Workshop Report for IPTPS'02
1st International Workshop on Peer-to-Peer Systems

7-8 March 2002 – MIT Faculty Club, Cambridge, MA, USA

Richard Clayton

University of Cambridge, Computer Laboratory, Gates Building,
JJ Thompson Avenue, Cambridge CB3 0FD, United Kingdom
richard.clayton@cl.cam.ac.uk

Attendees were welcomed to the Workshop by Frans Kaashoek and Peter Druschel who reported that although the original plan had been to invite 35 people to a $1\frac{1}{2}$ day event, 99 papers had been submitted. The workshop had therefore been scaled up to 50 authors with 32 position papers, 26 of which would be presented over a two day program.

Session 1: DHT routing protocols: State of the art and future directions. *Chair: Scott Shenker*

Scott Shenker gave a brief overview of the problem space, noting that a great deal of work was being done on Distributed Hash Tables (DHTs) that were location independent routing algorithms. Given a key, they would find the right node.

***Ben Y.Zhao, Yitao Duan, Ling Huang, Anthony D. Joseph, John D. Kubiatowitz*, "Brocade: Landmark Routing on Overlay Networks", presented by Ben Y. Zhao.** Recent advances in high-dimension routing research guarantee a sub-linear number of hops to locate nodes. However, they assume a uniformity of mesh that does not map well to the Internet. Brocade extends Tapestry to address this by eliminating hops across the wide area and by preventing traffic going through congested stub nodes. "Super-nodes" are spotted within connected domains and a secondary overlay layer is then used to transfer traffic between these super-nodes. Brocade is not useful for intra-domain messages, so it is necessary to classify traffic by destination. Old results for locality are cached and the super-node keeps a "cover net", an authoritative list of which nodes are local. The local super-nodes are found by snooping peer-to-peer traffic or by consulting the DNS. The super-nodes find each other by using Tapestry, which has built-in proximity metrics. The super-nodes do not need to be incredibly powerful. Simulations show that with 220M hosts, 20K AS's and 10% of the super-nodes "coming and going" then about 9% of the super-nodes' CPU time is spent dealing with changes. Publishing data about the changes uses about 160K bits/sec in bandwidth, and super-nodes need only about 2MB of storage. Simulations also show that the scheme halves delay for short hops and improves overall bandwidth usage.

Discussion: Q: Doesn't RON show the benefit of routing around problems in the wide area? A: Brocade doesn't preclude this and, naturally, Tapestry allows for routing around problems in its discovery protocols. Q: What about the cost of the Tapestry protocol itself? A: We try to optimize "behind the curtain" in the background so as not to affect performance. Also a lot of the behavior is local and nodes are replaced by a proximity network. Bottom line is we're getting a performance improvement of 50%, which is worth having and in the limit, you can always fall back into normal routing.

***David Liben-Nowell, Hari Balakrishnan, David Karger* "Observations on the Dynamic Evolution of Peer-to-peer Networks", presented by David Liben-Nowell.** The existing approach to describing DHTs is to carefully define an ideal state and show that whilst in that state the network has good properties. However, since this ideal state never happens "who cares?" Networks are dynamic and nodes come and go very frequently. One has to prove that the system gets back to an ideal state "soon after all the changes". Perhaps it would be better to define an "almost ideal state" that would be achievable in practice. One would require this state to be maintainable and for it to have good properties like fast search, good balancing and lookups that succeed when they should. With this approach traditional performance measures become meaningless. Because the network never stabilizes, the time to recovery and number of maintenance messages is infinite. A better measure would be to use the doubling time (until the network is twice the size) or the halving time (until $n/2$ nodes fail). The "half life" is the smaller of these and measures how long it is until only half of the system is as it was before. It is possible to prove that all peer-to-peer systems must be notified of $\Omega(log\ n)$ node changes every half-life and that systems will stay connected if nodes discover $log\ n$ new nodes per half-life. In Chord it is possible to find a protocol that runs in $O(log^2 n)$ messages – suggesting that further work is needed. It would also be helpful to understand the notion of half-life more clearly and measure some real values in the field.

Discussion: Q: Isn't joining or leaving from real networks somewhat bursty? A: If the changes aren't massive the proposed Chord protocol will cope. Q: What about things like BGP level events changing connectivity? A: The model may be broad enough to cope with this. More work is called for.

***Sylvia Ratnasamy, Scott Shenker, Ion Stoica*, "Routing Algorithms for DHTs: Some Open Questions", presented by Scott Shenker.** Routing algorithms for DHTs have more commonalities than differences. This is an illustrative list of what the open research questions seem to be, along with some initial answers:

Q1: What path lengths can you get with $O(1)$ neighbors?
Viceroy seems to manage $O(log\ n)$ and Small Worlds $O(log^2 n)$.

Q2: Does this cause other things to go wrong?
We don't yet know.

Q3: What are the costs and dynamics of full recovery? Viz: if some nodes fail can you still reach the ones that remain?

It seems that most systems are remarkably good, losing 2% of routes with a 20% node failure rate. However, maybe routing isn't the issue, but data replication is. Perhaps we will have lost 20% of the data?

Q4: Can we characterize the effectiveness of proximity (geographic) routing?

We've not yet got a good model to show that this works, although it is clear that it does.

Q5: Is proximity neighbor selection significantly better than proximity routing?

Yes, a little better. But there's no good model yet.

Q6: If we had the full n^2 latency matrix would one do optimal neighbor selection in algorithms not based on Plaxton trees?

Q7: Can we choose identifiers in a *1-D* keyspace that adequately captures the geographic nature of nodes?

Q8: Does geographic layout have an impact on resilience, hot-spots and other aspects of performance?

We expect load balancing to be hard!

Q9: Can the two local techniques of proximity routing and proximity neighbor selection achieve most of the benefit of global geographic layout?

We don't yet have a way of doing this comparison.

Q10: Nodes have varied performance, by several orders of magnitude. If powerful nodes pretend to be multiple less "able" nodes is this "cloning" effective?

Q11: How can we redesign routing algorithms to exploit heterogeneity?

and from this list, the burning questions are Q2, Q9 and Q11.

