Computer Networks 54 (2010) 1440–1459

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

HPC5: An efficient topology generation mechanism for Gnutella networks

Joydeep Chandra *, Santosh Kumar Shaw, Niloy Ganguly

Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur 721 602, India

ARTICLE INFO

Article history:
Received 15 April 2009
Received in revised form 10 November 2009
Accepted 24 November 2009
Available online 11 December 2009
Responsible Editor: I.F. Akyildiz

Keywords:
Peer-to-peer networks
Gnutella
Topology generation

ABSTRACT

In this paper, we propose a completely distributed topology generation mechanism named HPC5 for Gnutella network. A Gnutella topology will be efficient and scalable if it generates less number of redundant queries. This can be achieved if it consists of a fewer number of short length cycles. Based on this principle, our protocol directs each peer to select neighbors in such a way that any cyclic path present in the overlay network will not generate any redundant query. We show that our approach can be deployed into the existing Gnutella network without disturbing any of its parameters. We also show that the probability of inconsistencies arising during topology generation, using our mechanism, which may lead to the formation of a small number of short length cycles is very low. However, we have also proposed an inconsistency handling protocol that detects such short length cycles and effectively removes them. We implemented a Gnutella prototype to compare and validate the efficiency of our protocol over existing Gnutella. Simulation results indicate that our mechanism outperforms existing Gnutella in terms of network coverage (the number of unique peers explored during query propagation in limited flooding) and message complexity. Structural analysis indicates that the proposed enhancement conserves the robustness of existing Gnutella network. Finally, we draw comparisons of the proposed protocol with a state-of-the-art topology optimization protocol named Distributed Cycle Minimization Protocol (DCMP); the simulation results indicate that HPC5 outperforms DCMP in terms of message overhead and network coverage.

1. Introduction

Peer-to-peer (p2p) network is an overlay network, useful for many purposes like file-sharing, distributed computation, etc. Depending upon the topology formation, p2p networks are broadly classified as structured and unstructured. An unstructured p2p network is formed when the overlay links are established arbitrarily. Decentralized (fully distributed control), unstructured p2p networks (Gnutella, FastTrack etc.) are the most popular file-sharing overlay networks. The absence of a structure and central control makes such systems much more robust and highly self-healing compared to the structured systems [11,17]. But the main problem of these kinds of networks is scalability due to the generation of a large number of redundant messages during query search. Consequently, as these networks are becoming more popular, the quality of service is degrading rapidly [8,12].

To make the network scalable, *Gnutella* [1–3] is continuously upgrading it's features and introducing new concepts. All these improvements can be categorized into two broad areas: improvement in search techniques and modification of the topological structure of the overlay network to enhance search efficiency. In enhanced search techniques, several improvements like *Time-To-Live* (TTL), *Dynamic query*, *Query-caching* and *Query Routing Protocol* (QRP) have been introduced. One of the most significant topological modifications in unstructured network was done by inducing the concept of *super-peer* (ultra-peer) with a two-tier network topology.

* Corresponding author. Tel.: +91 9434305806.
E-mail addresses: joydeep@cse.iitkgp.ernet.in (J. Chandra), santosh.shaw@gmail.com (S.K. Shaw), niloy@cse.iitkgp.ernet.in (N. Ganguly).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.11.017

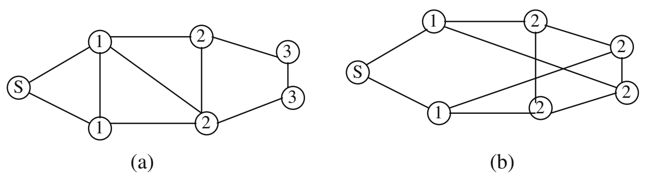


Fig. 1. Effect of topology structure on limited flood based search. The number inside the circle represents the TTL value required to reach that node from start node *S*.

The basic search mechanism adhered by Gnutella is *limited flooding* [1–3]. In flooding, a peer that searches for a file, issues a query and sends it to all of its neighbor peers. The peer that receives the query forwards it to all its neighbors except the neighbor from which it is received. By this way, a query is propagated up to a predefined number of hops (*TTL*) from the source peer. Recent versions of Gnutella uses dynamic querying protocol [5], where the *TTL* followed is generally 1 or 2 for popular searches; however, the query with *TTL*(3) (numeric value inside parenthesis represents the number of hops to search with) is initiated for rare searches.

