

# Building P2P networks with good topological properties

Gopal Pandurangan\*    Prabhakar Raghavan†    Eli Upfal\*

## Abstract

We describe a simple distributed protocol for building P2P networks with desirable topological properties: connectedness, low diameter and low degree. These properties are important for search (the most common application to date in P2P systems), especially in P2P networks using the Gnutella protocol.

An important feature of our protocol is that it operates without global knowledge of all the nodes in the network. Our protocol is also simple to implement and is easily scalable. Our protocol can be easily incorporated into the standard Gnutella protocol. Also, our stochastic model (used to analyze the protocol) is simple and reasonable and can be of independent interest as a model for P2P systems. We conclude with a discussion of open issues and pointers for future work.

---

\*Computer Science Department, Brown University, Box 1910, Providence, RI 02912-1910, USA. E-mail: {gopal,eli}@cs.brown.edu. Supported in part by the Air Force and the Defense Advanced Research Projects Agency of the Department of Defense under grant No. No. F30602-00-2-0599, and by NSF grant CCR-9731477.

†Verity Inc., 892 Ross Drive, Sunnyvale, CA 94089.

# 1 Introduction and Motivation

Building P2P networks with desirable topological properties is a fundamental problem in P2P computing. The natural question is: what topological properties are desirable? Of course, this depends on the P2P application. Consider search - the most common and pervasive P2P application to date. Clearly, connectivity is important for reachability of search queries. As a motivation for building networks with low diameter, consider P2P systems running the Gnutella protocol [1] (popular examples of such systems are Gnutella [2], Limewire, Bearshare, and Kazaa). They use a flooding-like routing algorithm: queries fan-out from the source node and the search radius is limited by a "Time to Live (TTL)" parameter (this is set to an initial value at the source node, and is decremented in each hop). Thus a low diameter is not only helpful in enlarging the scope of search but also in reducing the associated network traffic; indeed the traffic explosion problem is one of the major obstacles to scalability in Gnutella-like networks. The importance of having a low diameter was demonstrated by a study by the DSS group of the public Gnutella network [3]. It is also desirable to have the diameter grow slowly (say, logarithmic) with the size of the network - this is essential for scalability. In this paper, we focus on building networks of bounded degree (typically a small constant); this will also help the traffic explosion problem in Gnutella-like networks by reducing the fan-out in each stage.

Do the networks constructed by the Gnutella protocol possess the above desirable properties? No. Actually, there is no clear-cut strategy to build Gnutella-like networks. Each servent uses its own heuristic to decide when and where to connect (or reconnect): most begin by connecting to a generic set of neighbors that come with the download, then switch (in subsequent sessions) to a subset of nodes whose names they encountered on a previous session. This free-for-all situation was one of the reasons which led to the fragmentation of the network into disconnected pieces [3].

Our main contribution is a stochastic analysis of a simple local heuristic which, if followed by every servent, results in provably strong guarantees on network diameter and other properties. Our protocol is especially attractive for building enterprise P2P networks; here servents can be implemented to a standard, so that their local behavior results in good global properties for the P2P network they create.

The rest of the paper is organized as follows. In Section 2 we discuss our P2P protocol and its implementation. In Section 3 we propose a simple stochastic model to evaluate the protocol and then state key results with intuitive explanations. We conclude with open issues and a discussion of further work.

## 2 The P2P protocol

We assume that there is a central element called the *host server* which, at all times, maintains a *cache* containing a list of nodes (i.e., their IP addresses). The host server is similar to (or models) websites which maintain a list of host IP addresses to which clients visit to get entry points into the P2P network; for example, <http://www.gnufrog.com/> is a website which maintains a list of active Gnutella servents and a client can join the network by connecting to one or more of these servents.

We also assume (without loss of generality) that the host server maintains  $K$  nodes at all times, where  $K$  is a constant. The host server is reachable by all nodes at all times (this is again similar to the website analogy); however, it need not know of the topology of the network at any time, or even the identities of all nodes currently on the network. We only expect that (1) when the host server is contacted on its IP address it responds, and (2) any node on the P2P network can send messages to its neighbors. In this sense, our protocol demands far less from the network than do (for instance) current P2P proposals (e.g., the *reflectors* of [dss.clip2.com](http://dss.clip2.com), which maintain knowledge of the global topology).

