Topaz peer-to-peer platform

Brian F. Cooper
College of Computing
Georgia Institute of Technology

December 16, 2004

1 Introduction

Topaz is an platform for building peer-to-peer search systems. The Topaz software provides communications and message routing that underly the peer-to-peer application. Topaz is also extensible, providing support both for adding application-specific modules, and for changing the behavior of the underlying peer-to-peer protocol.

In this paper, we describe the design of the Topaz system. First, we list some sample applications that Topaz can support. Then, we present the SIL model, an abstract model of unstructured peer-to-peer overlays. Finally, we discuss the architecture of the Topaz system, a concrete implementation of the SIL model.

1.1 Applications

The current dominant application of peer-to-peer information networks is multimedia filesharing. However, with advances in technology P2P can be useful for a variety of other interesting applications. For example, P2P has been proposed as an architecture for searching the World Wide Web [5]. There are also interesting applications in other areas.

Corporate networks – A multinational corporation may have offices scattered all over the globe. Each of these offices can produce documents (design documents, sales reports, customer surveys, and so on) that are of interest to other parts of the company. Because these documents consist mainly of text data, and because offices continually produce and update these documents, it makes sense to leave them on the computers of the scattered offices and provide a distributed searching mechanism to find documents. A scalable peer-to-peer information infrastructure built on Topaz would serve as a useful tool for information discovery in this scenario.

Ubiquitous information sources – Many people have personal digital assistants (PDAs), PDA cell phones, personal websites, web logs, and other ubiquitous information sources. Some of these sources (such as web logs and personal websites) are already networked, while the increasing prevalence of wireless hotspots means that even PDAs are increasingly networked. Owners of these devices may decide to make some information available for use by others. A peer-to-peer information network that can scale to millions of devices and sources (as is our goal for Topaz) can provide a critical infrastructure for finding this shared information. For example, one user might use his PDA to "blog" that the traffic on I-75 is particularly bad on a certain day, and another user may wish to search for this information before she decides to take I-75. A third user may record his reactions to a movie on his PDA as he is walking out of the theater to allow other users to search for and collect such immediate reactions before deciding whether to see the movie.

Digital libraries – Increasingly, universities, public libraries, museums and government agencies are making digitial content available. These sites typically provide local searching capabilities through a website

or digital library protocol (such as Z39.50 [2]). However, in order to find content, users typically have to know which digital library to look in. Recent work in the digital library community has developed "meta-searchers" that can search multiple digital libraries in order to find information. However, meta-searchers encounter complex semantic issues, as each digital library uses its own information structure and query language. A peer-to-peer information network built on Topaz can allow digital libraries to participate in a distributed information discovery system by layering keyword search-based peer software on top of their existing local search infrastructure.

The requirements and workloads for each of these applications vary. Moreover, even within an application, the workload can vary over time. Traffic can suddenly shift as users search for "hot topics," such as the latest Mars rover pictures made available by NASA's digital library. Our goal is to design Topaz as a general framework that can adapt to the requirements of each application, as well as to other emerging applications.

2 Overlay topologies

A *peer-to-peer search network* is a set of peers that store, search for, and transfer digital documents. Users submit content-based searches, such as keyword searches, metadata searches, and so on. Each peer in the search network maintains an index over its content (such as an inverted list of the words in each document) to assist in processing searches. We assume that the index is sufficient to answer searches, even though it does not contain the whole content of the indexed documents. After a user receives search results, he can contact the peers holding the actual content directly to download documents listed in the results.

The search network forms a partially connected *overlay* network on top of the fully-connected underlying Internet. Peers are nodes in this overlay. The topology of the overlay determines where indexes are placed in the network, and how queries reach either a data repository or an index over that repository's content. Peers that are neighbors in the overlay are connected by network links that are logically persistent, though they may be implemented as connection-oriented or connectionless.