Discussion: Q: Doesn't the improvement from proximity routing depend on the number of nodes? A: We're looking for asymptotic behavior. Q: Doesn't load balancing interact poorly with security? A: It would be very useful for papers to clearly indicate how they felt security properties emerged from their routing properties. If the system is extended or optimized one would like to check that the security properties still hold. Q: Can you separate out two levels of behavior, one for correctness and one for speed? A: We tried to do this with CAN. Q: Aren't we being unfriendly by designing our own routing level which is moving away from the underlying TCP? A: This may not be unfriendly, but the way forward may be to re-examine the notion of proximity to relate it to a graph of the actual system, rather than to a mere count of hops.

Session 2: Deployed peer-to-peer systems *Chair: Roger Dingledine/Steve Gribble*

***Matei Ripeanu, Ian Foster*, "Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems", presented by Matei Ripeanu.** Gnutella is a large, fast growing peer-to-peer system. Further growth is being hindered by inefficient resource use: the overlay network does not match the underlying network infrastructure. This study examined the network during a period of rapid change from 1,000 nodes in November 2000 to 50,000

nodes in May 2001. By March 2002 the Gnutella network was about 500,000 nodes. The network grew partly because its users were prepared to tolerate high latency and low quality results for file searches. Also, DSL connected users were 25% of the network at the start of the study and about 40% at the end.

Gnutella was originally a power-law graph (with the number of nodes proportional to L^{-k} for some value of k), but by May 2001, at the end of the study, it had become bi-modal (there were too many nodes with low connectivity). Also, the average path length had grown only 25% rather than the expected 55%. This made Gnutella more resilient to random node failures. Each link was transferring 6-8 Kbytes/sec so the overall administrative traffic in May 2001 was about 1 Gbyte/sec, about 2% of the US "backbone" traffic levels. 90% of this volume is query and ping traffic. The overall topology does not match the Internet topology with 40% of the nodes in the 10 largest ASs and the "wiring" is essentially random.

Discussion: Q: Why were there some periods of very rapid change? A: Bearshare changed their protocol so that saturated nodes no longer answered pings. Over about a week Gnutella stopped being a power-law network as the Morpheus system started using it. Q: Are clients maturing and allowing more scaling? A: Gnutella is now a two-layer system, which was a change. Improvements have mainly come from the protocol changes (pings dropped from 50% of traffic to 5%) but also to some extent from better engineered clients.

***Qin Lv, Sylvia Ratnasamy, Scott Shenker*, "Can Heterogeneity Make Gnutella Scalable?", presented by Sylvia Ratnasamy.** The most common application of fielded Peer-To-Peer systems is file sharing. The current solutions are unstructured, the overlay is ad hoc (you can connect as you wish) and files may be placed almost anywhere. The only approach possible is random probing and unfortunately, poorly scaling ways of doing this have been chosen. DHTs are very structured and very scalable so they are good for finding a "needle in a haystack". But perhaps the unstructured solutions are "good enough", especially when one is looking for "hay", i.e. material of which there are many copies in the network. DHTs are bad at keyword searches (because they do exact matches) and cope poorly with rapid changes of network membership. A system like Gnutella has no structure to lose if many nodes leave or crash.

Gnutella would perform better with a scalable search – multiple "flood to one" random walks give a big performance gain. Biasing this to deal with node heterogeneity (Gnutella nodes have differences of 4 orders of magnitude in available bandwidth) or transient load changes would also help with the desirable state of "big nodes do more work". The idea is for nodes to have a capacity measure and replicate files and dump traffic onto less full neighbors. Where traffic cannot be dumped, the sender is asked to slow down. Results from simulations are encouraging, suggesting that although having "super-nodes" is a good idea, it may not be necessary to be explicit about having exactly two levels of traffic.

Discussion: Q: Can you measure performance directly rather than relying on a neighbor telling you that you're slow? A: Yes, we're aiming to do that. Q:

What are DHTs good for? A: "needles". Some problems are like that, though the type of improved Gnutella we want to build will be better at needles than Gnutella currently is. If DHTs were required to do keyword searches then they might be just as expensive.

***Bryce Wilcox-O'Hearn*, "Experiences Deploying a Large-Scale Emergent Network", presented by Bryce Wilcox-O'Hearn.** Mojo Nation was originally inspired by Ross Anderson's Eternity paper in the mid 90s. In 1998 it was put onto a commercial footing as a distributed data haven, but ran out of money in 2001. Mnet is the open source descendant.

Mojo Nation was an ambitious and complex system that incorporated digital cash as payment for resources. The digital cash worked and most of the rest did not. It didn't scale (never reaching 10,000 simultaneous nodes). 20,000 new people tried it each month, but its half-life was less than an hour. There were significant problems with the first connection to the system, which was needed to locate any neighbors. When the system was publicized the ensuing wave of people overwhelmed the server ("Slashdot killed my network"). "Original introduction" is an important and non-trivial issue and other systems such as LimeWire can be seen to be having problems with it. Of those who did connect 80% came, looked and left forever. This appears to be because there was no content they cared about on the system. Data had to be explicitly published and nodes didn't do this. The content was distributed as a 16 of 32 "erasure code" scheme, but 80% of all nodes were offline most of the time, so files could not be recreated. There were also problems with ISPs ("my enemy") who throttled bandwidth, forbade servers or changed their IP addresses on a regular basis. The original design was tunable to suggest network neighbors based on reliability and closeness (Round Trip Time). This formula was progressively updated until eventually RTT was scored at an extremely low level of significance.

Discussion: Q: How important was the mojo? A: This Chaumian blinded digital cash determined who could store files. It was complicated to make it secure, and people were always trying to steal it by exploiting bugs. It was also hard to verify whether services were actually performed. You needed something to discriminate against newcomers Q: Do we need a lightweight rendezvous protocol to solve this original introduction problem? A: Yes, but it needs to have no single points of failure or control, because people will want to control it. Q: Why use shares rather than multiple copies of documents? A: Less disk usage overall. Q: Would low churn have helped? A: Yes, it did work well sometimes, when nodes weren't coming and going quite so much. Q: Can one actually run a commercial file storing service? A: I'm not the person to ask.

Session 3: Anonymous overlays *Chair: Roger Dingledine*

***Andrei Serjantov*, "Anonymizing Censorship Resistant Systems", presented by Andrei Serjantov.** The idea of censorship resistance is to make it hard for someone more powerful to remove content from a distributed filestore.

Anonymity is also important, for publishers, readers and whoever may be storing the file. Issues such as searching or efficiency are of less concern – it matters more that the material is available than that it took a long time to arrive.