Before we state the protocol we need some terminology. When a node is in the cache we refer to it as a *cache node*. A node is *new* when it joins the network, otherwise it is *old*. Our protocol will ensure that the degree (number of neighbors) of all nodes will be in the interval  $[D, C + 1]$ , for two constants  $D$  and  $C$ ; thus the degree of each node never exceeds a *constant* - one of the guarantees of the protocol.

A new node first contacts the host server, which gives it  $D$  random nodes from the current cache to connect to (or equivalently, we can think of the contacting node as randomly choosing  $D$  nodes from the cache). The new node connects to these, and becomes a *d-node*; it remains a d-node until it subsequently either enters the cache or leaves the network. The degree of a d-node is always  $D$ . At some point the protocol may put a d-node into the cache, i.e., it will be listed as one of the nodes in the cache list. It stays in the cache until it acquires a total of  $C$  connections, at which point it leaves the cache, as a *c-node*. A c-node might lose connections after it leaves the cache, but its degree is always at least  $D$ . A c-node has always one *preferred* connection, made precise below.

## 2.1 P2P protocol for node $v$

Our protocol is summarized below as a set of rules applicable to various situations that a node may find itself in.

1. **On joining the network:** Connect to  $D$  cache nodes, chosen uniformly at random from the current cache.
2. **Reconnect rule:** If a neighbor of  $v$  leaves the network, and that connection was not a preferred connection, connect to a random node in cache with probability  $D/d(v)$ , where  $d(v)$  is the degree of  $v$  before losing the neighbor.
3. **Cache Replacement rule:** When  $v$  reaches degree  $C$  in the cache, it is replaced in the cache by a d-node from the network. Let  $r_0(v) = v$ , and let  $r_k(v)$  be the node replaced by  $r_{k-1}(v)$  in the cache. The replacement d-node is found by the following rule:

```

 $k = 0;$ 
while (a d-node is not found) do
    search neighbors of  $r_k(v)$  for a d-node;
     $k = k + 1;$ 
endwhile

```

4. **Preferred Node rule:** When  $v$  leaves the cache as a c-node it maintains a *preferred connection* to the d-node that replaced it in the cache. (If  $v$  is not already connected to that node this adds another connection to  $v$ .)
5. **Preferred Reconnect rule:** If  $v$  is a c-node and its preferred connection is lost, then  $v$  reconnects to a random node in the cache and this becomes its new preferred connection.

## 2.2 Implementation Notes

We brief remarks on the protocol and its implementation. Our protocol can be easily integrated into the existing Gnutella protocol specification [1] and needs only a little modification to the server software. Note that our protocol does not talk about how servers communicate with each other for distributed search; we assume the same search paradigm as in the Gnutella protocol specification.

1. It is clear from our protocol that it is essential for a node to know whether it is in the cache or not; thus each node maintains a flag for this purpose.
2. The cache replacement rule can be implemented in a distributed fashion by a *local message passing* scheme with *constant storage* per node. Each c-node  $v$  stores the address of the node that replaced it in the cache, i.e.,  $r(v)$ . Node  $v$  sends a message to  $r(v)$  when  $v$  itself doesn't have any d-node neighbors.

