

Fast Learning of Optimal Connections in a Peer-to-Peer Network

Aaron Harwood, Egemen Tanin, and Minh Tri Truong
Department of Computer Science and Software Engineering
University of Melbourne, Victoria, 3010, Australia
<http://www.cs.mu.oz.au/p2p>

Abstract—Peer-to-Peer (P2P) protocol design is widely undertaken with the assumptions that peers and network connections are homogeneous resources. In practice this assumption is untrue. Furthermore, while there are some P2P networks that provide asymptotic cost optimal topology or routing, very few existing protocols combine topology optimization and resource optimization, the resulting performance is resource oblivious. We propose a class of traffic based learning protocols, called *FLOC* protocols, that learn new connections between neighbors of neighbors. For an average routing table size of d and n peers, our protocol is seen to quickly converge from an inefficient network to an asymptotic cost optimal diameter of $O(\log_d n)$, and simultaneously reduce network delay by approximately 50% in highly heterogeneous networks. We provide extensive simulation results to show *FLOC*'s behavior.

I. INTRODUCTION

The performance of peer-to-peer [3] P2P protocols is an interesting research topic that requires studying both topology and optimization of resource usage. Results have led to various P2P protocols, with varying performance characteristics, e.g., *Chord* [22], *Tapestry* [24], *CAN* [17], and *Pastry* [20]. A widely held assumption is that the peers and network connections are homogeneous. In practice, peers are not homogeneous because, for instance, some peers are large multi-processor servers connected to the Internet via fast Ethernet and some peers are small desktop PCs connected to the Internet via slow dialup lines. In this paper we propose a traffic based learning¹ protocol that rapidly converges to asymptotic cost optimality and simultaneously takes into account the heterogeneity of peers. Also, without modification, our protocol can augment the wide variety of existing P2P protocols, with surprisingly little effort.

In general, peers follow a given protocol to form an interconnection network (e.g., using TCP/IP connections) and the network exhibits topological properties that describe its overall expected performance. P2P networks are dynamic and actual connections, along with resource capability, will vary according to when individual peers enter and leave the network. We consider P2P protocols that converge to a static network, i.e., if no peers are entering or leaving then, in a finite time, the P2P network is described by a directed, simple graph, $G = (V, E)$, where V is the set of peers and E is the set of connections between peers. Peers may make a persistent TCP connection to each peer in their connection table (in which

case the graph model is undirected) or make a connection only when messages need to be sent (in which case the graph model is directed).

Typically, a logical space of locations, $\mathcal{L} \subseteq \mathbb{Z}$, is distributed over the peers such that each peer is responsible for some subset. A hash function, $H : \mathcal{K} \rightarrow \mathcal{L}$, is a many-to-one mapping of *keys* (arbitrary byte strings) to locations. A good hash function will appear with high probability to be a one-to-one mapping. A P2P protocol specifies how peers must interact in order to maintain this space and in order to locate the peer responsible for any particular key. In our work, we simply use $u, v \in V$ to represent a peer's Internet address. Usually, u is hashed into the space, giving $H(u)$, to associate the peer with some subset of the space. Requests to access a $k \in \mathcal{K}$ are routed from the requesting or *source* peer, say u , to the *destination* peer, say v , which is responsible for maintaining $H(k)$. A path from u to v in G is a (logically) shortest sequence of connections from u to v and the path length is the number of such connections or hops. While the length of the path is minimum it may not be the path with minimum delay because delay is a function of resource capacity. The maximum of path lengths between all pairs of peers, i.e., for all possible requests, is called the *diameter* of the network and gives rise to a theoretical asymptotic behavior for request delay, e.g., $O(\log n)$ (*Chord*, *Tapestry*, and *Pastry*) or $O(n^{1-\epsilon})$ (*CAN*) where n is the number of peers and ϵ is some small positive constant. It is not hard to have a sub-logarithmic diameter, $O(\frac{\log n}{\log \log n})$, with logarithmic routing table size, $O(\log n)$, which is superior to *Chord* and others.

