Scalable P2P Search

Daniel A. Menascé • *George Mason University* • *menasce@cs.gmu.edu*

Although the traditional client-server model first established the Web's backbone, it tends to underuse the Internet's bandwidth and intensify the burden that dedicated servers face as their load increases.[1] Peer-to-peer computing relies on individual computers' computing power and storage capacity to better utilize bandwidth and distribute this load in a self-organizing manner. In P2P, nodes (or peers) act as both clients and servers, form an application-level network, and route messages (such as requests to locate a resource). The design of these routing protocols is of paramount importance to a P2P application's efficiency: naive approaches — such as Gnutella's flood routing, for example — can add traffic.[1,2] P2P systems that exhibit the "small world" property — in which most peers have few links to other peers, but a few of them have many — are robust to random attacks, but can be highly vulnerable to targeted ones.[3,4]

P2P computing also has the potential to enhance reliability and fault tolerance because it doesn't rely on dedicated servers.[1] Each peer maintains a local directory with entries to the resources it manages. It can also cache other peers' directory entries. Important applications of P2P technologies include distributed directory systems,[1] new e-commerce models,[5] and Web service discovery, all of which require efficient resource-location mechanisms.

We can view the resource-location problem simply as, "given a resource name, find the node or nodes that manage the resource." I call this the *deterministic location problem*. In a very large network, contacting all the nodes to locate a resource is clearly not feasible. Instead, consider this modified problem statement: "given a resource name, find with a given probability the node or nodes that manage the resource." I call this a *probabilistic location approach*. I present here a probabilistic resource-location protocol that can trade performance and scalability for the probability P_f that a resource will be located. The protocol behavior depends on probabilistic parameters that we can tune to obtain a desired value of P_f. The goal is to achieve a probability P_f close to 1 with a cost much lower than that of the deterministic case. We can measure cost in terms of the number of messages exchanged, bandwidth used, or number of peers contacted.

Probabilistic Search Protocol

Consider a large, fully connected network with N peer nodes and an unknown topology. Every peer has a local directory (LD) that points to locally managed resources (such as files, Web pages, processes, and devices). The LD of peer node D in Figure 1, for example, indicates that the node stores resources g and h. Each peer has total control over its local resources and its LD, which means that resources can be added or deleted from the LD without the other

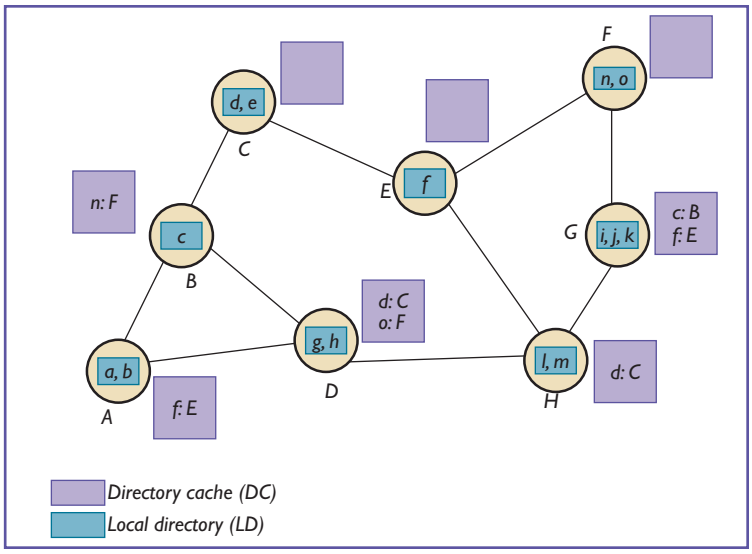

Figure 1. A peer-to-peer computing system. Each peer node has a local directory and a directory cache. The directory cache of peer H*, for example, indicates that resource* d *is located in peer* C*. The directory caches for nodes* C*,* E*, and* F *are empty.*

