

Kelips*: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead

Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, Robbert van Renesse[†]
Cornell University, Ithaca, NY, USA
{gupta, ken, linga, ademers, rvr}@cs.cornell.edu

Abstract

A peer-to-peer (p2p) distributed hash table (DHT) system allows hosts to join and fail silently (or leave), as well as to insert and retrieve files (objects). This paper explores a new point in design space in which increased memory usage and constant background communication overheads are tolerated to reduce file lookup times and increase stability to failures and churn. Our system, called *Kelips*, uses peer-to-peer gossip to partially replicate file index information. In *Kelips*, (a) under normal conditions, file lookups are resolved with $O(1)$ time and complexity (i.e., independent of system size), and (b) membership changes (e.g., even when a large number of nodes fail) are detected and disseminated to the system quickly. Per-node memory requirements are small in medium-sized systems. When there are failures, lookup success is ensured through query rerouting. *Kelips* achieves load balancing comparable to existing systems. Locality is supported by using topologically aware gossip mechanisms. Initial results of an ongoing experimental study are also discussed.

*System name derived from *kelip-kelip*, Malay name for the self-synchronizing fireflies that accumulate after dusk on branches of mangrove trees in Selangor, Malaysia [11]. Our system organizes similarly into affinity groups, and nodes in a group “synchronize” loosely to store information for a common set of file indices.

[†]The authors were supported in part by DARPA/AFRL-IFGA grant F30602-99-1-0532 and in part by a MURI grant AFOSR F49620-02-1-0233, with additional support from the AFRL-IFGA Information Assurance Institute, from Microsoft Research and from the Intel Corporation.

1 Introduction

A peer-to-peer (p2p) distributed hash table (DHT) implements operations allowing hosts or processes (nodes) to join the system, and fail silently (or leave the system), as well as to insert and retrieve files with known names. Many DHTs have been deployed (e.g. Fasttrack-based systems such as Kazaa) while several others are a focus of academic research, e.g., Chord [3], Pastry [6], Tapestry, etc. [8].

All p2p systems make tradeoffs between the amount of storage overhead at each node, the communication costs incurred while running, and the costs of file retrieval. With the exception of Gnutella, the work just cited has focused on a design point in which storage costs are logarithmic in system size and hence small, and lookup costs are also logarithmic (unless cache hits shortcut the search). However, a study [9] on file sharing systems such as Gnutella and Napster has shown that a significant fraction of nodes could be connected over high latency / low bandwidth links. The presence of even one such slow logical hop on a logarithmically long path is thus likely. This increases the overall cost of the lookup.

We argue that this can be avoided by exploring other potentially interesting points in the design of p2p DHTs. One could vary the soft state memory usage and background network communication overhead at a node in order to realize $O(1)$ lookup costs. For example, complete replication of soft state achieves this, but this approach has prohibitive memory and bandwidth requirements.

The *Kelips* system uses $O(\sqrt{n})$ space per node, where n is the number of nodes in the system. This soft state suffices to resolve lookups with $O(1)$ time

and message complexity. Continuous background communication with a constant overhead is used to maintain the index structure with high quality, as well as guarantee quick convergence after membership changes. The \sqrt{n} design point is of interest because, within Kelips, both the storage overhead associated with the membership data structure and that associated with replication of file-index (henceforth called *filetuple*) data impose the same $O(\sqrt{n})$ asymptotic cost. Kelips uses query rerouting to ensure lookup success in spite of failures. The mechanism also allows us to use round trip time estimates to select nearby peers for each node.

Memory usage is small for systems with moderate sizes - if 10 million files are inserted into a 100,000-node system, Kelips uses only 1.93 MB of memory at each node. The system exhibits stability in the face of node failures and packet losses, and hence would be expected to ride out “churn” arising in wide-area settings from rapid arrival and failure of nodes. This resilience is achieved through the use of a lightweight Epidemic multicast protocol for replication of system membership data and file indexing data [1, 4]. We note that whereas many DHT systems treat file replication as well as lookup, our work focuses only on the lookup problem, leaving replication to the application. For reasons of brevity, this paper also omits any discussion of privacy and security considerations.

