Overlay Networks

A Scalable Alternative for P2P

Diego Doval and Donal O'Mahony • *Trinity College Dublin*

Overlay networks create a structured virtual topology above the basic transport protocol level that facilitates deterministic search and guarantees convergence.

Peer-to-peer systems are distributed systems that operate without centralized organization or control. To find a particular piece of data within the network, P2P systems explicitly or implicitly provide a lookup mechanism, or *locator function,* that matches a given string, or *key*, to one or more network nodes responsible for the value associated with that key. P2P nodes interoperate by using the same software or the same set of network-based APIs.

Current Internet P2P applications typically provide locator functions using time-to-live (TTL) controlled-flooding mechanisms. With this approach, the querying node wraps the query in a single message and sends it to all known neighbors. The neighbors then check to see whether they can reply to the query by matching it to keys in their internal database. If they find a match, they reply; otherwise, they forward the query to their own neighbors and increase the message's hop count. If the hop count passes the TTL limit, forwarding stops. The TTL value thus defines a boundary or "horizon" for the query that controls its propagation.

However, flooding-based systems don't scale well because of the bandwidth and processing requirements they place on the network, and they provide no guarantees as to lookup times or content accessibility. Overlay networks can address these issues. Overlay networks have a network semantics layer above the basic transport protocol level that organizes the network topology according to the nodes' content, implementing a distributed hash table abstraction that provides load balancing, query forwarding, and bounded lookup times.

Overlay networks are evolving into a critical component for self-organizing systems (see, for example, the multigroup effort at www.project-iris.net). Here we outline the differences between flooding-style and overlay networks, and offer specific examples of how researchers are applying the latter to problems requiring high-speed, self-organizing network topologies.

Flooding-Style Networks

Figure 1 (next page) shows a P2P-style search with TTL-controlled flooding. In this example, node N_q is requesting the associated value of a key located in N_r (for which only N_r can provide the value). Nodes that can't answer the query forward it to their neighbors, eventually reaching N_r, which returns the result directly to the requesting node; the concentric circles indicate the number of message hops. In a sense, the network itself resolves the requested lookup.

This example underscores the problems of flooding-style P2P networks. Even though only N_r can answer the query, all the nodes within TTL-range must process it. Also, if the value had been stored in node N_{ar} the query result would not be found unless the message's TTL was set to a high-

 1089-7801/03/$17.00©2003 IEEE Published by the IEEE Computer Society

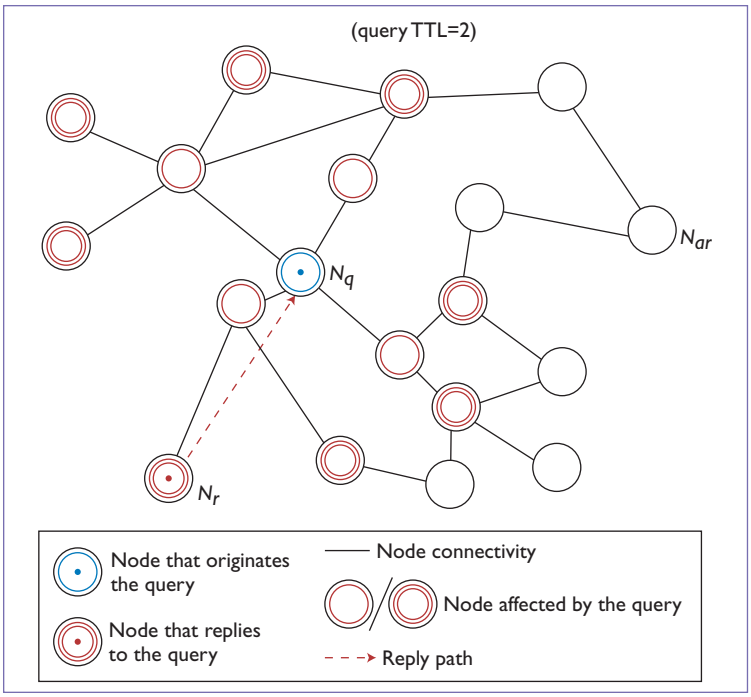

Figure 1. Sample TTL-based P2P network and query. The N_q node transmits a query requesting the value of a key located in N_r. Concentric circles indicate the number of message hops.

er value, potentially flooding the entire network. The Gnutella Network (www9.limewire.com/developer/gnutella_protocol_0.4.pdf) essentially uses this P2P scheme. In Freenet, the query is forwarded according to a more sophisticated cache-based routing strategy, and the result returns through the request's exact node-to-node path, therefore guaranteeing local anonymity.[1] Networks such as Gnutella organize nodes independently of the underlying physical topology; neighbors might exist within the same subnetwork or across the Internet.