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

```
r: SearchRequest (source, resource, (s1, …, sm), TTL)

If resource in LD
Then Send ResourceFound (source, resource, (s1, …, sm-1), r) to sm
Else  If (resource, loc) in DC
   Then /* send request to presumed peer */
            Send SearchRequest (source, resource, (s1, …, sm, r), TTL-1) to loc.
   Else If TTL > 0
            Then /* probabilistic broadcast */
               For every v ∈ N(r) do
               Send SearchRequest (source, resource, (s1, …, sm, r), TTL-1)]p;

r: ResourceFound (source, resource, (s1, …, sm), v)

If r ≠ source
Then Begin
       Send ResourceFound (source, resource, (s1, …, sm-1), v) to sm;
       Add (resource, v) to DC
     End
Else request source from peer v /* end */
```

Figure 2. The probabilistic resource-location protocol. It considers two cases: in one, peer r *receives a* `SearchRequest` *(SR) message; in the other, peer* r *receives a* `ResourceFound` *message. In this algorithm,* r: M(a, b, c) *means that message* M *with fields* a, b, *and* c *is received at peer* r; [x]p *means that* x *executes with probability* p.

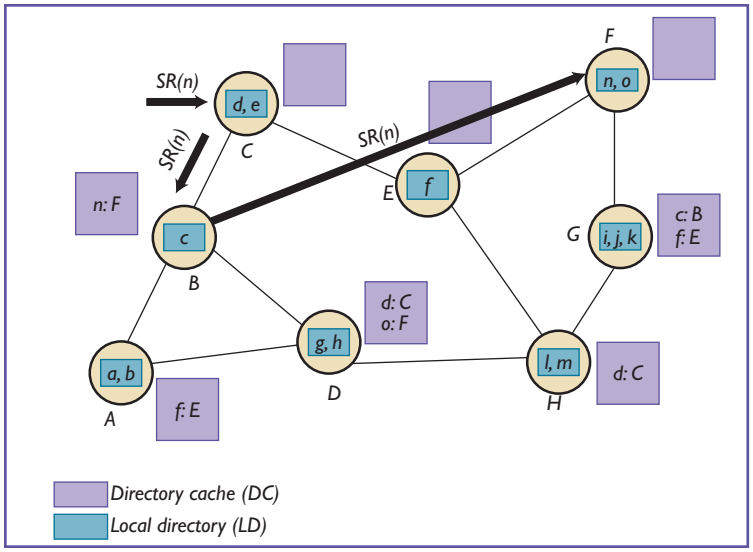

Figure 3. Example of the `SearchRequest` *(SR) message's propagation. The SR message arrives at peer* C *in search of resource* n. *In step 2, this request is probabilistically sent to peer* B, *which finds in its DC the information that resource* n *is in peer* F. *Peer* B *then sends the SR directly to peer* F.

peers' authorization. This assumption distinguishes the protocol described here from other work,[6,7] which assumes that resources can be replicated and can migrate to any node at the P2P system's will.

Each resource has a unique, location-independent, global identifier (GUID) that developers can generate by several means. Freenet, for example, uses secure SHA-1 hashes of file content to generate GUIDs.[3] In a distributed online bookstore application, developers could use ISBNs as GUIDs.

Each peer has a directory cache (DC) that points to the presumed location of resources managed by other peers. The DC of node *A* in Figure 1, for instance, indicates that resource *f* will be in node *E*. Entries in the DC might just hint at the location of remote resources. An entry in the DC is a pair (id, loc) in which id is the GUID of a resource and loc is the network address of a peer node that might store the resource locally. Directory entries migrate from LDs to DCs at other nodes to adjust to access patterns.

Each peer *s* has a local neighborhood *N*(*s*) defined as the set of peers close — one hop away or in the same local area network — to peer *s*. We can extend the notion of neighborhood to indicate a set of peers that have business dealings with a peer, as in an e-commerce setting.

Protocol Operation

Figure 2 provides a high-level description of the probabilistic resource-location protocol. The source is where a search starts in a peer node. The algo-

rithm in the figure uses the following messages:

- `SearchRequest` (*s*, *res*, RevPath, TTL) is a request sent by source *s* to locate resource *res*. RevPath is the sequence of peers that have received this message so far. This sequence is used as a reverse path to the source. The parameter TTL (time to live) limits this message's propagation.
- `ResourceFound` (*s*, *res*, RevPath, *v*) indicates that the resource *res* being searched by source *s* was found at peer *v*. RevPath is the reverse path to the source.