Topaz is a platform for creating unstructured peer-to-peer networks. The canonical examples of unstructured networks are Gnutella, with is pure-search topology, and Kazaa, with its supernode topology. Each of these networks are a special instance of the more general unstructured architecture. The essential characteristic of an unstructured topology is that peers can have the freedom to connect to a wide variety of other peers. In contrast, structured networks such as Chord [8] or CAN [7] have strict rules about which neighbors a node can have, in order to enforce certain performance properties. An unstructured network may not be able to enforce the performance properties offered by a structured network, but is significantly easier to maintain (since strict constraints do not have to be checked) and provides enhanced scalability and robustness (since peers need only know about, and react to, a small number of neighboring peers.

We have developed a general model for describing unstructured peer-to-peer networks, called the *Search/Index Link* (SIL) model. Topaz is a concrete implementation of the abstract SIL model. The SIL model allows us to describe and visualize the overlay topology of search networks. In the SIL model, there are four kinds of network links, distinguished by the types of messages that are sent, and whether a peer receiving a message forwards the message after processing it:

- A *non-forwarding search link* (NSL) carries search messages a single hop in the overlay from their origin. For example, a search generated at one peer A will be sent to another peer B, but not forwarded beyond B. Peer B processes each search message and returns results to A.
- A *forwarding search link* (FSL) carries search messages from A to B. Peer B will process each search message, return search results to A, and forward the message along other forwarding search links originating at B. If A is not the originator of the query, it should forward any search results received from B (and any other peers) along the FSL on which A received the query.

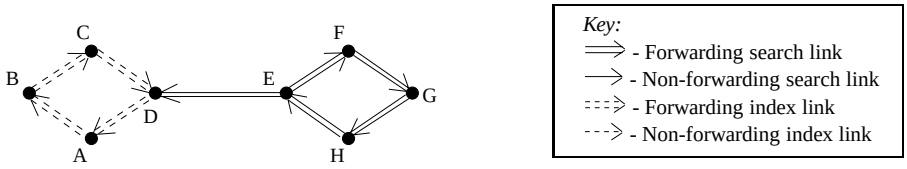

Figure 1: Sample network with forwarding search and index links.

- A *non-forwarding index link* (NIL) carries index update messages one hop in the overlay from their origin. That is, updates occurring at A will be sent to B, but not forwarded. Peer B adds A's index entries to its own index, and then effectively has a copy of A's index. Peer B need not have a full copy of A's content.
- A *forwarding index link* (FIL) carries index update messages from A to B, as with non-forwarding index links, but then B forwards the update message along other forwarding index links originating at B.

Note two points about this model. First, overlay network links are directed communications channels. A link from peer A to peer B indicates that A sends messages to B, but B only sends messages to A if there is also a separate link from B to A. Modeling links as directed channels results in a more general model. An undirected channel can be modeled as a pair of directed links going in opposite directions. For example, the links in Gnutella can be modeled as a pair of FSLs, one in each direction. Second, by having both forwarding and non-forwarding links, our model can describe a wide range of systems. For example, in some systems [9, 3], peers exchange indexing information directly with their neighbors but this information is not disseminated to the whole network.

In our initial model, we assume that a peer forwards search messages on all outgoing FSLs. Similarly, the peer forwards update messages on all outgoing FILs. Of course, there are other alternatives to this "flooding" approach. For example, messages may be forwarded on a randomly selected link (so called "random walk" searching [6, 1]) or on the link that seems most likely to lead to peers with matching content (so called "intelligent searching" [4]). We plan to investigate these options as part of this project to examine the interplay between search routing strategies and index routing strategies. However, as a starting point we can assume the flooding approach. To avoid infinite propagation of messages when using flooding, for FSLs and FILs, messages should have unique ids, and a peer should discard duplicate messages without processing or forwarding them.