A routing table is maintained by each peer and for each arriving request the table is consulted for the next peer to which the request should be forwarded. The size of the table is important for performance because it dictates how much overhead is required to maintain the diameter and to maintain the network connections in the presence of transient events, such as peers leaving and joining the network. Fundamentally, if the routing table is small then the diameter is large and the overhead for maintenance is small; and vice versa. For a peer, u , the size of its routing table, $d^+(u)$, is called the outdegree of u . The size of the set, $\{v : (v, u) \in E\}$, is the indegree, $d^-(u)$, of u . The degree, $d(u) = d^+(u) + d^-(u)$, is the total number of connections that a peer (and programatically the host upon which the peer is executing) must support at any one time. For generality, a directed graph is converted to an

¹By learning we mean caching with localized decision making.

undirected graph with the addition of at most $|E|$ directed edges. Then, for a degree of d and a diameter of k , the cost, $c = dk$, is a well-known performance indicator [8]. The properties are related via Moores Bound [10], in terms of the maximum number of peers, $n(k, d)$, that can be contained in a network of diameter k and degree d :

$$n(k, d) \leq \frac{d(d-1)^k - 2}{d-2} = O(d^k).$$

A network is said to be *asymptotically cost optimal* (a.c.o.) if its cost c is a constant factor from the minimum cost. It follows that with $d > 2$ and constant k :

$$c > d \log_{d-1} \left(n - \frac{2}{d}(n-1) \right) = \Omega(d \log_d n)$$

Extensive network theoretic analysis of P2P networks is provided by Loguinov, Kumar et. al. [12] and independently by others [13]. They also consider *clustering coefficient*, *expansion*, *bi-section width*, and *path overlap*, concluding that the *de Bruijn network* [4], [2], [9], [19] has properties that are superior to the topology of both CAN and Chord protocols. The de Bruijn graph is used in the *D2B* content-addressable network [5] and in *Koorde* [11]. The de Bruijn network is a.c.o. and the CAN and Chord networks are not.

There are recently well established approaches to constructing a.c.o. (or near) P2P networks, mostly using protocols that are based on randomization. The Symphony protocol [14] starts with all peers in a simple cycle and then builds random connections with a probability distribution that spreads the connections over the peers. Using a constant degree d , Symphony routes with an $O(\frac{1}{d} \log^2 n)$ average number of hops, which is sub-optimal but quite efficient. More recently, Manku et. al. [15] provide rigorous analysis of an elegant algorithm, called NoN-GREEDY, that uses a 2-hop lookahead (i.e., using the neighbor of a neighbor) to gain asymptotically optimal routing in small world networks. Their technique shows how a sub-optimal lookup algorithm that executes over an optimal network, e.g., a Randomized-Chord [23] or a skip-graph [1], can be modified to an optimal one. They do not learn new connections and so the lookahead is required for each request routed.

The Randomized-Chord is based on a class of randomized algorithms that use *lookup-parasitic random sampling* of destinations. All nodes along a request's path sample the destination of the request (e.g., using ping) and modify their routing tables appropriately. In the case of Chord, fingers are modified to point to peers that are closer to the target in terms of network delay. This is the only other significant traffic based approach that we know, that addresses resource heterogeneity.

Heterogeneity is important to consider. The study by Saroiu, Gummadi et. al. [21] gives detailed measurements of peer characteristics in the Napster and Gnutella [7], [16] networks. They built crawlers to automatically sample peer characteristics in live peer networks. Disappointingly, but not surprisingly, the results show that the behavior of users in a peer system may be categorized as client-like and server-like. Approximately 26% of Gnutella users share no data, 7% of

Gnutella users offer more files than all of the other users combined and on average 60-80% of Napster users share 80-100% of the files. In general, about 22% of the participating peers have upstream (from peer) bottleneck bandwidths of 100Kbps or less, which makes them unsuitable to provide content and data services. The median session (connected) time is approximately 60 minutes, corresponding to be the time taken to connect and download a small number of files. Further studies of P2P networks were reported in [6], [18].

A. Our contribution

We propose a class of P2P protocols, called *FLOC* (Fast Learning Of Connections) protocols, that learn new connections in a P2P network based on the flow of requests. Unlike the lookup-parasitic random sampling technique, we do not require all peers along a lookup path to sample the destination and we do not require an existing P2P protocol. Rather, we start with an extremely simple ring network and then show how new connections can be quickly formed, based on local decisions between neighbors, and converge to a.c.o. network. Unlike the NoN-GREEDY technique, our protocol forms new connections and discards unused connections. Thus, our work is unique and compares favorably to existing protocols. We can also cope with the heterogeneity in a P2P network.