The protocol assumes a DHT system that can store keys and inter-node communication via anonymous addressing "onions". The basic idea is to split the document into shares and ask for the item to be stored. Nodes called forwarders encrypt the shares, select the actual storage points and return identifiers that will later be used to identify the routes to these storage locations. The identifiers are combined by the publisher and the resulting address is then publicized out-of-band, perhaps by anonymously posting it to Usenet. Retrieval again uses the forwarders, but the stored shares are returned to the retriever via separate nodes that remove the encryption. The various roles in the storage/retrieval protocol have strong guarantees. The storers of the document are unaware what they are storing. The nodes that forward requests can deny doing this and the publisher and any retriever can deny almost anything. Future work will prove the properties of the system in a formal way and will demonstrate the resistance of the system to attacks.

Discussion: Q: Doesn't the forwarder know what's going on? A: They're just moving random numbers around. Q: Why not publish the storer onions directly? A: This would allow an attacker to take out the start of the MIX chain and deny access to the document. It's also possible to adjust the protocol to prevent the forwarders being vulnerable in this way.

***Steven Hazel, Brandon Wiley*, "Achord: A Variant of the Chord Lookup Service for Use in Censorship Resistant Peer-to-Peer Publishing Systems", presented by Brandon Wiley.** The idea was to create a Chord variant that would provide a DHT with anonymity properties. This ideas are #1 not to report intermediate lookup progress (so that the storer remains hidden), #2 make node discovery harder by restricting `find_successor`, #3 to be careful about how finger tables are updated by restricting the information flow. This third property proves to be hard because "stuff for my finger table that would be best for me" will vary and therefore, over time, nodes will learn more and more identifiers for participants in the Chord ring. Rate limiting merely delays this process. The other properties mean that nodes can hide, which makes them wonderfully anonymous, but may collapse the system to a small number of nodes. Finally, all these changes make Chord rather slower and this may mean that stability is rarely achieved.

Discussion: Q: Why do you want to restrict learning of other nodes? A: If you know the identity of a storer then you can censor them. Q: Can't you use TCP to ensure that nodes are who they say they are? A: Yes, that helps a lot with authentication. Q: Won't IPv6 mess this up by letting people choose their own IP addresses and enter the Chord ring close to what they want to learn about and censor? A: We need another identifier with the right properties. We aren't alone in wanting this!

***David Mazières* gave a talk on "Real World Attacks on Anonymizing Services".** Anonymous speech can upset people, so they try to shut down the systems that propagate it. They try to exploit software vulnerabilities, they use the system to attack someone powerful enough to themselves attack the system, they try to marginalize the system, they attract spam to the system, and they try to make life intolerable for the operator. Some attacks can be defeated by short-term logging or by providing some logs to users. Overloading can be addressed by trying to force the attacker to put a human into the loop, so as to increase the cost. Content based attacks can be countered by ensuring that it is easy for people to ignore anonymous content and by never storing or serving objectionable content. These issues must all be factored into the design of any anonymous service, where the most precious resource will be human time. This can be used by the defenders to slow down attackers, but also by attackers to wear down the operator of the service. Future work is looking at Tangler, which will entangle multiple documents together to make them harder to censor.

***Michael J. Freedman, Emil Sit, Josh Cates, Robert Morris*, "Tarzan: A Peer-to-Peer Anonymizing Network Layer", presented by Michael Freedman.** Tarzan provides a method for people to talk to servers without anyone knowing who they are. Millions of nodes will participate and bounce traffic off each other in an untraceable way. Peer-to-peer mechanisms are used to organize the nodes and the mechanism works at the IP layer so that existing applications will not need modification. Because so many nodes take part, it will not be practical to block them all. Because everyone is relaying for everyone else, there is no "network edge" at which to snoop traffic, and because relayed traffic cannot be distinguished from originated traffic there is plausible deniability.

When you join the system you make random requests from a Chord ring to determine a list of possible peers and select a small number of these to create a source routed UDP based tunnel with each hop secured by a symmetric key. The nodes on the path hold a flow identifier to determine what action to take with incoming traffic. NAT is used both within the tunnel and also at the far end where a connection is made to the true destination. Tarzan can also support anonymous servers. A C++ based prototype has been created that was able to saturate a 100Mbit/sec Ethernet. The overhead for setup is ∼20ms/hop and for packet forwarding ∼1ms/hop (each plus the data transmission time).

Discussion: Q: How well does TCP run over the tunnel? A: There are no detailed figures yet, and it is hard to say what effect rebuilding tunnels will have. Q: How capable does an observer have to be to break this system? A: Few are big enough. Q: How reliable does the PNAT at the end of the tunnel need to be? A: For things like SSH this is an issue. We may need to have a bit less heterogeneity in node selection. Q: What about cover traffic? A: We need traffic in the network to hide our traffic. So we create a sparse overlay network with this traffic and mix this with longer hops. Q: If you know two ends A and B can you say that you cannot link A with B? A: Yes, we hope to do this reasonably well, but it depends what "cannot" means.

Session 4: Applications I *Chair: Frans Kaashoek*

***Steven Hand, Timothy Roscoe*, "Mnemosyne: Peer-to-Peer Steganographic Storage", presented by Steven Hand.** Mnemosyne (pronounced *ne moz'nē*) concentrates on high value, small size, information. It is not an efficient global storage system, but is aimed instead at providing anonymity and anti-censorship properties. The basic idea is that blocks within the (highly distributed) system are filled with random data. The file is encrypted so that it too looks like noise and the data is then placed pseudo-randomly into the store. Collisions are clearly a problem, and although these could be dealt with by writing each block several times, it is preferable to use Rabin's Information Dispersal Algorithm instead. Traffic analysis is a problem, but it can be countered by routing data through many nodes, avoiding fetching all of the shares and by reading information that isn't required. Writes are more of a problem since specious writes would damage data unnecessarily. A working prototype has been created using Tapestry as an underlying peer-to-peer system; it runs at about 80Kbytes/sec reading and 160Kbytes/sec writing. The crypto aspects are currently being assessed. The longer-term aim is to use multiple DHTs and ensure they anonymize traffic. It is hoped to construct a commercial multi-homed data storage system.