Although some networks adapt to the underlying physical topology, such optimization is not required for the algorithm to operate properly. These networks are unstructured: nodes attach to the network according to measures unrelated to content, such as join-order, connection speed, and even physical proximity, creating a random connection topology. Although this approach makes maintaining connections simpler, it has two problems:

- *Content location and network topology are uncorrelated.* Network searches are essentially open ended, forcing protocols to use TTL measures to control message propagation and avoid flooding the whole network. Thus, available content might not be accessible to all network nodes, and a query hit cannot be guaranteed even if the target node is connected to the network.
- *The network is random.* As a result, searching for a particular element within the horizon has a theoretical limit of N hops, where N is the number of nodes within the query's reach. In practice, however, the networks typically traverse different sections of the graph in parallel, reducing lookup times. Still, strictly speaking, queries on an unstructured P2P network tend to have lookup complexity of the order of N, or $O(N)$, hops.

Such limitations are not critical for applications such as file sharing. However, for many content-location applications — such as reliable distributed data storage — networks must find content whenever it's available. All content must therefore be "reachable" by the content-location service, network load must be constrained, and search times must be bound by a predictable limit to avoid arbitrary reply times. In other words, search must be *deterministic*. Overlay networks fulfill these requirements.

Overlay Networks

Overlay networks such as the Content Addressable Network (CAN),[2] Chord,[3] Pastry,[4] and Viceroy[5] create a virtual topology on top of the physical topology. In this sense, TTL-based P2P networks are also a type of overlay, but we use the term here to refer only to networks that create virtual topologies based on node-content attributes. Some networks, such as Chord, organize the network on the basis of each participating node's IP address; other networks use the node's stored data as the organizing content.

How They Work

Overlay networks share four qualities:

- Guaranteed data retrieval
- Provable lookup-time horizons (typically $O(\log N)$ with N being the number of network nodes)
- Automatic load balancing
- Self-organization

Because overlay networks define neighbor nodes by content stored, they can change search from a standard graph-traversal problem into a localized iterative process. In this process, each hop brings the query closer to its target set of hops, which can be calculated according to a mathematical function. This reduces the overall network load and

makes the query process deterministic. In abstract terms, an overlay network operates like a distributed hash table by allowing key insertion, querying, and removal. Typically, it derives those keys from the node-exposed content by, for example, using a consistent hashing algorithm such as the secure hash algorithm (SHA-1).

An overlay network's connectivity pattern is different from that obtained using a TTL-based algorithm in that it is structured and typically symmetrical. The structure is based on one or more mathematical functions that determine how the nodes are connected. The network's structure contributes to the overlays' bound lookup times. When nodes fail, overlay network algorithms provide mechanisms that let the network recover and recreate or maintain an appropriate network structure.

An important difference between overlay networks and unstructured P2P networks is that overlays lookup data on the basis of identifiers derived from the content, and thus don't directly support keyword-based searching. Although work is ongoing for layering keyword searching on top of overlays, whether it can be done efficiently enough to support large-scale networks is still an open problem.

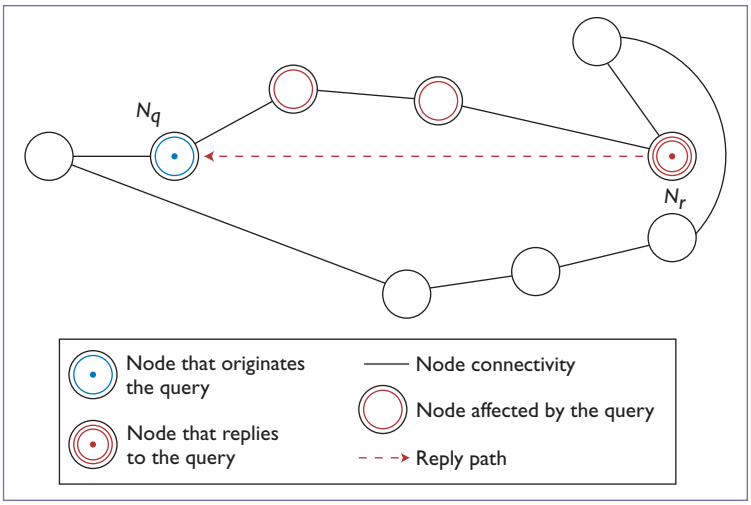

Figure 2. Sample overlay network and query. This structured topology is typical of overlay networks, though the algorithms that build the overlay (and resulting structure) vary according to network type.

Example Overlay Networks

As an example, let's consider a network in which nodes want to publish a given storage identifier, as in a distributed database system. In this case, the node-exposed content is also its identifier; we define it as a positive integer value and skip the hashing step. If the identifiers are universally unique, we can establish a few simple rules to create an overlay topology:

- Each overlay node has two neighbors: the node whose value is the next available (higher) integer, and the node whose value is the previous available (lower) integer.
- If the current node is the network's lowest or highest identifier, one of the neighbors will be the opposite value in the available node range (that is, the highest or the lowest, respectively).
- To join the network, a node must perform an out-of-band request — such as a broadcast — to find another network node. The incoming node can then use the search function to find the network "slot" where it should insert itself.

The search process is simple: the node that initiates the query determines the relation between its own value and the target value. If the target value is higher than the node's value, the node passes the request to its higher-value neighbor, if it is lower it passes it to its lower-value network. This local decision process continues until the request reaches the destination node, which replies directly to the requester, sending its physical network address for additional operations.