2 Core Design

Kelips consists of k virtual *affinity groups*, numbered 0 through $(k - 1)$. Each node lies in an affinity group determined by using a consistent hashing function to map the node’s identifier (IP address and port number) into the integer interval $[0, k - 1]$. Let n be the number of nodes currently in the system. The use of a cryptographic hash function such as SHA-1 ensures that with high probability, the number of nodes in each affinity group is around $\frac{n}{k}$.

Node soft state consists of the following entries:

- **Affinity Group View:** A (partial) set of other nodes lying in the same affinity group. Each entry carries additional fields such as round-trip time estimate, heartbeat count, etc. for the other node.

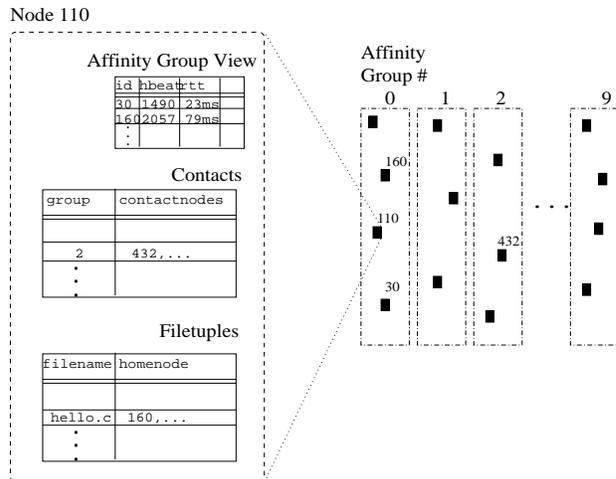


Figure 1: **Soft State at a Node:** A Kelips system with nodes distributed across 10 affinity groups, and soft state at a hypothetical node.

- **Contacts:** For each of the other affinity groups in the system, a small (constant-sized) set of nodes lying in the foreign affinity group. Entries contain the same additional fields as in the affinity group view.
- **Filetuples:** A (partial) set of tuples, each detailing a file name and host IP address of the node storing the file (called the file’s *homenode*). A node stores a filetuple only if the file’s *homenode* lies in this node’s affinity group. Filetuples are also associated with heartbeat counts.

Figure 1 illustrates an example. Entries are stored in AVL trees to support efficient operations. **Memory Usage at a node:** The total storage requirements for a Kelips node are $S(k, n) = \frac{n}{k} + c \times (k - 1) + \frac{F}{k}$ entries (c is the number of contacts per foreign affinity group and F the total number of files present in the system). For fixed n , $S(k, n)$ is minimized at $k = \sqrt{\frac{n+F}{c}}$. Assuming the total number of files is proportional to n , and that c is fixed, the optimal k then varies as $O(\sqrt{n})$. The minimum $S(k, n)$ varies as $O(\sqrt{n})$. This is asymptotically larger than Chord or Pastry, but turns out to be reasonably small for most medium-sized p2p systems.

Consider a system with $n = 100,000$ nodes over $k = \lceil \sqrt{n} \rceil = 317$ affinity groups. Our current implementation uses 60 B filetuple entries and 40 B membership entries, and maintains 2 contacts per foreign affinity group. Inserting a total of 10

million files into the system thus entails 1.93 MB of node soft state. With such memory requirements, file lookup queries return the location of the file within $O(1)$ time and message complexity, i.e., these costs are invariant with system size n .

2.1 Background Overhead

Existing view, contact and filetuple entries are refreshed periodically within and across groups. This occurs through a heartbeating mechanism. Each view, contact or filetuple entry stored at a node is associated with an integer heartbeat count. If the heartbeat count for an entry is not updated over a pre-specified time-out period, the entry is deleted. Heartbeat updates originate at the responsible node (for filetuples, this is the homenode) and are disseminated through a peer-to-peer epidemic-style (or gossip-style) protocol [7]. This gossip communication constitutes the background communication within a group. This continuous *gossip stream* is also used to disseminate new view, contact and filetuple entries to the system.