Discussion: Q: Are locations chosen randomly? A: Not entirely, there is some directory and file structure present. Q: Do you need to check if data is still there? A: Yes, because after a while it disappears. For some levels of availability the files may need to be refreshed. However, disappearing may be an appropriate fit with an application such as a personal messaging system. Q: How does it compare with OceanStore? A: It's not solving the same problem.

***Sameer Ajmani, Dwaine Clarke, Chuang-Hue Moh, Steven Richman*, "ConChord: Cooperative SDSI Certificate Storage and Name Resolution", presented by Sameer Ajmani.** SDSI is a proposed decentralized public key infrastructure that allows for mapping of principals (keys) to locally specified names and the use of chains of certificates to delegate trust to other organizations via the use of groups of principals. Name resolution becomes a key location problem that turns out to be hard because of the local namespaces, global loops and other complications. ConChord computes derived certificates to eliminate loops and make lookups fast, but this means significant extra work is needed for insertions and significant additional storage is required. This pays off if the number of resolutions is high as compared to insertions. A centralized resolver might be possible, but it turns out to be hard to locate servers (SPKI/SDSI recommends embedding URLs into keys to fix this) and it is unclear where derived certificates are to be stored. A DHT looks like a good match to the problem both for lookup and storage, and trust issues don't arise because the certificates are all self-signed.

ConChord was developed using Chord as the DHT. Two main problems arose. Firstly, maintaining closure (derived certificates) while supporting concurrent updates proved difficult because of the lack of atomic updates or locking. The

solution is to provide eventual consistency using periodic replays of the updates. Secondly, large datasets cause load imbalance, but traditional (CFS-style) data distribution does not support concurrent updates to those datasets. The solution here is to distribute the sets over multiple nodes but serialize updates through a single node.

A system has been built and evaluation is very promising with fast resolution, single lookups for membership checks and insertion that is quick enough. Resolvers can also use ConChord to share resolutions and thus save work. Future work will tackle replication of data, malicious clients and storage limiting.

Discussion: Q: What about revocation? A: Planned for future work. Revocation (and revalidation) lists can be stored in ConChord. SPKI specifies that proofs that contain revocable certificates must also contain the relevant CRL(s). Q: What about deletions (certificate expirations)? A: We serialize deletions through a single node, which simplifies this. Q: Is there a better solution than DHTs? A: We did need some extra code on the DHT nodes to do some application-specific stuff to provide invariants, handle expirations, etc. We'd like to soften the closure ideas as part of future work.

***Russ Cox, Athicha Muthitacharoen, Robert Morris*, "Serving DNS using Chord", presented by Russ Cox.** The idea was to rework the Domain Name Service so that it runs over a Chord, but unfortunately the result "sucked". This was unexpected because DNS was originally created to replace a distributed file called `hosts.txt`, but the new system meant that everyone had to become a DNS administrator, everyone needed a 24x7 running machine and data can now be locally correct, yet globally wrong. Peer-to-peer should address these issues by providing simple ways to distribute data to a permanently running system that has a single view of what is stored. The arrival of DNSSEC means that the data can be distributed to untrusted systems. Furthermore, DNS data is entirely "needles", so it ought to be a killer app for DHTs!

The idea was to lookup SHA-1{hostname, data type} on the Chord ring. But latency on cache misses is $O(log\ n)$ whereas DNS effectively has log base one million. Using Chord is five times slower on average. Robustness should be a plus, but DNS is already very robust and the new design introduces new vulnerabilities. Network outages also cause significant problems with no locality of storage for local names. Examining the O'Reilly "Bind and DNS" book shows thirteen common errors, but 9 are arguably bugs in BIND, 3 are at the protocol level (and reoccur here, albeit in different forms) and only 1 disappears entirely (there are no slave servers to make a mess of configuring). DNS has evolved over the last twenty years to include server-side computation; it's not just a distributed `hosts.txt` anymore. DHTs will be unable to replace distributed databases that include server-side computation, unless that can be expanded to include some form of mobile code. Questions also arise as to how much systems can be trusted to continue working without any incentives for doing this; there'd be no-one to blame if one's DNS disappeared.

Discussion: Q: So your conclusion is that DNS does not work well with DHTs? A:*(from the floor)* "I disagree that other DHTs would be as bad, Tapestry wouldn't have the same latency on lookup." "If you used Pastry then you would get less latency and a better fit." "Maybe DHTs are a good tool if they're being used for the wrong things?"

Session 5: Are we on the right track? *Chair: John Kubiatowicz*

***Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble*, "Exploring the Design Space of Distributed and Peer-to-Peer Systems: Comparing the Web, TRIAD, and Chord/CFS", presented by Stefan Saroiu.** Peer-to-peer has arrived, but will these systems stay? They do many things well but crucial "ilities" are missing: securability, composability and predictability. Because in DHTs a name is an address, this means that the name of content dictates which node it has to be placed upon, which might not be secure. Because routing is name based and servers are routers, you cannot trust routers more than servers. Because topology is dictated by keys you can surround and hijack content. Equally, it's hard to provide extra resources for "hot content" – you don't control your neighbors, but they are providing the Quality of Service. A Chord network with 20% modems has 80% slow paths. Moving forward it is necessary to enforce who publishes or participates, engineer according to specific load and value of content, and it must be possible to delegate, engineer responsibilities and isolate failures in a predictable way.

Discussion: Q: When the systems stop being overlays and start becoming infrastructure does this change things? A: Yes, that's exactly what I hope to influence. Q: For censorship resistance you may not want controllability? A: You need to think before losing that sort of property.

***Pete Keleher, Bobby Bhattacharjee, Bujor Silaghi*, "Are Virtualized Overlay Networks Too Much of a Good Thing?", presented by Bobby Bhattacharjee.** Material is published by one node on a DHT, but stored by another. The virtualization provided by the overlay network gives relatively short paths to any node and load balancing is straightforward. These are clean elegant abstractions with provable properties. However, locality of access is lost. One cannot prefetch, search nearby or do other useful things with application specific information or with names that are naturally hierarchical. The choices are to add locality back at higher levels, use a higher granularity of exports, or to get rid of the virtualization.

TerraDir is a non-virtualized overlay directory service that assumes a static rooted hierarchical namespace. It caches paths and also returns a digest of everything else at an accessed node. This allows more/less specific queries to be done within the digest without further communication. Another important design feature is that the higher something is placed in the hierarchy, the more it is replicated. The system has been simulated. It was found that higher levels did

more work, but as cache was added load balancing improved, as did the latency on requests. The system was resilient with >90% successful searches with 30% of systems failed. Current work is looking at load adaptive replication (where the main issue is consistency) and at improving searches where there is no good match between the search and the name hierarchy.

