN, B.
Free riding on Gnutella.
Tech. rep., 2001.

2
ANDERSON, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R.
The Case for Resilient Overlay Networks.
Proc. *of the 8th Annual Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

3
ANDERSON, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R.
Resilient Overlay Networks.
In Proc. *18th ACM Symposium on Operating Systems Principles*, Lake Luise, Canada, October 2001.

4
ANDERSON, R. J.
The Eternity Service, June 1997.
http://www.cl.cam.ac.uk/users/rja14/eternity/ eternity.html.

5
ANDRZEJAK, A., AND XU, Z.
Scalable, efficient range queries for grid information services.
Tech. Rep. HPL-2002-215, Hewlett-Packard Laboratories, Palo Alto, 2002.

7
ASPNES, J., DIAMADI, Z., AND SHAH, G.
Fault-tolerant routing in peer-to-peer systems.
In *Twenty-First ACM Symposium on Principles of Distributed Computing* (Monterey, USA, July 2002), pp. 223-232.

8
BELLOVIN, S.
Security aspects of Napster and Gnutella.
Tech. rep.

9
BLOOM, B. H.
Space/time tradeoffs in hash coding with allowable errors.
Comm. of the ACM 13, 7 (July 1970), 422.

10
CHU, Y., RAO, S., SESHAN, S., AND ZHANG, H.
Enabling conferencing applications on the internet using an overlay multicast architecture.
Tech. rep., 2001.

11
CHU, Y., RAO, S., AND ZHANG, H.
A case for end system multicast.
Tech. rep., 2000.

12

CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W.
Freenet: A distributed anonymous information storage and retrieval system.
Lecture Notes in Computer Science 2009 (2001).

13
COMMITTEE ON RESEARCH HORIZONS IN NETWORKING.
Looking over the fence at networks: A neighbour's view of networking research.
Tech. rep.

14
CUENCA-ACUNA, F. M., AND NGUYEN, T. D.
Text-based content search and retrieval in ad hoc p2p communities, 2002.

15
DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I.
Wide-area cooperative storage with CFS.
In *Symposium on Operating Systems Principles* (2001), pp. 202-215.

16
DESHPANDE, H., BAWA, M., AND GARCIA-MOLINA, H.
Streaming live media over a peer-to-peer network.
Tech. rep.

17
FRANCIS, P., JAMIN, S., JIN, C., JIN, Y., RAZ, D., SHAVITT, Y., AND ZHANG, L.
Idmaps: a global internet host distance estimation service.
IEEE/ACM Transactions on Networking (TON) 9, 5 (2001), 525-540.

18
FREEDMAN, M. J., AND VINGRALEK, R.
Efficient peer-to-peer lookup based on a distributed trie.
In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)* (Cambridge, MA, March 2002).

19
GRIBBLE, S., HALEVY, A., IVES, Z., RODRIG, M., AND SUCIU, D.
What can peer-to-peer do for databases, and vice versa?

20
GUMMADI, K. P., SAROIU, S., AND GRIBBLE, S. D.
King: Estimating latency between arbitrary internet end hosts.
In *SIGCOMM Internet Mesurement Workshop 2002, Marseille, France* (November 2002).

21
HANNA, K. M., NATARAJAN, N., AND LEVINE, B.
Evaluation of a novel two-step server selection metric.
Tech. rep., 2001.

22
HARCHOL-BALTER, M., LEIGHTON, F. T., AND LEWIN, D.
Resource discovery in distributed networks.
In *Symposium on Principles of Distributed Computing* (1999), pp. 229-237.

23
JANNOTTI, J., GIFFORD, D. K., JOHNSON, K. L., KAASHOEK, M. F., AND
J. W. O'TOOLE, J.

Overcast: Reliable multicasting with an overlay network.
Tech. rep., 2000.

24

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B.
Oceanstore: An architecture for global-scale persistent storage.
In *Proceedings of ACM ASPLOS* (November 2000), ACM.

25

MAHALINGAM, M., TANG, C., AND XU, Z.
Towards a semantic, deep archival file system.
Tech. rep., Hewlett-Packard Research Labs, July 2002.

26

MAYMOUNKOV, P., AND MAZIÈRES, D.
Kademlia: A peer-to-peer information system based on the xor metric.
In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)* (March 2002), MIT Faculty Club, Cambridge, MA, USA.
http://www.cs.rice.edu/Conferences/IPTPS02/.

27

MICHAEL D. SCHROEDER, A. D. B., AND NEEDHAM, R. M.
Experience with grapevine: The growth of a distributed system.
ACM Transactions on Computer Systems, vol. 2, no. 1, pp. 3-23 (Feb. 1984.).