We first outline gossip-style dissemination within one affinity group. A piece of information (e.g., a heartbeat update for a filetuple) is multicasted to the group by using a constant background bandwidth, incurring latency that increases polylogarithmically with group size. Once a node receives the piece of information to be multicast (either from some other node or from the application), the node gossips about this information for a number of *rounds*, where a round is a fixed local time interval at the node. During each round, the node selects a small constant-sized set of target nodes from the group membership, and sends each of these nodes a copy of the information. Gossiping thus uses constant bandwidth. With high probability, the protocol transmits the multicast to all nodes. The latency can be shown to vary with the logarithm of affinity group size. Gossip messages are transmitted via a lightweight unreliable protocol such as UDP. Gossip target nodes are selected through a weighted scheme based on round-trip time estimates, preferring nodes that are topologically closer in the network. Kelips uses the spatially weighted gossip proposed in [5] towards this. A node with round-trip time estimate r_{tt} is selected as gossip target with probability proportional to

$\frac{1}{r_{tt}^r}$. As suggested in [5], we use a value of $r = 2$, where the latency is polylogarithmic ($O(\log^2(n))$).

Analysis and experimental studies have revealed that epidemic style dissemination protocols are robust to network packet losses, as well as to transient and permanent node failures. They maintain stable multicast throughput to the affinity group even in the presence of such failures. See references [1, 2, 4].

Information such as heartbeats also need to propagate across affinity groups (e.g., to keep contact entries for this affinity group from expiring). This is achieved by selecting a few of the contacts as gossip targets in each gossip round. Such cross-group dissemination implies a two-level gossiping scheme similar to [7]. With uniform cross-group target selection, latency is more than that of single group gossip by a multiplicative factor of $O(\log(k))$ (with $k = \sqrt{n}$ affinity groups, this is the same as $O(\log(n))$).

Gossip messages in Kelips carry not just a single entry, but several filetuple and membership entries. This includes entries that are new, were recently deleted, or have an updated heartbeat. Since Kelips limits bandwidth use at each node, not all the soft state can be packed into a gossip message. Maximum rations are imposed on the numbers of view entries, contact entries and filetuple entries that a gossip message may contain. For each entry type, the ration subdivides equally for fresh entries (ones that have so far been included in fewer than a threshold number of gossip messages sent out from this node) and for older entries. Entries are chosen uniformly at random, and unused rations (e.g., due to fewer fresh entries) are filled with older entries.

Ration sizes do not vary with n . With $k = \sqrt{n}$, this increases dissemination latencies a factor of $O(\sqrt{n})$ above that of the Epidemic protocol (since soft state is $O(\sqrt{n})$). Heartbeat timeouts thus need to vary as $O(\sqrt{n} \times \log^2(n))$ for view and filetuple entries, and $O(\sqrt{n} \times \log^3(n))$ for contact entries.

These numbers are thus the convergence times for the system after membership changes. Such low convergence times are achieved through only the gossip messages sent and received at a node. This gossip stream imposes a constant per-node background overhead. The gossip stream is able to disseminate heartbeats and new entries despite node and packet delivery failures.

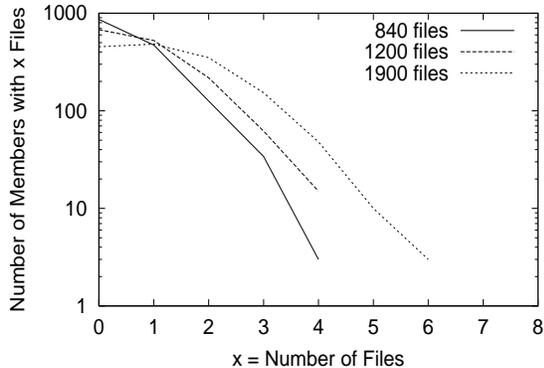


Figure 2: **Load Balancing I:** Number of nodes (*y-axis*) storing given number of files (*x-axis*), in a Kelips system with 1500 nodes (38 affinity groups).