Discussion: Q: Why are there only 32K nodes in your simulation? A: We only had 24 SPARCs, though they did all have loads of RAM. We wanted the simulation to finish!

Session 6: Searching and indexing *Chair: Robert Morris*

***Adriana Iamnitchi, Matei Ripeanu, Ian Foster*, "Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations", presented by Adriana Iamnitchi** Scientific collaborations are characterized by groups of users sharing files and mainly reading them. Other group members will also wish to see any new files, so there is strong group locality and also time locality, in that the same file may be requested multiple times. It is an open problem as to whether scientific collaborations exhibit particular patterns and whether they can be exploited to create self-configuring networks that match the collaboration network characteristics. The Fermi high-energy physics collaboration was studied. It consists of 1000+ physicists at 70 institutions in 18 countries. Examining file usage it could be seen that path lengths were similar to a random network, but there was significant clustering of connections. This is the classic definition of a "small world".

A small world can be seen as a network of loosely connected clusters. The idea was to build a search system to take advantage of usage patterns. The search combines information dissemination within clusters and query routing/flooding among clusters. Gossip is used to maintain cluster membership and disseminate location info. Bloom filters are used to compress file location information. Requests are flooded to the same cluster and then to other clusters if requested information is not found locally. The system needs to adapt to the users' changing data interests, which is done by connecting nodes if they share many files in common or disconnecting them as the number of shared files drops. This ensures that the network mirrors the usage patterns. For future work it remains to be seen if there are general lessons or if this approach is domain specific.

Discussion: Q: What information is disseminated? The data itself by replicating files, or the file location? A: The data is large and dynamic as new files are inserted into the network, so copying and indexing is expensive. Q. Are there many new files? A. Yes, it is characteristic of scientific collaborations that new files are created. Q. Is there any software available or in development? A. This problem came from Grid computing where there is a lot of effort in sharing computational resources (not only files). The Globus toolkit `www.globus.org` is freely available software for creating and collaborating in Grids.

***Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, Ion Stoica*, "Complex Queries in DHT-based Peer-to-Peer Networks", presented by Ryan Huebsch.** There is a poor match between databases and peer-to-peer systems. The former have strong semantics and powerful queries, the latter, flexibility, fault-tolerance and decentralization. However, query processing is better matched to peer-to-peer systems. A keyword search corresponds to a simple "canned" SQL query and the idea is to add more types of query. For example, a "join" can be done by hashing and hoping to get the data back from the node where a local join is performed. The motivation is to provide a decentralized system, not to provide improved performance or to replace relational databases.

A system called PIER has been developed on top of CAN. It has a simple API: "publish", "lookup", "multicast" (restricted to particular namespaces rather than the whole network), "lscan" (to retrieve data from a local namespace) and when new data is added a callback mechanism is used. Current work is on the effect of particular algorithms and looking for new types of operation. Several issues arise which are common across the peer-to-peer community: caching, using replication, and security. Specific database issues are when to pre-compute intermediate results, how to handle continuous queries and alerters, choice of performance metrics, and query optimization as it relates to routing.

Discussion: Q: What's the difference between this and distributed databases? A: They haven't taken off. What is actually built is parallel databases under one administrator, whereas this is decentralized. Q: Astrolabe focuses on continuous queries and aims to have constant loading. Won't these joins have a nasty complexity in terms of communications loading? A: This is the realm of performance metrics, and yes we're concerned about these. Q: Surely users will remove the application if its resource requirements are intrusive? A: It might well be important to see what users will tolerate, we haven't looked at this yet. Q: Is this a read-only system? A: One reads the raw data, but it is also necessary to keep track of indexes and optimize them.

***David Karger* gave a talk on "Text Retrieval in Peer-to-Peer Systems".** The traditional approach to text retrieval is to observe that the majority of the information base is text. The user formulates a text query (often inaccurately), the system processes the corpus and extracts documents. The user then refines the query and the procedure iterates.

The procedure has two metrics, "recall" is the percentage of relevant documents in the corpus that are retrieved and "precision" is the percentage of retrieved documents that have relevance. There is always a trade-off between performance on these two metrics. The procedure is speeded up by pre-processing, since users notice delays of more than 0.5sec and give up after 10 seconds. Searching the web has non-traditional aspects. People care more about precision than about recall, and they are prepared to wait rather longer for results.

Boolean keyword searches are done by using inverted indexes and performing intersections and joins. They can also be done by direct list merging, which

gives a complexity of $O(size\ of\ list\ of\ smallest\ term)$. Problems arise with "synonymy" (several words for the same thing – fixed by using a thesaurus to increase recall but lower precision), and "polysemy" (one word means several things, increasing precision but lowering recall – unfortunately asking for the user's assistance may just confuse).

Further issues are "harsh cutoffs" (relevant documents are missed because they don't contain all the required keywords – fixed by quorum systems such as Altavista's) and "uniform influence" (no allowance in the scoring for multiple usage of a word, and rare terms have no special influence). These latter problems can be addressed by vector space models where each co-ordinate has a value expressing occurrence and the "dot product" measures similarity. These systems inherently provide rankings of results and are easy to expand with synonyms. In some systems, if the user indicates that a particular document located by a first search is relevant then a refined search can be done with hundreds or thousands of terms taken from that document.

Google has a corpus of about three billion pages, average size 10K (i.e. 30TB of data). Their inverted index is of similar size. They use a boolean vector space model. Experience has been that most queries are only two terms and queries have a Zipf distribution so caching helps a bit. Their scoring scheme also looks at links, raising the relevance of highly linked pages and observing that the anchor text in the link may be a better description of the contents than the actual page. Google uses a server farm of several thousand machines. In principle a moderate sized peer-to-peer system of perhaps ∼30,000 nodes should be able to perform the same function...