28

MORETON, T. D., PRATT, I. A., AND HARRIS, T. L.
Storage, Mutability and Naming in *Pasta*.
In *Proceedings of the International Workshop on Peer-to-Peer Computing at Networking 2002, Pisa, Italy.* (May 2002).

29

NAPSTER.
Napster media sharing system.
http://www.napster.com/.

30

NG, E., AND ZHANG, H.
Predicting internet network distance with coordiantes-based approaches.
In *INFOCOM'02, New York, USA* (2002).

31

PADMANABHAN, V. N., AND SUBRAMANIAN, L.
An investigation of geographic mapping techniques for internet hosts.
Proceedings of SIGCOMM'2001 (2001), 13.

32

PIAS, M., CROWCROFT, J., AND WILBUR, S.
Lighthouse: A QoS metric space to maintain network proximity.
UNPUBLISHED, October 2002.

33

PLAXTON, C. G., RAJARAMAN, R., AND RICHA, A. W.
Accessing nearby copies of replicated objects in a distributed environment.
In *ACM Symposium on Parallel Algorithms and Architectures* (1997), pp. 311-320.

34

R. GOLDING, E. B.
Fault tolerant replication management in large scale distributed storage systems.
In *Proceedings of Symposium on Reliable Distributed Systems* (1999).

35
RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S.
A scalable content addressable network.
Tech. Rep. TR-00-010, Berkeley, CA, 2000.

36
REED, D., PRATT, I., MENAGE, P., EARLY, S., AND STRATFORD, N.
Xenoservers: Accountable execution of untrusted programs.
http://www.cl.cam.ac.uk/Research/SRG/netos/ xeno/hotos1/index.html, November 1998.

37
RIPEANU, M.
Peer-to-peer architecture case study: Gnutella network.
In *2001 International conference on P2P computing* (August 2001).
http://people.cs.uchicago.edu/~matei/PAPERS/ P2P2001.pdf.

38
ROSCOE, T., AND HAND, S.
Transaction-based Charging in Mnemosyne: a Peer-to-Peer Steganographic Storage System.
In *Proceedings of the International Workshop on Peer-to-Peer Computing at Networking 2002, Pisa, Italy.* (May 2002).

39
ROWSTRON, A., AND DRUSCHEL, P.
Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems.
Lecture Notes in Computer Science 2218 (2001), 329-350.

40
ROWSTRON, A. I. T., AND DRUSCHEL, P.
Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility.
In *Symposium on Operating Systems Principles* (2001), pp. 188-201.

41
SHI, S., AND TURNER, J.
Routing in overlay multicast networks.
Tech. rep.

42
STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H.
Chord: A scalable Peer-To-Peer lookup service for internet applications.
In *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)* (New York, August 2001), R. Guerin, Ed., vol. 31, 4 of *Computer Communication Review*, ACM Press, pp. 149-160.

43
SUN.
Jxta peer-peer system, April 2002.
http://www.jxta.org/.

44

TANG, C., XU, Z., AND MAHALINGAM, M.
pSearch: Information Retrieval in Structured Overlays.
In *First Workshop on Hot Topics in Networking* (October 2002).

45

TRUELOVE, K., AND CHASIN, A.
Morpheus out of the underworld.
http://www.openp2p.com/pub/ a/p2p/2001/07/02/morhpeus.html?page=1.

46

ZEGURA, E., AMMAR, M., FEI, Z., AND BHATTACHARJEE, S.
Application-level anycasting: a server selection architecture and use in a replicated web service.
Tech. rep., 2000.

47

ZHANG, Y., PAXSON, V., AND SHENKER, S.
The stationarity of internet path properties: Routing, loss, and throughput.
ACIRI Technical Report (2000).

48

ZHAO, B. Y., KUBIATOWICZ, J., AND JOSEPH, A. D.
Tapestry: an infrastructure for fault-resilient wide-area location and routing.
Tech. Rep. UCB//CSD-01-1141, University of California at Berkeley, April 2001.

49

Litzkow, M., Livny, M. and Mutka, M.
Condor – A Hunter of Idle Workstations.
In Proc 8th Intl Conf. on Distributed Computer Systems, 1988.

50

Foster, I., and Kesselman, C.
The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.
Technical Report, Globus Project, 2002.

51

Moreton, T., and Twigg, A.
Enforcing Collaboration in Peer-to-Peer Routing Services
In *Proc. First International Conference on Trust Management*, May 2003.

52

Harvey, N., Jones, M. B., Saroiu, S., Theimer, M., and Wolman, A.
SkipNet: A Scalable Overlay Network with Practical Locality Properties.
In Proc *of Fourth USENIX Symposium on Internet Technologies and Systems (USITS ' 03,)*March 2003.