When the peer receives the `SearchRequest` message, it searches its LD first and then its DC. If it finds the resource in the LD, it sends a `ResourceFound` message along the path the `SearchRequest` message traverses until it reaches the source. This message updates the DC at every peer it visits. If the peer finds the resource in the DC, it sends the `SearchRequest` message to the peer pointed to by that DC. This peer might no longer have the resource, so the search continues from that point forward. If a peer does not find the resource in its LD or DC, it will send the request to each peer in its neighborhood with a certain probability *p*, called the *broadcast probability*. This probability could vary with the length of the path that the request traverses.

Figure 3 shows how the `SearchRequest` message is propagated when a request for resource *n* arrives at peer *C*. Resource *n* is not in peer *C*'s LD, so a probabilistic broadcast sends the `SearchRequest` message on to peer *B*. This node checks its LD, but does not find resource *n*. However, the DC in node *B* has an indication that resource *n* might be in peer *F*. Peer B then sends the `SearchRequest` message to *F*.

Figure 4 shows how the `ResourceFound` message travels back to the source (node *C*) via the reverse path used by the `SearchRequest` message — from *F* to *B* to *C*. Note how the DC in node *C* gets updated: the next time resource *n* is requested, node *C* can contact node *F* directly.

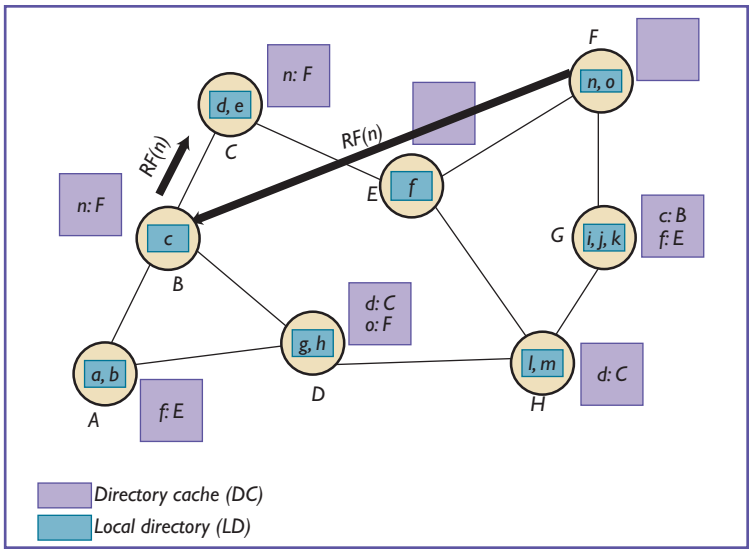

Figure 4. Example of the `ResourceFound` *(RF) message's propagation. The RF message returns to the source (node* C*) following the reverse path used by the SR message. The RF message updates the DC in each of the nodes. In step 3, the DC in node* C *is updated to reflect the fact that resource* n *is in peer* F.

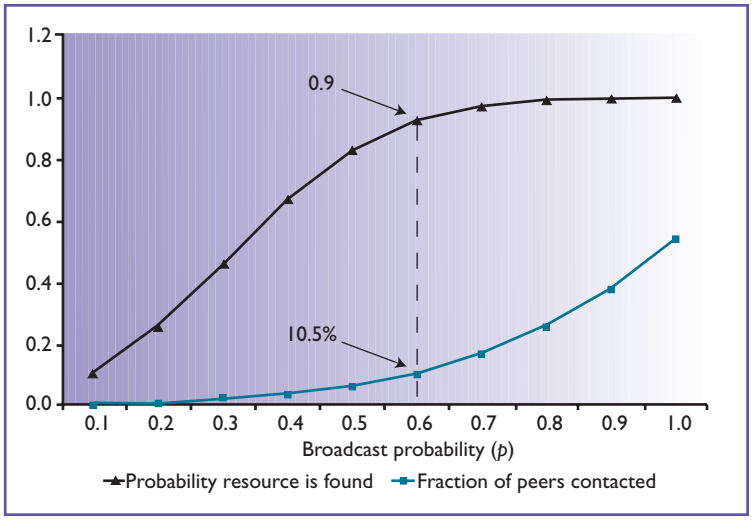

Figure 5. Probability that the resource will be found and the fraction of peers contacted versus the broadcast probability. The curves show that a relatively high probability of finding a resource can be achieved by contacting a small fraction of all peer nodes.