3. For rules 2 and 5 a node has to determine whether a neighbor is down. This can be done easily - each node can periodically ping its neighbors to check whether any of its neighbors have gone offline. If a reconnection is needed then the host server is contacted to get the address of a new node.
4. We note that the overhead in implementing each rule of the protocol is constant (or expected constant). This is very important in practice, because even if a protocol is local, it is desirable that neither too much (local) computation nor too many local messages be sent per node. Rules 1, 2, 4 and 5 can be easily implemented with constant overhead. It follows from our stochastic analysis (next section) that the overhead incurred in replacing a full cache node (rule 3) is constant on the average, and with high probability is at most logarithmic in the size of the network (see Section 3).
5. We note that the host server is contacted whenever a node needs to reconnect (rules 2 and 5), and when a new node joins the network. We show that the expected number of contacts the host server receives per unit time interval is constant in our model and with high probability only *logarithmic* in the size of the network; this implies that the network also scales well in terms of the number of “hits” the central server receives.
6. In the stochastic analysis that follows, the protocol does have a minuscule probability of catastrophic failure: for instance, in the cache replacement step, there is a very small probability that no replacement d-node is found. A practical implementation of this step would either cause some nodes to exceed the maximum capacity of  $C + 1$  connections, or to reject new connections. In either case, the system would speedily “self-correct” itself out of this situation (failing to do so with an even more minuscule probability). For either such implementation choice, our analysis can be extended.
7. One may wonder about the preferred connection which is somewhat non-intuitive: it turns out that this feature is essential to maintain connectivity and low-diameter.

### 3 Analysis under a stochastic model

In evaluating the performance of our protocol we focus on the long term behavior of the system in a fully decentralized environment in which nodes arrive and depart in an uncoordinated, and unpredictable fashion. This setting is best modeled by a stochastic, memoryless, continuous-time setting. The arrival of new nodes is modeled by Poisson distribution with rate  $\lambda$ , and the duration of time a node stays connected to the network has an exponential distribution with parameter  $\mu$ . There are many reasons why this is a good model (apart from being mathematically elegant to analyze!). First, this is a natural model if we assume that users are behaving *independently*; second, if we assume that users are behaving unpredictably, then memorylessness property of the exponential distribution is quite reasonable. (This is the reason for a similar model being used to model duration of telephone calls which has proven quite successful in capturing the actual behavior in practice.) One may wonder about nodes which stay connected for an indefinite period of time - e.g., servers. This is not a serious problem for two reasons: first, the exponential distribution does allow the possibility of a few nodes staying for a very long time; second, we can easily extend our analysis to handle the above scenario and all our guarantees still hold.

A useful property of the above model which proves to be convenient for theoretical analysis is that the size of the network converges rapidly to a value very close to  $\lambda/\mu$  (henceforth we denote this quantity by the parameter  $N$ ). This is because the model behaves as an infinite server Poisson queue ([7]).

Before we state our results we need a little terminology. Let  $G_t$  be the network at time  $t$  ( $G_0$  has no vertices). We analyze the evolution in time of the stochastic process  $\mathcal{G} = (G_t)_{t \geq 0}$ . We state only the main results and their intuitive justifications; for rigorous proofs we refer to [5].

Our results are in terms of  $N$ , the size of the network (with high probability) after the process converges (this convergence can be shown to take place after only  $\Omega(N)$  steps).

### 3.1 Connectivity

We show that our protocol keeps the network connected with large probability at *any* instant of time (after an initial time period of  $\Omega(\log N)$ ).

**Theorem 3.1** *There is a constant  $c$  such that at any given time  $t > c \log N$ ,*

$$\Pr(G_t \text{ is connected}) \geq 1 - O\left(\frac{\log^2 N}{N}\right).$$

The main idea in the proof of the above theorem is using the preferred connections as a “backbone”: we can show that at all times, each node is connected to some cache node directly or through a path in the network. The “backbone” is kept connected with large probability by incoming nodes: here the “randomness” in choosing the cache node by an incoming node proves crucial. Also, an important consequence from the proof is that the above theorem does not depend on the state of the network at time  $t - c \log N$ ; therefore it can be shown to recover rapidly from fragmentation - a nice “self-correcting” property of the protocol. More precisely we have the following corollary:

**Corollary 3.1** *There is a constant  $c$  such that if the network is disconnected at time  $t$ ,*

$$\Pr(G_{t+c \log N} \text{ is connected}) \geq 1 - O\left(\frac{\log^2 N}{N}\right).$$

We also answer another natural question: if the network becomes disconnected, what can we say about the connected components in the network? The following theorem tells us that a very large fraction of the network (a constant fraction of the network) remains connected with high probability.