53

FIPS 180-1.
Secure hash standard.
Technical Report Publication 180-1, Federal Information Processing Standards, NIST, US Dept. of Commerce, April 1995.

54

B. Wilcox-O'Hearn.
Experiences deploying a large-scale emergent network.

In *Pro.c of the First International Workshop on Peer-to-Peer Systems (IPTPS ' 02)* Cambridge, MA, March 2002.

55

Security for structured peer-to-peer overlay networks.
Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D.
Submitted for publication.

56

Douceur, J.
The Sybil Attack.
In *Proc. of the First International Workshop on Peer-to-Peer Systems (IPTPS ' 02)* Cambridge, MA, March 2002.

57

Yang, B., and Garcia-Molina, H.
Efficient Search in peer-to-peer networks.
In Proc. of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS), 2002.

58

Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S.
Search and replacement in unstructured peer-to-peer networks.
In Proc. of the 16th ACM International Conference on Supercomputing (ICS), 2002.

59

Hong, T.
Performance.
In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, ed. A. Oram.
O'Reilly and Associates, 2001.

60

Karger, D., Lehman, E., Leighton, F., Levine, M., Lewin, D., and Panigrahy R.
Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web.
In Proc. ACM SOTC, May 1997.

61

Brocade: Landmark Routing on Overlay Networks.
Zhao, B., Duan Y., Huang, L., Joseph, A., and Kubiatowicz J.
In *Proc. First International Workshop on Peer-to-Peer Systems (IPTPS '02),* Cambridge, MA, March 2002.

62

Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B.
Design and Implementation of the Sun Network Filesystem.
In *Proc. Summer USENIX*, June 1985.

63

Kazaa Media Desktop.
http://www.kazaa.com/

64

Dublin Core Metadata Initiative
http://www.dublincore.org/

65

Dingledine, R., Freedman, M., Molnar, D.
The Free Haven Project: Distributed Anonymous Storage Service.
In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability,* 2000.

66
Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M.
Scale and Performance in a Distributed File System.
In *ACM Trans. on Computer Systems*, Feb. 1988.

67
Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., and Kubiatowicz, J.
Pond: the OceanStore Prototype.
In *Proc. the 2nd USENIX Conference on File and Storage Technologies (FAST ' 03),* March 2003.

68
Castro, M., and Liskov, B.
Practical Byzantine Fault Tolerance.
In Proc. OSDI 1999.

69
Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., and Hauser, C.
Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System.
In Proc. of 15th Symposium on Operating Systems Principles (SOSP-15) , Cooper Mountain, Colorado, 1995

70
Seti@Home Project
http://setiathome.ssl.berkeley.edu/

71
Foster, I., and Kesselman, C.
Globus: A Toolkit-Based Grid Architecture.
In *The Grid: Blueprint for a new Computing Infrastructure*, Foster, I and Kesselman, C., eds. 1999.

72
United Devices.
http://www.ud.com/

73
Spence, D., and Harris, T.
XenoSearch: Distributed Resource Discovery in the Xenoserver Open Platform.
In *Proc. 12th International Symposium on High Performance Distributed Computing*, 2003.

74
Welsh, M., Culler, D., and Brewer, E.
SEDA: An Architecture for Well-Conditioned, Scalable Internet Services.
In *Proc. 18th Symposium on Operating Systems Principles (SOSP-18)*, Banff, Canada, 2001.

75
Geels, D., and Kubiatowicz, J.
Replica Management Should Be A Game.
In Proc. *SIGOPS European Workshop* 2002.
76

76
Acquisti, A., Dingledine, R., and Syverson, P.
On the Economics of Anonymity.
http://freehaven.net/doc/fc03/econymics.pdf

77
Aberer, K., and Despotovic, Z.
Managing trust in a peer-to-peer information system.
In ACM Conference on Information and Knowledge Management, 2001.

78
Lee, S., and Bhattacharjee, B.
Cooperative Peer Groups in NICE.
In Proc. *IEEE Infocom,* 2003.

79
Ivy: A Read/Write Peer-to-peer File System.
Muthitacharoen, A., Morris, R., Gil, T., and Chen, B.
In *Proc. of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI ' 02,)*2002.

80
Reiter, M., and Rubin, A.
Crowds: anonymity for Web transactions.
In *Proc. ACM Transactions on Information and System Security,* 1998.

81
Beardsmore, A., Hartley, K., Hawkins, S., Laws, S., Magowan, J., Twigg, A.
GSAX Grid Service Accounting Extensions
http://www.gridforum.org/meetings/ggf6/ggf6_wg_papers/ggf-rus-gsax-01.doc

82
Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron A., and Singh, A.
SplitStream: High-bandwidth content distribution in a cooperative environment.
In *Proc. IPTPS' 03,*Berkeley, CA, February, 2003.