One novel aspect of our model is explicit modeling of index links, which allows us to examine index routing in the same way as query routing. A peer uses an *index link* to send copies of index entries to its neighbors. These index entries allow the content to be searched by the neighbors without the neighbors having to store the peer's actual content. For example, consider a peer A that has an index link to a peer B. When B processes a query, it will return search results both for its own content as well as for the content stored at A. Peer A need not process the query at all. We say that B is *searched directly* in this case, while A is *searched indirectly*. Whenever a peer creates a new index entry or modifies an existing entry, it should send a message indicating the change along all of its outgoing index links. A peer might create an index over all of its locally stored documents when it first starts up, and should send all of the index entries to each of its index link neighbors. Similarly, if a peer deletes a document, it would remove the corresponding entries from its own index as well as notifying its index link neighbors to do the same.

Figure 1 shows a network that contains both search and index links. This figure contains only forwarding links; we will see examples later of non-forwarding links. Peers A, B, C and D are connected by a "ring" of FILs. An index update occurring at peer A will thus be forwarded to B, C, D and back to A (A will not forward the update again). In fact, all four of the peers $A...D$ will have complete copies of the indexes at the other three peers in the index "ring". Peers E, F, G and H are connected by FSLs, and a search originating

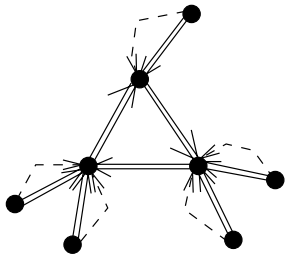

Figure 2: A supernode network.

at any peer $E...H$ will reach, and be processed by, the three other peers on the search "ring." Notice that there is also an FSL between E and D. Any query that is processed by E will be forwarded to D, who will also process the query. Since D has a copy of the indexes from $A...C$, this means that any query generated at E, F, G and H will effectively search the content of all eight peers in the network. In contrast, a query generated at peers $A...D$ will be processed at the peer generating the query, and will only search the indexes of the peers $A...D$.

A search path from X to Y indicates that queries submitted to X will eventually be forwarded to Y, either through a sequence of FSLs or by a single NSL. For example, in Figure 1 there is a search path from F to D but not from D to F. There is (trivially) a search path from a peer to itself. Similarly, an *index path* from X to Y is a sequence of FILs from X to Y, or one NIL from X to Y. In this case, X's index updates will be sent to Y, and Y will have a copy of X's index.

The SIL model can be used to describe a wide variety of unstructured network topologies. An example supernode network, modeled using SIL's search and index links, is shown in Figure 2. The supernodes in the center of the network store indexes for all of the leaf (non-supernode) peers. This property is indicated by the NIL connections between the leaf peers and supernodes. Leaf peers also submit their searches to supernodes, who process them and forward them to other supernodes. This forwarding process is represented by the network of FSL connections between leaf peers and supernodes, and between a supernode and other supernodes.

3 Topaz architecture

In this section we present the architecture of the Topaz system. Topaz is an extensible platform for building peer-to-peer applications. Fundamentally, Topaz implements an unstructured overlay network consisting of search and index links, as described in Section 2. This network consists of a set of peers, each running client software consisting of the Topaz client infrastructure and any additional application specific modules.

Each Topaz peer is identified by a node ID (IP address and TCP port). Each peer is responsible for receiving, processing and/or forwarding several types of messages:

- *Search messages*, containing a message ID and a search string.
- *Search responses*, containing the message ID of a corresponding search message, a node ID and search results from that node ID.
- *Update messages*, containing a node ID and index entries from that node ID.
- *Ping messages*, containing a message ID and node ID
- *Pong messages*, containing a message ID of a corresponding Ping message, and a node ID of the responding node.