2.2 File Lookup and Insertion

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

Insertion: A node (*origin node*) that wants to insert a given file f , maps the file name to the appropriate affinity group, and sends an insert request to the topologically closest known contact for that affinity group. This contact picks a node h from its affinity group, uniformly at random, and forwards the insert request to it. The node h is now the *homenode* of the file. The file is transferred from the origin node to the homenode. A new file tuple is created to map the file f to homenode h , and is inserted into the gossip stream. Thus, file tuple insertion also occurs in $O(1)$ time and with $O(1)$ message complexity. The origin node periodically refreshes the file tuple entry at homenode h in order to keep it from expiring.

Clearly, factors such as empty contact sets or incomplete file tuple replication might cause such

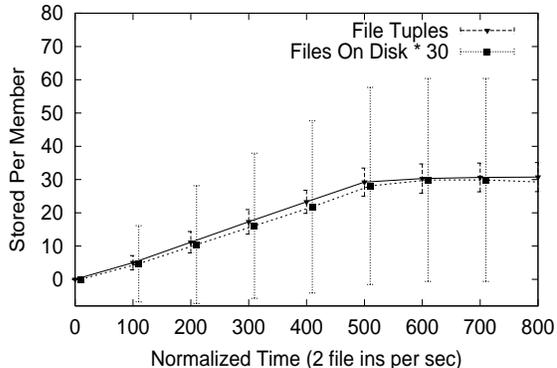


Figure 3: **Load Balancing II:** Files are inserted into a 1000 node system (30 affinity groups), 2 insertions per sec between $t=0$ and $t=500$. Plot shows variation, over time, of number of files and file tuples at a node (average and one standard deviation).

one-hop lookup or insertion to fail. Biased partial membership information might cause uneven load balancing. This is addressed by the general multi-hop multi-try query routing scheme of Section 3.

3 Auxiliary Protocols and Algorithms

We outline Kelips' protocols for node arrival, membership and contact maintenance, topological considerations and multi-hop query routing.

Joining protocol: Like in several existing p2p systems, a node joins the Kelips system by contacting a well-known introducer node (or group), e.g., a well-known http URL could be used. The joiner view returned by the introducer is used by the new node to warm up its soft state and allow it to start gossiping and populating its view, contact and file tuple set. The gossip stream spreads news about the new node quickly throughout the system.

Spatial Considerations: Each node periodically pings a small set of other nodes it knows about. Response times are included in round-trip time estimates used in spatial gossip.

Contact maintenance: The maximum number of contacts is fixed, yet the gossip stream supplies potential contacts continuously. *Contact replacement* policy can affect lookup/insert performance and system partitionability. It could be either proactive or reactive, and takes into account factors such as node distance, accessibility (e.g., firewalls

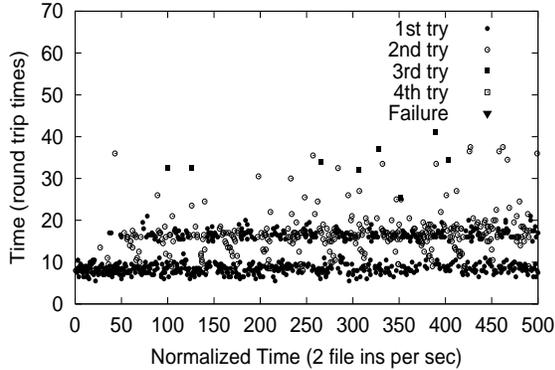


Figure 4: **File Insertion:** Turnaround times (in round-trip time units) for file insertion in a 1000-node Kelips system (30 affinity groups).

in between), etc. Currently, we use a proactive policy that chooses the farthest contact as victim for replacement.