B. Organization of the paper

In Section II we define the *FLOC* protocols. In Section III we provide simulation results. In Section IV we make some concluding remarks.

Throughout the paper we assume that our hash function H is perfect (no collisions), in the sense that it does not map different keys onto the same location. Uniformly at random is abbreviated as u.a.r. We also use $[x]_n$ to mean $x \bmod n$.

II. CONNECTION LEARNING PROTOCOL

In this section we first present a basic P2P cycle network and a greedy routing algorithm. We then describe a homogeneous, *FLOC₀*, protocol and a heterogeneous *FLOC₁* protocol.

A. Cycle network

Consider a relatively simple but largely inefficient P2P network, based on a cycle, C , embedded on \mathcal{L} . Let c_i refer to the unique peer u for which $H(u) = c_i$ and $C = \{c_0, c_1, \dots, c_{n-1}\}$ be the ordered locations, $c_i < c_{i+1}$ ($i < n-1$), of peers in the cycle at some time instant. Then peer c_i is connected to peers $\{c_{[i+1]_n}, c_{[i-1]_n}\}$. The degree of C is 4 (two outgoing and two incoming connections for each peer).

Note that the protocol for constructing C , as peers join and leave the network, is particularly simple. It is essentially identical to the Chord protocol, with the important exception that only successors and predecessors need be maintained. The successor of c_i is $c_{[i+1]_n}$ and the predecessor is $c_{[i-1]_n}$.

The fundamental operation carried out by a P2P network is to route from any peer, u , to the peer, v , that is responsible for the element x . Let peer c_i be responsible for x if for all other $j \neq i$ there is no c_j that is nearer in \mathcal{L} to x than c_i .

Tie-breakers are given to c_i if $i < j$. Consider the general recursive greedy function $v = \text{Route}(u, x)$.

$\text{Route}(u, x)$ returns v that is closest to x , starting at u .

- 1) Let $\Gamma_u = \{u_1, u_2, \dots, u_d\} \subset C$ be the neighbors of node u (represented by their locations in \mathcal{L}).
- 2) Let i be such that $u_i \in \Gamma_u$ is nearest to x . (Tie-breakers based on whether $u_i < u_j$ or not.)
- 3) If u is nearer to x than u_i then $v := u$ and stop, else $v := \text{Route}(u_i, x)$.

In a P2P network, step 3 is carried out by routing the request to peer u_i . The algorithm converges to the peer that is nearest to x but is shown to be sub-optimal, even in optimal networks (because it is greedy). In this case, the diameter of the cycle is $\lfloor \frac{n}{2} \rfloor$ and so the cost is $O(n)$, which is sub-optimal. We will use the greedy algorithm and connections will be formed by augmenting it.

B. Learning connections in the cycle

We consider how to assign more connections in the C network so that the number of steps required by $\text{Route}(u, x)$ to find v is small, while also ensuring that each peer is not overloaded with a large routing table. Existing connections in C are called *fundamental* and they are not changed. Let C^* be the network with added connections. Added connections are *transient*, they may be deleted at any time (e.g., if a peer leaves the network). In the absence of transient connections, the performance of C^* degrades to the performance of C ; before any learning takes place $C^* = C$.

Generally, our class of learning protocols, which we refer to as *FLOC* protocols, are traffic based: we form new connections between peers through which requests are seen to be regularly routed. All peers in the network apply *FLOC* locally and obviously to all other peers, which conforms to the P2P paradigm. The homogeneous-network protocol, $FLOC_0$, takes two parameters, τ_{in} and τ_{out} , that represent *time-in* and *time-out* parameters respectively; $FLOC_0$ is followed by every peer.

$FLOC_0(\tau_{in}, \tau_{out})$ protocol.

- 1) Peer u enters the network. Fundamental connections to successor and predecessor in the cycle are made and there are no existing transient connections from or to u .
- 2) Let a request that arrives at peer u , from neighboring peer v and going to $v' \neq u$, be called a (v, v') request. Note that v and v' are not necessarily the source and destination of the request. If the number of (v, v') requests observed by u in a time interval τ_{in} is greater than 1 then instruct v to form a transient edge from v to v' .
- 3) If peer u is instructed to form a transient edge to u' then it does so: $d^+(u)$ and $d^-(u')$ both increase by one. Each transient edge is pruned if it times out in τ_{out} time, i.e., if there is a period of time τ_{out} for which no request traverses the transient edge.