Protocol Performance

Figure 5 shows two curves generated from models that a student and I developed.[8] The figure's graphs assume a network with 120 peers, with an average node degree equal to 5 and a decreasing hit ratio at the DC. The first curve in Figure 5 illustrates the variation of the probability P_f that a resource will be found with the broadcast probability *p*. As the broadcast probability *p* increases, the probability that the resource will be found increases and reaches 0.9 for a value of *p* equal to 0.6. At that value, only 10.5 percent of the peers are involved in the search, as shown in the second curve, which displays the variation of the fraction of peers involved in the search. As the broadcast probability approaches one, the probability that a resource will be found also approaches one. At that point,

Other P2P Efforts

I should mention several relevant P2P systems here, including Gnutella, Freenet, Gridella, and Sun's JXTA Search. Gnutella is an open, decentralized group membership and search protocol used mainly for file sharing. Its goals are dynamic operability, performance and scalability, reliability, and anonymity,[1] but it uses flood routing to broadcast queries, which can generate unnecessary traffic.[2] Ripeanu and colleagues[1] identified a mismatch between Gnutella's overlay network topology and the Internet infrastructure, which they show has critical performance implications.

Aberer and colleagues[2] discuss Gridella — a Gnutella-compatible P2P system that distributes replication over a community of peers and reduces bandwidth requirements because of its superimposition of a binary tree on top of the P2P network. Freenet is a distributed information storage system designed to address information privacy and survivability concerns.[3] Freenet operates a self-organizing P2P network that pools unused disk space across potentially hundreds of thousands of computers to create a collaborative virtual file system. JXTA[4] is a suite of protocols that facilitates P2P communication. JXTA Search is a decentralized P2P search engine in which information providers publish a description of queries they are willing to answer. JXTA Search uses specialized peers (called hubs) that can register with other hubs as information providers. This search engine is a middle ground between the decentralized Gnutella model and the centralized Napster approach.

References

1. M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," *IEEE Internet Computing*, vol. 5, no. 1, 2002, pp. 50–57.
2. K. Aberer et al., "Improving Data Access in P2P Systems," *IEEE Internet Computing*, vol. 5, no. 1, 2002, pp. 58–67.
3. I. Clarke et al., "Protecting Free Expression Online with Freenet," *IEEE Internet Computing*, vol. 5, no. 1, 2002, pp. 40–49.
4. S. Waterhouse et al., "Distributed Search in P2P Networks," *IEEE Internet Computing*, vol. 5, no. 1, 2002, pp. 68–72.

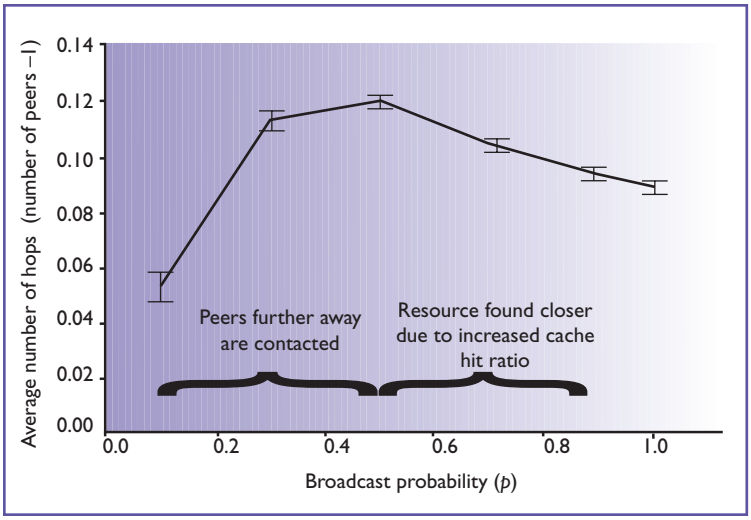

Figure 6. Normalized average number of hops versus broadcast probability. As the broadcast probability increases, more nodes are contacted, and the probability of finding a resource closer to the source increases thanks to an increased cache hit ratio.

a little over 50 percent of the peer nodes are contacted because of the DC's effect. What is most interesting here is that a relatively small broadcast probability can generate a reasonable probability of finding a resource at a very low cost (meaning a small fraction of the peer nodes are involved).