Multi-hop Query routing: When a file lookup or insert query fails, the querying node retries the query. Query (re-) tries may occur along several axes: a) the querying node could ask multiple contacts, b) contacts could be asked to forward the query within their affinity group (up to a specified TTL), c) the querying node could request the query to be executed at another node in its own affinity group (if this is different from the file’s affinity group). Query routing occurs as a random walk within the file affinity group in (b), and within the querying node’s affinity group in (c). TTL values on multi-hop routed queries and the maximum numbers of tries trade off between lookup query success rate and maximum processing time. The normal case lookup processing time and message complexity stay $O(1)$.

File insertion occurs through a similar multi-hop multi-try scheme, except the file is inserted exactly at the node where the TTL expires. This helps achieve good load balancing, although it increases the normal case insertion time to grow as $O(\log(\sqrt{n}))$. However, this is competitive with existing systems.

4 Experimental Results

We are evaluating a C WinAPI prototype implementation of Kelips. This section reveals preliminary numbers from trace-based experiments, most done along similar lines as previous work [3, 6]. Multiple nodes were run on a single host (1 GHz

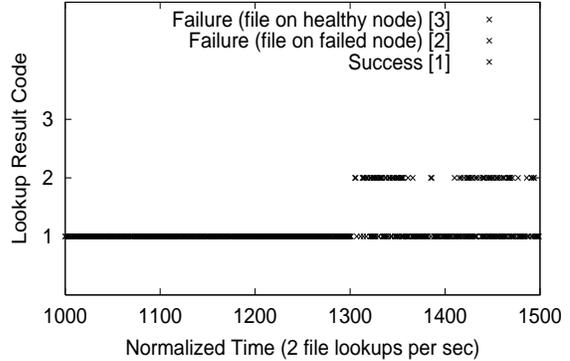


Figure 5: **Fault-tolerance of Lookups I:** In a 1000 node (30 affinity groups) system, lookups are generated 2 per sec. At time $t = 1300$, 500 nodes are selected at random and caused to fail. This plot shows for each lookup if it was successful [y -axis = 1], or if it failed because the homenode failed [y -axis = 2], or if it failed in spite of the homenode being alive [y -axis = 3].

CPU, 1GB RAM, Win2K) with an emulated network topology layer. Unfortunately, limitations on resources and memory requirements restrict currently simulated system sizes to a few thousand.

Background overhead in the current configuration consists of each node gossiping once every 2 (normalized) seconds. Rations limit gossip message size to 272 B. 6 gossip targets are chosen, 3 of them among contacts.

Load Balancing: Files are inserted into a stable Kelips system. The file name distribution used is a set of anonymized web URLs obtained from the Berkeley Home IP traces at [10]. The load balancing characteristics are better than exponential (Figure 2). File and filetuple distribution as files are inserted (2 insertions per normalized second of time) is shown in Figure 3; the plot shows that filetuple distribution has small deviation around the mean.

File Insertion: This occurs through a multi-try (4 tries) and multi-hop scheme (TTL set to $3 * \log N$ logical hops). Figure 4 shows the turnaround times for insertion of 1000 different files. 66.2% complete in 1 try, 33% take 2 tries, and 0.8% take 3 tries. None fail or require more than 3 tries. Views were found to be well replicated in this instance. In a different experiment with 1500 nodes and views only 55.8% of the maximum size, 47.2% inserts required 1 try, 47.04% required 2 tries, 3.76% required 3 tries, 0.96% needed 4 tries, and 1.04% failed. Multi-hop routing thus provides resilience

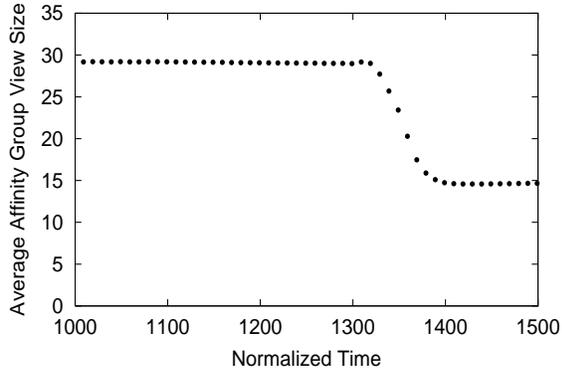


Figure 6: **Fault-tolerance of Lookups II:** At time $t=1300$, 500 out of 1000 nodes in a 30 affinity group system fail. This plot shows that failure detection and view (and hence filetuple) stabilization occurs by time $t=1380$.

to incomplete replication of soft state.