Fig. 1 illustrates the $FLOC_0$ protocol. Note that the existing connections may themselves be transient connections that were formed earlier. The protocol allows new connections to be formed as needed and prunes transient connections that are not used; clearly, there are many factors that determine when and where this occurs.

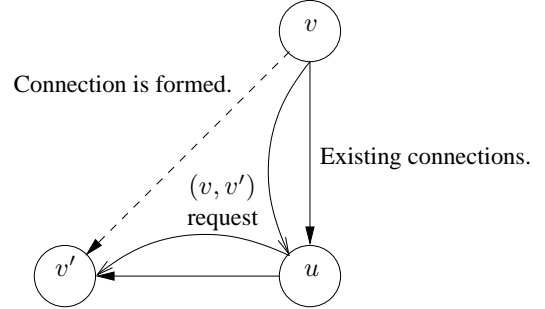


Fig. 1. Forming a connection from v to v' .

The rate of (v, v') -requests determines whether a new connection is formed or not. Similarly the rate of requests across a connection determines if the connection is pruned or not. Assume that requests are generated at an average rate of λ_R at each peer and that the destination of each request is distributed u.a.r. over the peers. Then the total request rate is $n\lambda_R$. On average each request traverses k_{avg} connections before reaching its destination. If each connection is equally likely to be used then each connection receives requests at a rate

$$\lambda_{connection} = \frac{n\lambda_R k_{avg}}{2n + \Pi},$$

where Π is the number of transient edges and by definition $|E| = 2n$ fundamental edges. The rate of requests that flow from v to u and then from u to v' is then

$$\lambda_{(v,v')} = \frac{\lambda_{connection} - \lambda_R}{d^+(u)}.$$

For a cycle with $\Pi = 0$, $d^+(u) = 2$ and $k_{avg} > 4$, then new connections are formed (i.e., Π increases) when

$$\tau_{in} \geq \frac{8}{\lambda_R(k_{avg} - 2)} = \Omega\left(\frac{1}{\lambda_R k_{avg}}\right)$$

Since $\lambda_{connection}$ decreases as Π increases and as k_{avg} decreases, over time Π reaches a steady state. We show this steady state by simulation in Section III. The τ_{out} parameter is required to prune connections that were needed at early stages of learning. Without pruning, useless edges would remain and cause the value of d^+ to be unnecessarily large.

The heterogeneous-network protocol, $FLOC_1$, requires an additional factor which we call *fudge*, f . The $FLOC_1(\tau_{in}, \tau_{out}, f)$ protocol is an extension to the $FLOC_0$ protocol. The difference is that when a new connection is to be formed, e.g., from v to peer v' , then instead of blindly forming the connection to peer v' , a connection is formed either to v' or to a neighbor of v' (only considering cycle neighbors) that is no more than f hops from v' . The actual

peer to connect to is based on the peer with minimum network delay. If a connection is already established from v to a chosen neighboring peer of v' then the new connection is established to v' regardless of its delay (so that the learning is progressive). Neighbors of v' which have equal delay are chosen randomly. The $FLOC_0$ is a special case of $FLOC_1$ with $f = 0$. In a heterogeneous network, $FLOC_1$ attempts to form connections only to peers that offer a small network delay.

III. SIMULATION AND RESULTS

In this section we describe our simulation parameters and then provide a number of simulation results that show the behavior of our $FLOC$ protocols in both the homogeneous (or topological) case and the heterogeneous case. We also show the effect of dynamicity on our protocol.