An obvious design would be to partition the documents between the nodes and each node then builds the inverted index for its documents. The results then need to be merged, which creates an n^2 connectivity problem. Alternatively, a DHT could be used to partition the terms and a query is done by talking to appropriate nodes. The drawback is that the lists could be very long which creates a bandwidth problem as the lists are merged. An improvement would be to create an inverted index on term pairs (i.e. pre-answer all two term queries), however this would generate $n^2/2$ pairs and for a Google sized database would create ∼15,000TB of index. However, the idea does work well for some special cases where n is small (song titles) or where a windowing approach can be used (relevant documents will tend to have the terms near to each other).

However, the economics favor a centralized approach and Google already uses a type of peer-to-peer design in their data center where they can exploit high inter-machine bandwidths. Their main bottleneck is content providers restricting their rate of "crawl" across the web, which can make their index old and out-of-date. Distributing the crawler function will not assist, but making the content providers into peers might assist, assuming their input could be trusted.

Distributed systems do differ from "big iron" in that it is possible to gain a feeling of privacy and to provide some anonymity from the partition of control and knowledge. Expertise networks such as MIT's Haystack or HP's Shock try to route questions to experts and the challenge is to index the experts' abilities.

Session 7: Security in peer-to-peer systems *Chair: Steve Gribble*

***Emil Sit, Robert Morris*, "Security Considerations for Peer-to-Peer Distributed Hash Tables", presented by Emil Sit.** In a distributed peer-to-peer system there is no "Trusted Computing Base". Peers can only trust themselves and the protocols need to be designed to allow peers to verify the correct operation of others. A reasonable adversary model is that any node can be malicious, discarding traffic or sending malicious packets. Malicious nodes can collude with each other, but they do not have the ability to intercept and read traffic for arbitrary users.

One idea is to develop system invariants and verify them. For example, in Chord you expect to halve the distance to your goal on each step, so this should be verified. If recursive queries were allowed then this might improve latency but a malicious node could cause incorrect results or cause traffic to loop forever. If progress is being monitored then bad behavior can be detected. If one wishes to know whether a node is the correct endpoint – and hence its answer cannot be bettered – then the rule assigning keys to nodes needs to be verifiable so that one can check that the value hashes correctly to the node and that the node is placed correctly according to some relatively hard to forge value such as an IP address. Malicious nodes may conspire to suck nodes or queries into an "evil ring" which does not contain all the participants in the system. New nodes being added into the DHT needs to cross-check the answers with other nodes to ensure they are consistent. Independent routes to resources (such as in CAN) can be utilized for checking. Another important problem is that malicious nodes may fail to create the replicas that they should and this may not be detected if a single node is responsible for this action.

General design principles for security are: #1 define verifiable system invariants, and verify them; #2 allow the querier to observe lookup progress; #3 assign keys to nodes in a verifiable way; #4 be wary of server selection in routing; #5 cross-check routing tables using random queries; and #6 avoid single points of responsibility.

Discussion: Q: What do you do when you detect an error? A: You may be able to route in another way. Q: Can you tell what the spacing in a Chord ring should be? A: Yes, you can look at the number of participants or just look at the spacing of your own successors.

***John R. Douceur*, "The Sybil Attack", presented by John R. Douceur.** In large distributed systems you can make assumptions about what percentage of the participants are conspiring against you. However, it is hard to substantiate these assumptions and in fact, all other participants may be under a single opponent's control. The "Sybil Attack" (named after a 1973 book on multiple personalities) is an attempt to break down the determination of identity distinctness, where an identity is an abstract notion connected in a provable way to a persistent item such as a public key.

An obvious source of identity information is a trustworthy authority such as Verisign or, more subtly when using IP addresses or DNS, ICANN. Another source of information would be yourself, in that you can test if two remote systems can do something that they couldn't manage if they were a single entity. You might also be able to get others to assist in this testing and accept identities that others have endorsed. The type of tests that could be attempted would be a communications resource challenge (can they handle large volumes of traffic), a storage resource challenge (can they store lots of data) or a computational resource challenge (can they do sums very fast). There are problems with such tests – the challenges must require simultaneous solving by all the identities (because otherwise the opponent can solve one problem at a time) and of course the opponent may command more resources than a standard node anyway. You might hope to leverage something from other people's view of identities, but of course the identities who are vouching for each other may all be under the opponents control. The paper formalizes all of this, but the conclusion is that distinctness can only be verified by means of a certificating authority or by measurement of some, yet to be discovered, uniform constraint. At the moment identity verification does not scale.

Discussion: Q: You can't really fix this at the system level. In World War I the entire German spy network in the UK was run by the British. In World War II the Germans faked the SOE network in The Netherlands. You won't be able to tell if the rest of the network isn't the 50,000 people at the NSA. A: Worse than that, it could be just one person. Q: Where did you get all the marvelous animated graphics you've used? A: Standard with PowerPoint 2002.

***Jared Saia, Amos Fiat, Steve Gribble, Anna Karlin, Stefan Saroiu*, "Dynamically Fault-Tolerant Content Addressable Networks", presented by Jared Saia.** Napster was shut down by legal attacks on a central server and research shows that Gnutella would shatter if a relative handful of peers were removed. It is easy to shut down a single machine, which has limited bandwidth or a limited number of lawyers. However, it is harder to remove large numbers of machines. The Deletion Resistant Network (DRN) is a scalable, distributed, peer-to-peer system that after the removal of $\frac{2}{3}$ of the peers by an omniscient adversary who can choose which to destroy, 99% of the rest can access 99% of the remaining data. However, DRN is only robust against a static attack. If all the original peers are removed, then the system fails even if many new peers have joined. The Dynamic DRN has stronger properties against the same adversary, and for a fixed n and $\epsilon > 0$, if there are $O(n)$ data items in the network then in any period where $(1 - 2\epsilon)n$ peers are deleted and n join, then with high probability all but the fraction ϵ of the live peers can access a $1 - \epsilon$ fraction of the content.

These networks are based on butterfly networks (a constant degree version of a hypercube). The network properties can be proved probabilistically using expander graphs. Peer join time and search time require $O(log\ n)$ messages, but in the whole system $O(log^3 n)$ storage and $O(log^3 n)$ messages are needed, ie: the

network has some very desirable robustness properties, but the time and space bounds remain comparable with other systems.

Discussion: Q: What happens when the system becomes so big that you need a new level of butterfly linkage? A: We don't yet have any good ideas how to deal with large changes in size. Best we can do is $O(n^2)$ messages and $O(n)$ broadcasts. Q: This structure protects the network, but what about the data items? A: We're not protecting any particular data item against attacks directed specifically at it, but they can be duplicated.