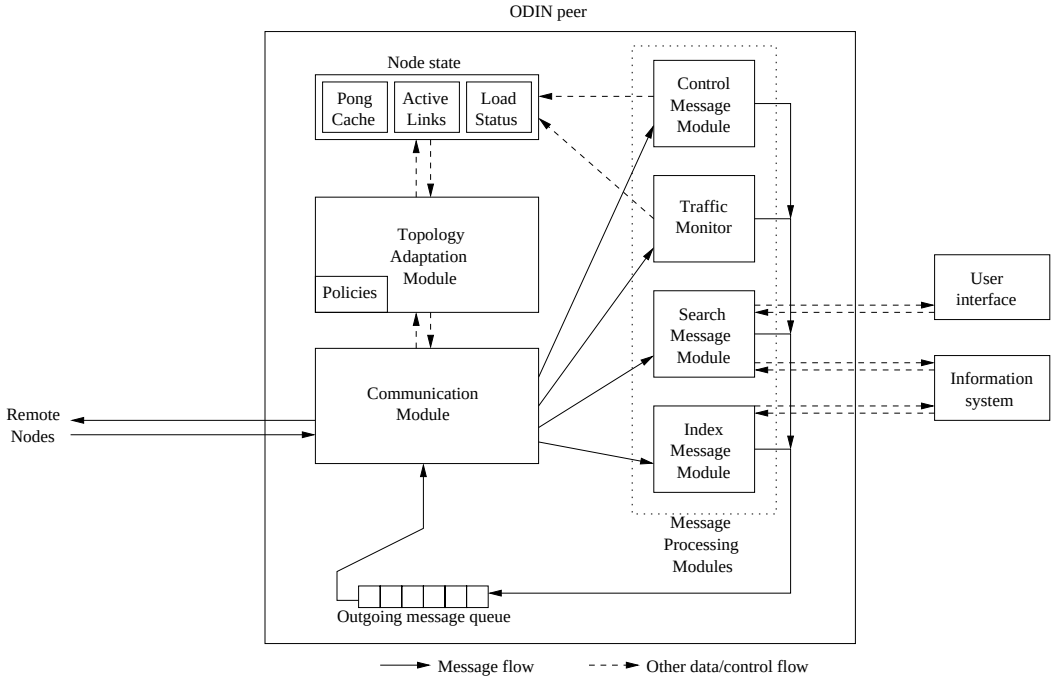

Figure 3: Topaz peer architecture.

This is a simple protocol based on the original Gnutella protocol. Of course, as our implementation proceeds, we may have to modify or enhance this protocol.

When a user inputs a search at an Topaz peer, that peer generates a *search message* containing that search. This search message is forwarded according to the FSL and NSL structure of the overlay. Any peers that have matching content reply with a *search response message*, which is routed back to the peer initiating the search along the same path that the search message reached the peer. When a peer's content changes, it generates an *update message* which it forwards according the the FIL and NIL structure of the overlay. When a node joins, and periodically thereafter, each node generates a *ping message*, announcing its presence to the network. This allows nodes to learn about each other. When a node receives a ping message, it responds with a *pong message*, identifying itelf to the pinging node, and this pong is routed back to the pinging node along the same path that the ping message reached the responding peer.

The architecture for a single peer is shown in Figure 3. Each peer contains a series of functional modules, whose purpose is described below, as well as some node state. This node state includes:

- *Pong cache*: a list of ids of other nodes in the network. On startup, this cache will be empty, and the node will have to pre-cache ids received from a hostcatcher at a well known address, as is common in P2P systems. Alternatively, an anycast service can be used to locate some Topaz peers and add their IDs to the cache.
- *Active links*: a list of the node's currently active neighbors, along with the link type (FSL, NIL, etc).
- *Load status*: current statistics about the load on the node.

The node also interacts with a *user interface*, through which a user submits queries and receives results, and an *information system*, which handles searches over the node's local content as well as searches over the cached indexes of remote nodes.

The *communication module* handles the low level details of forming and breaking network connections to remote nodes. In our initial prototype, these will be TCP connections, although since the connection

functionality is encapsulated in the communication module it will be easy to change to some other protocol (say, UDP) if necessary. Outside of the communication module, the peer understands only the concept of FSL, FIL, NSL, and NIL links to neighbor nodes and does not have to worry about the underlying communication infrastructure.