A. Parameters

A population, P , of peers is defined at time 0. At any time $t = 0, 1, 2, \dots$, the subset $P_t \subset P$ contains those peers that are in the P2P network. All other peers are considered to be out. The minimum time step is Δt and all time intervals use integral multiples of this time step. An interval of time, $[0, \Delta t, 2\Delta t, \dots, (T-1)\Delta t]$, is used to define the random arrival and departure processes. The number of arrivals in an interval is $\lambda_N T \Delta t$ where λ_N is the mean arrival rate. The number of departures in an interval is $\mu_N T \Delta t$ where μ_N is the mean departure rate. The arrival process is defined by picking arrival points u.a.r. in the interval and bucketing them in T different buckets. Peers arrive at the end of a time step, and any number of peers may arrive concurrently. A peer arriving at time t changes from being out to being in P_t ; the arriving peer carries with it no information. Similarly, the departure process is defined. Departures also take place at the end of each time step. A peer that departs at time t is chosen u.a.r. from the set P_t . Departures are carried out before arrivals so that arriving peers do not instantly depart. If the size of P_t decreases to 0 then excess departures are ignored. If $\lambda_N = \mu_N$ then the size of P_t roughly fluctuates around a mean constant (ignoring the loss of departures if 0 size is reached). If $\lambda_N > \mu_N$ then P_t grows steadily and if $\lambda_N < \mu_N$ then P_t shrinks steadily.

Without loss in generality, arrivals and departures are considered to take place instantaneously. This includes the connections and disconnections of fundamental connections. Generality is maintained if the arrival or departure is seen as offset in time, i.e., an arriving peer actually started arriving some time before its actual arrival time, whereby the appropriate connections are made, and similarly for departing peers. An arriving peer picks a location u.a.r. from the set $\{0, 1, \dots, L-1\}$ and continues to do so until it finds a location that is not already occupied by an existing peer.

Requests arrive randomly at each node and the number of requests that arrive at each node in an interval is $\lambda_R T \Delta t$ where λ_R is the mean request arrival rate per node such that requests arrive with a total mean rate of $|P_t| \lambda_R$. The source and destination of a request are chosen u.a.r. When a request arrives at a peer, the next peer is instantly computed

and the request is sent to that peer. A request traversing a connection from u to v takes $\max\{\beta_u, \beta_v\} \Delta t$ time to traverse the connection where β_u is the ‘‘bottleneck’’ factor at peer u caused by its network connection. A large factor implies a small bit-rate connection. We consider two possible bottleneck factors, $\{b_1, b_2\}$, small delay and large delay bottlenecks, respectively. With probability p , u has $\beta_u = b_2$ and b_1 otherwise.

In the simulations that follow, unless otherwise stated, we have $\Delta t = 0.1$, $L = 10^9$, $|P_t| = 1000$, $\lambda_N = \mu_N = 0$, $\lambda_R = 0.001$, $\beta_u = 1$ for all peers, $p = 0$, $b_1 = 1$, and we simulate for a total time of $12T\Delta t$ where $T = 6000$. This corresponds to a static homogeneous network of 1000 peers, where each peer makes about 3.6 requests per hour, each request takes 0.1 seconds to traverse a connection and the simulated time is 2 hours. Unless otherwise stated, the network starts with no transient connections. We change a number of these parameters to show how our $FLOC$ protocol performs. Parameters for the $FLOC$ protocol are explicitly stated. Each measurement point is an average of 8 independent trials.

B. Transient connection formation

Fig. 2 shows how the formation of transient connections reaches a steady state. In this case, the parameters are as just given in the previous section. We set τ_{in} 16.6, 33.3, and 66.7 minutes respectively. In all cases $\tau_{out} = t_{in}$. The mean d^+ begins at 2, reaches a breakpoint and then recedes to a steady state. This is because no new connections are being formed and useless connections are being pruned. In this case, the network of 1000 peers reaches a steady average d^+ of 5 in about 40 minutes.

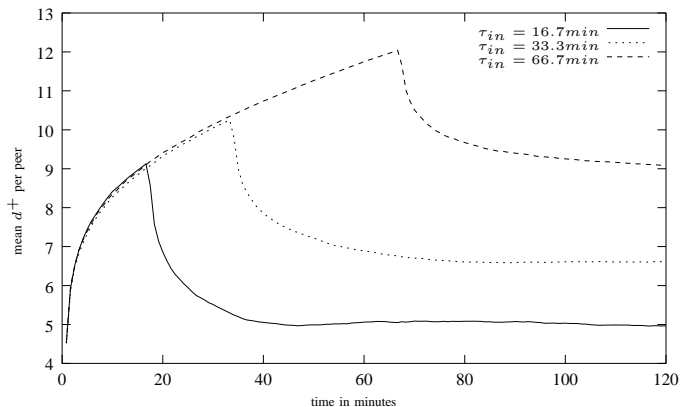


Fig. 2. The steady state out degree for various τ_{in} and 1000 peers.