Session 8: Applications II *Chair: Ion Stoica*

***Sridhar Srinivasan, Ellen Zegura*, "Network Measurement as a Cooperative Enterprise", presented by Sridhar Srinivasan.** Network measurement is undertaken to improve performance and to assess the utilization of resources. The challenge is to deploy an Internet-wide service, keep the overhead low (to avoid perturbing the network) and to be assured about the accuracy of reported values. M-Coop is a peer-to-peer measurement architecture. Each peer has an area of responsibility (AOR) which it reports upon, ideally at least as small as an AS. An overlay network is constructed, with peers selecting neighbors in adjacent ASs. A second overlay network is built within the AS, the nodes of which peer entirely within the AS and have AORs of parts of the AS. When measurement queries are made they are passed over the overlay network and it is the data collected from the experiences of these packets which is reported back as the metric. Data is also given a "trust" component, which is a measure of past reliability. This is done by comparing results with other measurements of the link to a neighbor as well as "pings" of nearby machines.

Future challenges are to improve the composition of measurements (the path through the overlay may not be the same as the general experience for other traffic); to deal with colluding malicious nodes; to verify all measurements; to assess what level of participation is needed to get a good answer to queries; and to determine how useful the information will be in practice.

Discussion: Q: Latency is easy to verify but bandwidth isn't? A: Exactly so.

***Venkata N. Padmanabhan, Kunwadee Sripanidkulchai*, "The Case for Cooperative Networking", presented by Kunwadee Sripanidkulchai.** CoopNet is a peer-to-peer system that is intended to complement and work in conjunction with existing client-server architectures. Even when making minimal assumptions about peer participation it provides a way of dealing with "flash crowds" on web sites such as occurred on news sites on 9/11 or the notorious Slashdot effect. The bottleneck in such cases is not disk transfer, since everyone is fetching the same "hot" content. CPU cycles are an issue, though changing from dynamic to static content fixes this. The main problem is bandwidth, so the idea is to serve the content from co-operating peers, with clients being redirected to an appropriate peer – the redirection being a relatively small (1%) bandwidth imposition. The co-operating peers announce themselves as they fetch

the content by means of an HTTP pragma, and the server can then record their identities in case their assistance is needed. Returning appropriate alternative sites is a complex problem. Using BGP prefix clusters is lightweight but crude and various better schemes are under investigation. Simulations using the traces of the MSNBC website on 9/11 show that good results can be obtained with just 200 co-operating peers. The peers were busy just 20% of the time and the bandwidth they had to donate was low. Unfortunately, there were long tails on some of the resource usage distributions and further work is needed to minimize these.

Discussion: Q: How does this compare with DHTs? A: This is push technology oriented. Q: When do you decide to start redirecting? A: Preferably just as you start to be overloaded Q: How does this compare with systems such as Scribe? A: Those systems require you to commit what you belong to ahead of time, rather than dynamically. Q: Aren't there concerns with privacy here? A: This is only for popular documents and the server is always in control of when it redirects.

***Ion Stoica, Dan Adkins, Sylvia Ratnasamy, Scott Shenker, Sonesh Surana, Shelley Zhuang*, "Internet Indirection Infrastructure", presented by Ion Stoica.** Today's Internet has a point-to-point communications abstraction. It doesn't work well for multicast, anycast or mobility. Existing solutions to these problems change IP (into mobile IP or IP multicast). These solutions are hard to implement whilst maintaining scalability, they do not interoperate or compose, and people may not be incentivized to provide them. The result has been provision of facilities at the application layer (such as Narada, Overcast, Scattercast...) but efficiency is hard to achieve. It should be noted that all previous schemes have used indirection, so perhaps there should be an indirection layer as an overlay network placed over the IP layer.

The service model is "best efforts" and data is exchanged by name. To receive a packet a trigger is maintained in the overlay network by the end-point that owns it. This trigger will know how to reach the end point. This scheme is capable of supporting many different types of services. Mobility is simple, the end-point tells the trigger where it has moved to. In multicast many hosts all insert the same named trigger. For anycast there is an exact match on the first part of a name and a longest prefix match on the last part. Composable services can be done by having a stack of triggers and sending packets off to each service in turn (e.g. sending data via an HTML→WML converter before delivering it to a wireless device). These stacks can be built by both sender and receiver without the other needing to be aware of the processing. Load balancing or location proximity can be expressed by putting semantics into red tape bits in the trigger. An early prototype has been implemented based on Chord. Each trigger is stored on a server and the DHT is used to find the best matching trigger. The results can be cached and further packets sent to the server directly.

Discussion: Q: What about authorizations? A: You can have public and private identifiers, and this leads to the idea of changing from a public to a

private trigger once authorization has been given. Q: Does this system defend against Denial of Service attacks? A: At the old layer, yes, but there are new possibilities at the overlay layer such as creating circular routes. Q: Perhaps you are putting too much into this new layer? A: Many problems can be solved in a new way by using this indirection layer. For example you can now seriously consider providing a reliable multicast.

***Tyron Stading, Petros Maniatis, Mary Baker*, "Peer-to-Peer Caching Schemes to Address Flash Crowds", presented by Tyron Stading.** Non-commercial sites do not usually expect flash crowds and do not have the resources to pay for commercial systems to mitigate their effects. "Backslash" is a peer-to-peer collaborative web mirroring system suitable for use by collectives of websites to provide protection against unusually high traffic loads. The idea is to create a load balancing system that will usually direct requests to the main site for page content. If it perceives that an overload is about to occur, the pages are rewritten into the load-balancing collective and further requests are served from there until the overload condition finishes.

The collective is based on a DHT (currently CAN) and the data (currently assumed to be static) is distributed to members of the collective using cache diffusion techniques. The idea is that popular content will be pushed out to more nodes. This can produce a "bubble effect" where the inner nodes of the collective become idle and the outer nodes do all the serving. To prevent this, some probabilistic forwarding of requests is done to try and use all of the collective nodes. The system has been simulated handling two flash crowds at once and it worked well in balancing the load. However, with too much diffusion agility the second flash crowd caused competition for entries and performance suffered. The probabilistic forwarding also worked less well than expected. Future work will simulate the system at a higher fidelity and will look at the effect of cache invalidation for changing content.