To further evaluate the probabilistic search protocol, I implemented it in a LAN in which each machine simulates multiple peer nodes. I also experimented with randomly generated topologies. To generate the topology for a network with N peers I start with a regular graph with k neighbors in every node and then rewire the graph by taking each edge in turn. With a certain probability, I move it to connect to a randomly chosen vertex. If the resulting graph is not connected, I restart the process. You can use this approach to generate graphs that exhibit the small-world behavior mentioned earlier. Such networks are characterized by a power-law distribution of graph degree $f(x) \sim x^{-t}$ in which $f(x)$ is the distribution of the number of routing table entries.[3]

For each combination of the set of parameters, I generated 10 random topologies with the same initial node degree k and rewiring probability and then generated as many requests per peer as necessary to achieve at least a five-percent precision with a 95-percent confidence interval. The results in Figure 6 assume 30 peers, $k = 4$, a rewiring probability equal to 0.1, and a cache size per peer equal to 1 percent of the total number of resources stored by all peers. Least recently used (LRU) is the cache replacement policy.

Figure 6 shows the variation of the average number of hops $\overline{H}$ needed to find a resource normalized by the total number of peers that must be contacted if all peers (except the source) received a search request message. The curve also shows the measurements' 95-percent confidence intervals. Initially, the average number of hops increases as peers further away from the source are contacted; the resource tends to be found at a greater distance from the source. As p continues to increase, the

increased value of the cache hit ratio allows resources to be found closer to the source, decreasing the value of $\overline{H}$. Note that the maximum value of the curve in Figure 6 occurs for $p = 0.5$. At this value, a resource will be found with probability 0.84 within 3.4 hops from the source on average.

Conclusions

The protocol described here does not address cache replacement policies in the DC or cache invalidation options. I could extend the protocol to allow for invalidation of DC entries when a `SearchRequest` is received because the resource was found in the DC. The figure doesn't show other details of the protocol such as a provision to avoid broadcasting a message more than once. An optimization to the protocol would be to notify the source as soon as the resource is found in addition to sending the `ResourceFound` message to the source through the reverse path. This would reduce search response time.

The design of scalable P2P resource location protocols has drawn the attention of many researchers given that resource location is at the heart of any middleware service. Efficient middleware services for P2P are essential glue for applications running on a multitude of devices including wireless devices and all sorts of appliances.

References

1. L. Gong, "Peer-to-Peer Networks in Action," *IEEE Internet Computing*, vol. 5, no. 1, 2002, pp. 37–39.
2. K. Aberer et al., "Improving Data Access in P2P Systems," *IEEE Internet Computing*, vol. 5, no. 1, 2002, pp. 58–67.
3. I. Clarke et al., "Protecting Free Expression Online with Freenet," *IEEE Internet Computing*, vol. 5, no. 1, 2002, pp. 40–49.
4. T. Hong, "Performance," *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, A. Oram, ed., O'Reilly, 2001, pp. 203–241.
5. W. Meira Jr. et al., "E-Representative: A Scalability Scheme for E-Commerce," *Proc. 2nd Int'l Workshop on Advanced Issues of E-commerce and Web-Based Information Systems* (WECWIS 2000), IEEE CS Press, 2000.
6. S. Ratnasamy et al., "A Scalable Content Addressable Network," *Proc. ACM SIGCOMM*, ACM Press, 2001, pp. 161–172.
7. A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed, Object Location and Routing for Large-Scale Peer-to-Peer Systems," *IFIP/ACM Int'l Conf. Distributed System Platforms (Middleware)*, ACM Press, 2001, pp. 329–350.
8. D.A. Menascé and L. Kanchanapalli, "Probabilistic Scalable P2P Resource Location Services," *ACM Sigmetrics Performance Evaluation Rev.*, vol. 30, no. 2, 2002, pp. 48–58.

Daniel Menascé is a professor of computer science, the codirector of the E-center for E-Business and the director of the MS in E-commerce program at George Mason University. He received a PhD in computer science from UCLA and published the books *Capacity Planning for Web Services* and *Scaling for E-Business* (Prentice Hall, 2002 and 2000). He is a fellow of the ACM and the recipient of the 2001 A.A. Michelson Award from the Computer Measurement Group.