Fig. 3 shows the same simulation, when $|P_t| = 100$, for comparison. Note that the learning takes place at the same rate independently of the size of the network. The difference is in the steady state degree reached which is larger for larger networks.

C. Scalability

Fig. 4 shows the average number of hops per request and the average out degree (on the same plot) as the network scales

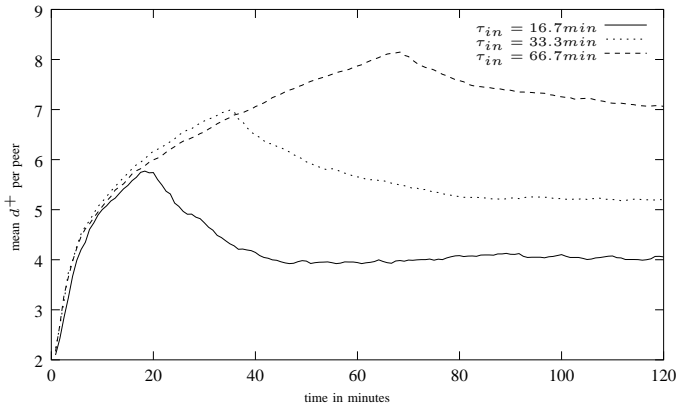


Fig. 3. The steady state out degree for various τ_{in} and 100 peers.

from 100 to almost 10000 peers. Each run is a static network of a given size (peers do not dynamically join). We held $\tau_{in} = \tau_{out} = 16.7min$ and considered roughly the last half of all completed requests (to avoid the learning period).

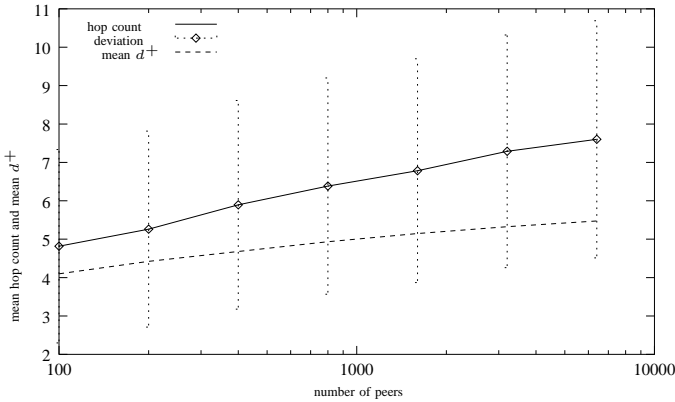


Fig. 4. Mean hop count and average out degree on the same plot versus number of peers.

D. Optimality of transient connections

Fig. 5 shows the average number of hops per request and the average out degree (on the same plot) for a static network of 1000 peers and increasing values of τ_{in} . The asymptotic minimum dashed line shows $\log_{d^+} 1000$ and the mean hop count is converging with the asymptotic minimum. Remarkably, for $\tau_{in} = 6400$ seconds, the mean d^+ rises to slightly over 10 and the mean hop count drops to about 2.5.

E. Heterogeneous network

For a heterogeneous network, we used $p = 0.5$, $b_1 = 1$, and $b_2 = 10$. This means that roughly half of the peers have a network delay (e.g., their network connection) that is 10 times slower than the others. The delay parameters are in units of Δt . Fig. 6 shows the average and standard deviation of delays incurred by requests as a function of network size. As expected, the average delay is $\frac{\Delta t \cdot (3b_2 + b_1)}{4}$ times the average hops (as given in Fig. 4). The deviation in delay is large, but

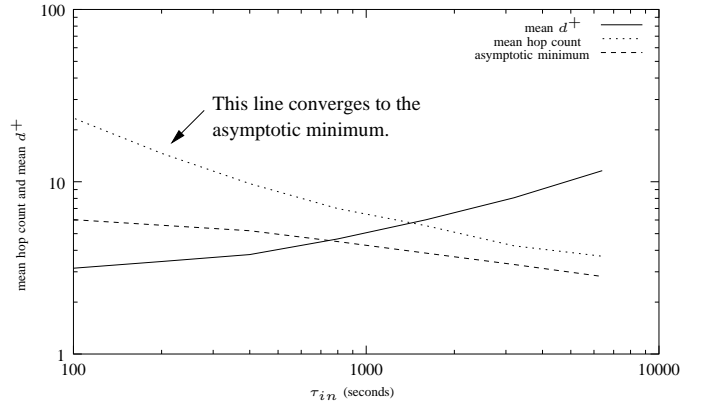


Fig. 5. Mean hop count and average out degree on the same plot versus τ_{in} .

relatively constant and the growth in delay is optimal since it is proportional to the average number of hops.