Fault-tolerance: P2P DHTs are required to be tolerant to dynamic conditions and high churn rates. We measure the resilience of Kelips to a scenario where half of the nodes in the system are caused to fail simultaneously. Figures 5 and 6 show the effectiveness of using background gossip communication. Lookups were initiated at a constant rate and were found to fail only if the homenode had also failed (Figure 5). In other words, multi-hop rerouting and redundant membership information ensures successful lookups despite failures. Responsiveness to failures is good, and membership and filetuple entry information stabilize quickly after a membership change (Figure 6).

5 Conclusion

We are investigating a new design point for DHT systems, based on increased memory usage (for replication of filetuple and membership information), as well as a constant and low background overhead at a node, in order to enable $O(1)$ file lookup operations and ensure stability despite high failure and churn rates. Per-node memory requirements are small in medium-sized systems (less than 2 MB with 10 million files in a 100,000 node system). Multi-hop (and multi-try) query routing enables file lookup and insertion to succeed even when bandwidth limitations or network disconnectivity lead to only partial replication of soft state. We observe satisfactory load balancing.

Kelips and other DHTs: Memory usage can be traded off for faster lookup times in systems like

Chord, Pastry, Tapestry, e.g., by varying the value of the base (the parameter d in Pastry, base value of 2 in Chord) that determines the branching factor of the overlay structure. This would however make routing table entries large and lead to high network traffic to keep them updated as nodes join, leave and fail. Kelips is loosely structured, and it does not need to maintain a structure and invariants (e.g., the ring, routing table entries, etc.) – the soft state allows object lookups and insertions to succeed in spite of stale membership or contact entries.

References

- [1] N.T.J. Bailey, “Epidemic Theory of Infectious Diseases and its Applications”, Hafner Press, Second Edition, 1975.
- [2] K.P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, Y. Minsky, “Bimodal Multicast”, *ACM Trans. Comp. Syst.*, 17:2, pp. 41-88, May 1999.
- [3] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, “Building peer-to-peer systems with Chord, a distributed lookup service”, *Proc. 8th Wshp. Hot Topics in Operating Syst., (HOTOS-VIII)*, May 2001.
- [4] A. Demers, D.H. Greene, J. Hauser, W. Irish, J. Larson, “Epidemic algorithms for replicated database maintenance”, *Proc. 6th ACM Symp. Principles of Distributed Computing (PODC)*, pp. 1-12, 1987.
- [5] D. Kempe, J. Kleinberg, A. Demers. “Spatial gossip and resource location protocols”, *Proc. 33rd ACM Symp. Theory of Computing (STOC)*, pp. 163-172, 2001.
- [6] A. Rowstron, P. Druschel, “Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems”, *Proc. IFIP/ACM Middleware*, 2001.
- [7] R. van Renesse, Y. Minsky, M. Hayden, “A gossip-style failure detection service”, *Proc. IFIP Middleware*, 1998.
- [8] *Proc. 1st Intl. Wshp. Peer-to-Peer Systems (IPTPS), LNCS 2429, Springer-Verlag*, 2002.
- [9] S. Saroiu, P.K. Gummadi, S.D. Gribble, “A measurement study of peer-to-peer file sharing systems”, *Proc. Multimedia Computing and Networking (MMCN)*, 2002.
- [10] Internet Traffic Archive, <http://ita.ee.lbl.gov>
- [11] Fireflies of Selangor River, Malaysia, <http://www.firefly-selangor-msia.com/fabout.htm>