**Theorem 3.2** *At any given time  $t$  (after the network has converged) if the network is not connected then it has a connected component of size  $N(1 - o(1))$ .*

### 3.2 Diameter

Our main result is that of the diameter of the network. We show that not only the network remains connected (previous subsection), but also the diameter is only logarithmic in the size of the network at any time  $t$  with large probability. Formally,

**Theorem 3.3** *For any any time  $t$  (after the network has converged), with high probability, the largest connected component of  $G_t$  has diameter  $O(\log N)$ . In particular, if the network is connected (which has probability  $1 - O(\frac{\log^2 N}{N})$ ) then with high probability its diameter is  $O(\log N)$ .*

The proof of the above theorem is beyond the scope of this brief exposition. However, we would like to point out two key ideas. First, it can be shown that there is a very small (but non-zero) probability that *any* two  $c$ -nodes are connected to each other; however, this alone is not sufficient to guarantee low diameter. Second, using preferred connections we can show that there is a large chunk of connected nodes (at least  $\Omega(\log N)$ ); this is crucial to argue that with high probability *every* node has at least a constant fraction of the nodes in the network within a logarithmic distance from it.

### 3.3 Network stability

It is important to show that the network can maintain a bounded number of connections at each node with high probability - this is crucial to the stability of the protocol. This implies showing there is always a d-node in the network to replace a cache node that reaches capacity  $C$ . Thus the network is “self-sustaining”. We further show that the replacement d-node can be found efficiently - the overhead is expected constant and with high probability is only logarithmic in the size of the network. The crucial component of the protocol which guarantees the above properties is the *probabilistic* feature of the reconnect rule (rule 2): a node reconnects (in the event of losing its neighbor) not deterministically but with probability  $D/d(v)$ , where  $d(v)$  is the degree of  $v$  before losing its neighbor. This essentially guarantees that there will a “large” number of d-nodes in the network at any time with high probability.

Our proofs also yield constraints on the protocol constants  $C$  and  $D$  - this helps choosing their values in practice.

## 4 Discussion and further work

It is important to point out our protocol is concerned with building a good *virtual* network topology which may not match the underlying Internet topology (of course, this may not be a big issue for enterprise P2P). In fact, evidence [6] suggests that these two topologies do not match well. It will be of practical interest [6] to construct topologies that exploits the underlying physical topology - this an area for further theoretical and practical research.

How realistic is our stochastic model? To our knowledge there is no data on the arrival and holding time of users in a P2P network; such a study (say on a public P2P network) will be very useful.

It turns out that the preferred connection component in our protocol is essential: running the protocol without it leads to formation of many small disconnected components. A similar argument can work for other fully decentralized protocols that maintain a minimum and maximum node degree and treat all edges equally, i.e., do not have preferred connections. A nice theoretical problem is to rigorously prove this.

In our protocol we implicitly assume that all nodes have equal capabilities (i.e., storage and number of connections supported) and all links have equal bandwidth. In enterprises with homogeneous systems this is closer to reality, however this is not the case in the Internet. It will be nice to adapt our protocol to incorporate heterogeneous nodes and links.

## References

- [1] The Gnutella Protocol Specification v0.4. [http://www9.limewire.com/developer/gnutella\\_protocol\\_0.4.pdf](http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf)
- [2] Gnutella website. <http://gnutella.wego.com/>
- [3] DSS Group, Gnutella: To the Bandwidth Barrier and Beyond, <http://dss.clip2.com/gnutella.html>, Nov. 6, 2000.
- [4] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.
- [5] G. Pandurangan, P. Raghavan, and E. Upfal. Building Low-Diameter P2P Networks, in *Proceedings of the 42nd annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, 2001.
- [6] M. Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network, Technical Report, University of Chicago, 2001. <http://www.cs.uchicago.edu/%7Ematei/PAPERS/gnutella-rc.pdf>
- [7] S. Ross. *Applied Probability Models with Optimization Applications*, Holden-Day, San Francisco, 1970.