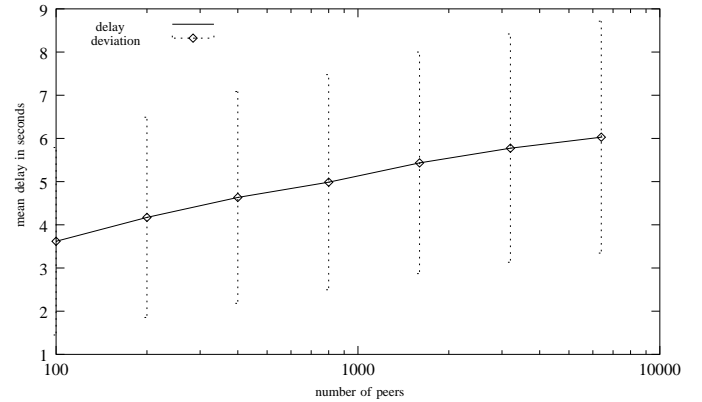


Fig. 6. Mean delay versus number of peers.

However the delay is not minimum because so far we have considered when the fudge factor $f = 0$. Fig. 7 shows how the delay can be reduced by as much as 50% with an increase in f . Furthermore, increasing f beyond 2 does not yield significant additional reduction. From other results, not shown in this paper for brevity, the decrease in average delay is not accompanied by any noticeable increase in average hop length, though a larger f is required when p approaches to one. Thus the *FLOC* protocol does simultaneously take into account the network resources while forming an a.c.o. network.

F. Effects of dynamicity

Fig. 8 shows the performance of learning in a dynamic environment. We set $\lambda_N = \mu_N = \lambda$ such that the value of λ provided a network of constant size with peer average connection time being one hour. From Fig. 8, while dynamicity dampens the initial overshoot of new connections formed, when the breakpoint time is near the average connection time, the dynamicity does not significantly impact the steady state degree. Our auxiliary observations show that the mean hop count is also not significantly affected. Actually, it is widely known that dynamicity can induce randomness and that randomness can have a positive affect on optimality.

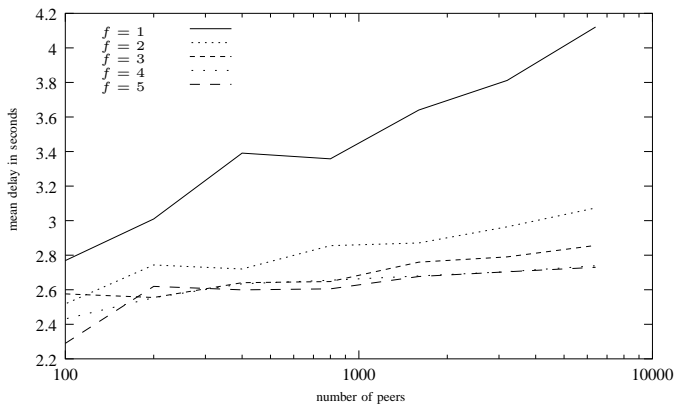


Fig. 7. Mean delay versus number of peers.

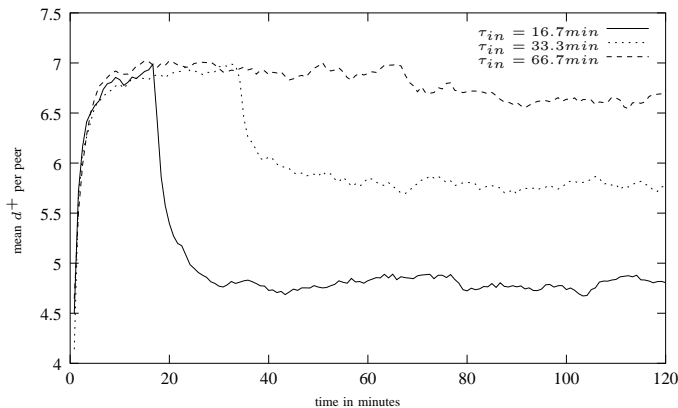


Fig. 8. The steady state out degree for various τ_{in} and 1000 peers, when peers stay for an average 60 minutes at a time.