Each message that the communication module receives is sent to a series of *message processing modules* (MPMs). A message processing module is a module that takes messages, performs some processing, and then optionally generates new messages to be sent to other neighbors. These new messages may just be copies of the input messages that need to be forwarded, such as with search messages, or may be completely new messages, such as when a node generates a pong message to respond to a ping message. Each message that the communication module receives from a remote peer is forwarded to each of the MPMs.

Our initial design utilizes four different MPMs:

- *Control message module*: handles pings and pongs. Whenever this module receives a ping message, it caches the pinger's node id in the pong cache, responds with a pong, and forwards the ping. Whenever this module receives a pong message, it caches the ponger's node id, and forwards the pong if the corresponding pinger was a remote node. Periodically this module generates new pings.
- *Traffic monitor*: monitors all messages to get a picture of the load on the node. Whenever a message is received, this module updates the load status.
- *Search message module*: handles search and search response messages. Whenever the user enters a search, this module generates a search message. Whenever this module receives a search message, it searches the information system and generates a search response message if any results are found. The search message is also forwarded. Whenever a search response message for a user's search is received, the results are reported to the user.
- *Index message module*: handles update messages. Whenever a change occurs in the information system, this module generates an update message. Whenever this module receives an update message, it updates the information system with the index information, and forwards the update message.

When an MPM generates a message, it is added to the *outgoing message queue* so that the communication module can forward it to the node's neighbors. MPMs tag each generated message with a set of ids representing the neighbor to which to send the message. In this way, the routing and forwarding decisions are made entirely by the MPMs.

MPMs are a general abstraction, and the peer is extended by adding new modules, or replacing existing modules, to provide new functionality. For example, a basic peer's search message module forwards search messages on all outgoing search message links. To implement an alternative protocol, say, random walk searches, we would replace the search MPM with one that selects one random link to forward the search message on. The other message processing modules would be unaffected by this change. Similarly, if a new type of message was added to the system, a new module could be created and registered with the communication module to handle this new message. Because message types and MPMs are extensible, the communication module simply sends all messages to all MPMs, and each MPM decides on its own if it is interested in that particular message type. For example, the search message module will only be interested in search messages and search responses, while the traffic monitor will be interested in all of the messages, so that it can accurately measure the total load on the node. Thus, the communication module does not have to understand or process messages, and all message processing functionality is encapsulated in the MPMs.

The final piece of the architecture is the *topology adaptation module*. This module implements the dynamic topology adaptation by telling the communication module to **connect()** and **break()** links. The topology adaptation module maintains the active links list of neighbors, which is used by the MPMs to determine which nodes are to receive messages. The topology module also uses the load status to determine when to make or break links, and the pong cache to determine which new nodes to connect to. The topology

module uses a set of *policies* to guide its actions. These policies determine how many connections each peer tries to make, what types of connections are made, which peers are preferred as neighbors, and any other configuration setting that influences the topology construction. In our initial prototype, changes must be made by replacing the basic policy module with a new module. As the project progresses, we plan to implement a policy language that will allow policies to be specified in a text configuration file.

The architecture is implemented in Java, as it provides good support for networking, threading, timers and other components we need. Our message protocol is designed to be both open and extensible, and we plan to use XML to encode the messages (and to publish the XML schema). The current protocol is based on transferring serialized Java objects, but this will soon be replaced by XML messages. These design decisions, like our others, may change as we gain experience with our implementation.

References

[1] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev. E*, 64:46135–46143, 2001.

[2] ANSI/NISO. Z39.50-1992, American national standard information retrieval application service definition and protocol specification for open systems interconnection, 1992.

[3] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scale. In *Proc. SIGCOMM*, 2003.

[4] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proc. CIKM*, 2002.

[5] Jinyang Li, Boon Thau Loo, Joe Hellerstein, Frans Kaashoek, David R. Karger, and Robert Morris. On the feasibility of peer-to-peer web indexing and search. In *Proc. IPTPS*, 2003.

[6] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of ACM Int'l Conf. on Supercomputing (ICS'02)*, June 2002.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM*, Aug. 2001.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM*, Aug. 2001.

[9] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.