Discussion: Q: Who is going to use this service? What are the incentives? A: We concentrated on non-profits because they are more likely to work to help each other. Q: Do you need to copy ahead of time? A: Yes. If you don't copy the files before the flash crowd gets huge then there's no guarantee the files will be available.

Session 9: Data Management *Chair: David Karger*

***Robbert Van Renesse, Kenneth Birman*, "Scalable Management and Data Mining Using Astrolabe", presented by Robbert Van Renesse.** Astrolabe takes snapshots of the global state of a system and distributes summaries to its clients. Its hierarchy gives it scalability, the use of mobile SQL gives it flexibility, robustness is achieved by using epidemic (Gossip) protocols and security comes from its use of certificates.

Astrolabe has a DNS-like domain hierarchy with domain names identified by path names within the hierarchy. Each domain has an attribute list called a MIB

which identifies a domain, lists its Gossip contacts, lists server addresses for data access and indicates how many local hosts there are. These MIBs can be aggregated into parent domains. There is a simple API of `Get_MIB(`*domain_name*`)`, `Get_Children(`*domain_name*`)` and `Set_Attr(`*domain_name, attribute, value*`)`. All hosts hold their own MIB and also the MIBs of sibling domains, giving a storage requirement of $O(log\ n)$ per host. The MIBs are extensible with further information as required. Standard SQL queries are gossiped to summarize data into parents. This allows aggregation queries to be made such as "where is the highest loaded host" or "which domains have subscribers who are interested in this topic" or "have all hosts received the latest software update".

The system works by passing messages using a simple epidemic protocol that uses randomized communications between nearby hosts. This is fast (latency grows $O(log\ n)$ with probabilistic guarantees on maxima, assuming trees with constant branching factors) and is robust even in the face of denial-of-service attacks. Failure is detected when timestamps in MIB copies do not update and new domains are found by gossiping, occasional broadcasts and by configuration files.

Discussion: Q: Would an overlay network with a general tree structure be better suited? A: We're doing aggregation, and DHTs don't help with that. We started by trying to build a scalable multicast system and we needed ways to get past firewalls etc. The routing is all proximity based and every aspect of the system reflects locality. Q: What is being gained by the massive replication of data? A: Results are available in a single step. Q: Aren't some queries expensive? A: We restrict join queries to make the load manageable.

***Nancy Lynch, Dahlia Malkhi, David Ratajczak*, "Atomic Data Access in Content Addressable Networks", presented by David Ratajczak.** This work shows that atomicity is useful and achievable in a peer-to-peer DHTs with practical fault-tolerance. Atomicity can be seen as consistency with the effect of having a single copy of a piece of data being accessed serially. It is a crucial property where there are multiple writers and can be used as a building block for many distributed primitive operations. The other important property that is demonstrated is "liveness", viz that any submission to an active node is guaranteed a response. However, one cannot achieve atomicity or liveness in an asynchronous failure-prone system and of course peer-to-peer systems are dynamic and an unfortunate sequence of events can disconnect the network. The asynchronous property means that you need to use timeouts to detect failures and this can cause problems if the original request is still active. Therefore, it is assumed that communication links are reliable FIFO channels and that node failures do not occur. This can be done by local fault-tolerant, redundant, hardware (a "replica group") which turns failures into graceful disengagement from the network.

The API for the system is `Join()`, `Update()` and `Leave()`, with the Leave function now being a non-trivial operation. The guarantees provided include atomicity and liveness. The algorithms are in the paper and repay careful study.

Future work will determine if consecutive nodes in the DHT ring can form a replica group, with a multi-level structure around the ring. It looks as if setting thresholds for splitting large replica groups and merging small ones can be set in the $O(log\ n)$ region – and then some good properties will result. At present the algorithm presented does not fail especially gracefully and more details of the fault-tolerance properties need to be worked out. The system will need to be built before it can be fully understood.

Discussion: Q: Isn't it hard to leave whilst you're receiving constant updates? A: In practice, all the operations in one node are in one thread. You can prove that you will eventually make progress and leave (or indeed join) the network.

***Yan Chen, Randy Katz, John Kubiatowicz*, "Dynamic Replica Placement for Scalable Content Delivery", presented by Yan Chen.** Content Distribution Networks (CDNs) attempt to position data so as to have useful things in local "replicas", so as to improve the user experience of the web and of streaming media, whilst minimizing resource consumption. The problem is how to choose replica locations and then keep them up-to-date, viz: an adaptive cache coherence system is required.

Previous work concentrated on static replica placement, assuming that the clients' locations and access patterns were known in advance. Data transfer cannot be done by IP multicast because this is impractical from one domain to another and application layer multicast (ALM) fails to scale. The usual solution is to replicate the root of the ALM system, but this suffers from consistency problems and communication overhead. This work uses a peer-to-peer overlay location service for scalability and locality of search. Tapestry is used because it already has some locality within it.

The system simultaneously creates a tree for disseminating data and decides upon replica placement. The idea is to search for qualified local replicas first and then place new replicas on the Tapestry overlay path. Two algorithms were investigated. A naïve scheme allows a node that is holding the data to decide whether it is a suitable parent to hold the data given the identity of the requesting client and if so it then puts the replica as close to the client as possible. The smart scheme is prepared to consider its parent, siblings and server children as possible parents as well and then chooses the node with the lightest load. It then places the replica a long way from the client. This latter scheme has a higher overhead in messages, but results in better placement, with fewer replicas required. It performs almost as well as in the ideal case where all requests are known in advance. Future work will evaluate the system with diverse topologies and real workloads. Dynamic deletion and insertion will be considered so that the system can adapt as user interests change, and the system will be integrated into the OceanStore project.

Discussion: Q: How important is Tapestry to this system? A: Tapestry is providing proximity properties, identifying potential replica placement points, and improving the scalability of the CDN for update dissemination so that each node only has to maintain states for its parent and direct children.

Thanks

The workshop ended with thanks being expressed to the organizers and in particular to Frans Kaashoek, who in turn expressed his gratitude to Neena Lyall for all her work on the logistics of running the workshop and Microsoft Research for their generous financial support.

Acknowledgements

My attendance at the workshop was due to the financial assistance of the Cambridge MIT Institute.

Thanks to George Danezis for comparing this account to his own detailed notes of the event and pointing out everything I'd forgotten to include. Thanks also to the other workshop participants for their corrections to my original inaccurate reporting of their contributions.