IV. CONCLUSION

We provided the *FLOC* protocols for fast learning of optimal connections in a P2P network. The homogeneous-network *FLOC*₀ protocol learns neighbors of neighbors, forming new connections, in any P2P network. Unused connections (apart from a fundamental cycle) are pruned. This learning process is shown, by simulation, to rapidly converge to an a.c.o. network. The heterogeneous-network *FLOC*₁ protocol reduces network delay by learning new connections in the vicinity of the neighbor's neighbor, i.e., to a peer that provides a low delay. This is shown to reduce network delay by 50% when roughly half the network consists of slow peers and the remaining are fast peers.

Compared to other recent proposals in the literature, our protocol is unique because it combines optimal topological behavior, simultaneously reduces network delay, and is based on a simple cycle which is practical to implement. The dynamic learning process is significantly challenging to study and we are currently working on further analyzing our *FLOC* protocol behavior.

REFERENCES

[1] J. Aspnes and G. Shah. Skip graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393. Society for Industrial and Applied Mathematics, 2003.

[2] J.-C. Bermond and C. Peyrat. De Bruijn and Kautz networks: a competitor for the hypercube. In J. P. Verjus and F. Andr'e, editors, *Hypercube and Distributed Computers*, pages 279–294. North-Holland, 1989.

[3] D. Clark. Face-to-face with peer-to-peer networking. *IEEE Computer*, 34:18–21, January 2001.

[4] N. G. de Bruijn. A combinatorial problem. *Koninklijke Nederlands: Academie Van Wetenschappen*, 49(20):758–764, 1946.

[5] P. Fraignaud and P. Gauron. Brief announcement: an overview of the content-addressable network D2B. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing*, pages 151–151. ACM Press, 2003.

[6] N. S. Good and A. Krekelberg. Usability and privacy: a study of Kazaa P2P file-sharing. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 137–144. ACM Press, 2003.

[7] P. K. Gummadi, S. Saroiu, and S. D. Gribble. A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems. *SIGCOMM Comput. Commun. Rev.*, 32(1):82–82, 2002.

[8] A. Harwood. *High Performance Interconnection Networks*. PhD thesis, Computer Science, Griffith University, 2002.

[9] M. Heydemann, J. Opatrný, and D. Sotteau. Broadcasting and spanning trees in de Bruijn and Kautz networks. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 38:297–317, 1992.

[10] A. J. Hoffman and R. Singleton. On Moore graphs with diameters 2 and 3. *IBM J. Res. Develop.*, 4:497–504, 1960.

[11] M. Kaashoek and D. Karger. Koode: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd IPTPS*, 2003.

[12] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications*, pages 395–406. ACM Press, 2003.

[13] G. S. Manku. Routing networks for distributed hash tables. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing*, pages 133–142. ACM Press, 2003.

[14] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003.

[15] G. S. Manku, M. Naor, and U. Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks. Accepted for publication in *Proceedings of the 36th ACM Symposium on Theory of Computing*, 2004.

[16] M. K. Ramanathan, V. Kalogeraki, and J. Pruyne. Finding good peers in peer-to-peer networks. Technical Report HPL-2001-271, Hewlett-Packard Labs, 2001.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM*, pages 161–172, San Diego, CA, August 2001. ACM Press.

[18] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the Gnutella network. *IEEE Internet Computing*, pages 50–57, Jan. 2002.

[19] J. Rol, P. Tvrdik, J. Trdlicka, and I. Vrto. Bisecting de Bruijn and Kautz graphs. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 85:87–97, 1998.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the ACM Middleware*, pages 329–350, Heidelberg, Germany, November 2001. ACM Press.

[21] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, January 2002.

[22] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networks*, 11(1):17–32, Feb. 2003.

[23] H. Zhang, A. Goel, and R. Govindan. Incrementally improving lookup latency in distributed hash table systems. In *Proceedings of the 2003 ACM SIGMETRICS international conference on measurement and modeling of computer systems*, pages 114–125. ACM Press, 2003.

[24] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB-CSD-01-1141, Department of Computer Science, University of California, Berkeley, April 2001.