Figure 2 shows a hypothetical overlay network built using our simple example algorithm propagating a query. Because overlay nodes are connected according to the content stored in them, queries can be routed efficiently to the target.

This example is unrealistic because the search time is bound but linear (the maximum number of hops is $N/2$), which creates unacceptable lookup times. It also fails to deal with recovery and possible loops created by missing nodes in the topology. However, the example does show how a set of simple rules lets nodes use their content to self-organize and provide bound lookup times. Real-world overlay network algorithms use more complex rules to organize the nodes.

Chord's algorithm offers a more relevant example. Chord establishes a single successor — a next node in the chain — for each node, thus defining a global ring topology (similar to our simple example, but unidirectional). This basic algorithm implies one connection per node, and is thus inherently resilient to node joins, leaves, or failures. Chord then extends the basic successor node with a set of "fingers" — to other, more distant nodes — according to powers of two. So, for a value n, the neighbors will be those nodes matching a rule that evolves according to $n + 2^0$, $n + 2^1$, $n + 2^2$ ... $n + 2^m$ where m is the number of bits in

the identifier (2^m thus defines the maximum number of nodes supported). Formally, a Chord node's fingers are defined as each of the ring's successors for node $(n + 2^{k-1}) \bmod 2^m$ $1 < k < m$.

Because the fingers increase according to the power of two, each step cuts the distance to the target by half, resulting in a lookup complexity of $O(\log n)$. To maintain correctness, Chord requires only that the successor node pointer be correct, and it can revert to the basic successor-only scheme at any point if the fingers table is damaged (by node failures, for example). Because finger tables are small, nodes can keep them valid via a periodic "stabilization" algorithm. As long as the initial network is stable, researchers have found that the system maintains $O(\log N)$ lookup times even with 50 percent probability of node failure.[3] Finally, Chord stores multiple key-value pairs in each node, automatically balancing the node load as new network entrants arrive.

Current overlay networks are useful for applications that require reliable, highly scalable, and self-organizing storage and lookup for unique key-value pairs. This includes distributed databases, processing clusters, and deterministic search applications. Many of the systems we describe here provide open-source library implementations that developers can use directly for building distributed applications. See, for example, the project pages on Chord (www.pdos.lcs.mit.edu/chord), Tapestry (www.cs.berkeley.edu/~ravenben/tapestry), and Pastry (http://research.microsoft.com/~antr/Pastry/download.htm).

Conclusion

Researchers are using overlay networks in diverse applications, ranging from Internet routing to distributed network storage. The overlay-based Internet Indirection Infrastructure (i3) routing system, for example, aims to simplify network services' deployment and management by decoupling the acts of sending and receiving.[6] This additional level of indirection allows for more flexibility in node mobility and in service location and deployment. Researchers are also successfully deploying overlay networks as part of distributed storage systems, such as the cooperative file system.[7] CFS interprets the Chord network's stored values as a file system, and includes features such as replication for increased robustness. Finally, researchers have applied the Pastry system to various end-user applications, such as cooperative Web caching, group notification, and instant messaging.

Overlay network algorithms are the subject of ongoing research and development. In particular, researchers are working to reduce network operation costs, such as multiple concurrent node join and leave, fault tolerance, security, and physical proximity (by modifying the overlay to adapt better to the underlying physical topology).

References

1. I. Clarke et al., "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Designing Privacy Enhancing Technologies: Int'l Workshop Design Issues in Anonymity and Unobservability*, H. Federrath, ed., Springer-Verlag, 2001, pp. 46–66.
2. S. Ratnasamy et al., "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM*, ACM Press, 2001, pp. 161–172.
3. I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM*, ACM Press, 2001, pp. 149–160.
4. A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. Int'l Conf. Distributed Systems Platforms (Middleware)*, ACM Press, 2001, pp. 329–350.
5. D. Malki, M. Naor, and D. Ratajczak, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," *Proc. ACM Principles of Distributed Computing*, ACM Press, 2002, pp. 183–192.
6. I. Stoica et al., "Internet Indirection Infrastructure," *Proc. ACM SIGCOMM,* ACM Press, 2002, pp. 73–88.
7. F. Dabek et al., "Wide-Area Cooperative Storage with CFS," *ACM Symp. Operating System Principles,* ACM Press, 2001, pp. 202–215.

Diego Doval is CTO and cofounder of Clevercactus, and a doctoral student at Trinity College, Dublin. His research interests include self-organizing and wireless networks, complexity theory, dynamical systems, and molecular nanotechnology. He has a BS in computer science from Drexel University, Philadelphia. He is a member of the ACM. Contact him at diego.doval@cs.tcd.ie.

Donal O'Mahony is senior lecturer at Trinity College where he leads the Networks and Telecommunications Research Group, which investigates a range of core networking and telecommunications issues involved with fourth-generation mobile systems. His other research interests include cryptographic techniques and protocols, particularly in the area of electronic payment. He is the author of several books, including *Electronic Payment Systems for E-Commerce* (Artech House, 2001). Contact him at donal.omahony@cs.tcd.ie.