The main goal of this paper is to improve the scalability of the Gnutella network by reducing the redundant messages. One of the ways to achieve this is to modify the overlay network, so that small size loops get eliminated from the overlay topology. The rationale behind the proposition is explained through Fig. 1. In this figure, both networks have the same number of connections. With a *TTL*(2) flooding, the network in Fig. 1a discovers four peers at the expense of seven messages, whereas the network in Fig. 1b discovers six peers without any redundant messages. This happens due to the absence of any 3-length cycle in the network of Fig. 1b. On generalizing, we can say that for a *TTL*(r) flooding, networks devoid of cycles of length less than $(2r+1)$ do not generate any redundant messages. We refer such networks which do not have any cycles up to length $(r-1)$ as cycle-r network.[1] In this paper, we propose a handshake protocol that generates a cycle-5 network topology. This topology will thus rarely produce any redundant message while performing normal *TTL*(2) search. The strength of our proposed mechanism is its simplicity and the ease of deployment over existing Gnutella networks along with its power to generate topologies having high efficiency in terms of message complexity and network coverage [18].

1.1. Related work

P2P traffic has grown immensely in the recent years and they constitute a large portion of the total Internet traffic. Hence developing suitable mechanisms so as to control excessive traffic, thus reducing the burden on ISP's, besides maintaining an adequate search performance, has become an important research issue. We discuss certain proposed mechanisms that broadly attempt to modify the topology in unstructured p2p networks to solve the excessive traffic problem and improve search performance.

[1] A formal proof of the statement for $r=2$ is stated in Appendix B.

Liu et al. [9,10] observed that the structural mismatch between the overlay and the underlying network topology is a major cause for traffic redundancy in p2p networks. Peers forming blind overlay connection without knowledge of the underlying physical network can generate huge redundant traffic between autonomous systems. They proposed a location aware topology matching algorithm (LTM) to handle this problem. The basic objective of LTM is to detect the peers within same autonomous systems on the basis of delay incurred by the messages, and then reorganize the overlay links so that peers within the same autonomous systems are grouped together. LTM uses three fundamental operations to achieve this objective. (a) *TTL2-Detector Flooding*: Each peer floods a special message, named TTL2-detector message, periodically. The source peer of this message stamps its IP address and the time-stamp; a neighbor on receiving this message appends its IP address and the TTL1 time-stamp and forwards it to its neighbors. Peers receiving the detector message can determine the connection speeds from the message time-stamps. If a peer *P* receives multiple detector messages from same source, it initiates the second operation named, slow connection cutting. (b) *Slow Connection Cutting*: Peer *P*, on receiving duplicate detector messages from a source node determine the slowest connection link from the message time-stamps. If the slowest link is a link of node *P*, the link is disconnected. (c) *Source Peer Probing*: A peer *P* on receiving a message with $TTL = 0$ from source *S* through peer *N* probes the connection cost of the link *SP* and also obtains the connection cost of links *SN* and *NP* from the message. If link *SP* does not have the highest cost, the link, *SP*, is created and *NP* is disconnected. Although, this algorithm improves search efficiency but the traffic generated due to the detector messages is enormously high; with n peers, the traffic incurred is $O(n^2)$. A similar class of overlay topology based on distance between a node and its neighbors in the physical network structure is presented in [13].

Papadakis et al. [15] presented an algorithm to monitor the ratio of duplicate messages through each network connection. Each peer in the network monitors the number of duplicates received through each of its link, over the total number of received messages. On receiving a duplicate message through a link, a peer *P* sends a feedback to the upstream node, say *Q*, of the link. The upstream peer *Q* maintains a record of the number of duplicate messages sent by itself to the downstream node. Consequently, when the ratio exceeds a certain threshold value, the node does not forward any query through that connection. However, this mechanism suffers from certain inherent drawbacks. Although a peer can receive several duplicate messages through a link from a particular source node, but that link might be the only path to several other source nodes. Disconnecting that link reduces the network coverage of the peers. Moreover this method is not suitable in case of high peer churn where peers arrive and depart frequently.

Zhu et al. [25] very recently presented a *Distributed Cycle Minimization Protocol* (DCMP) to improve the scalability of Gnutella-like networks by reducing redundant messages. They have pointed the same concept of elimination of short-length cycles cycles. However, this is demand driven and involves a lot of control overhead. The algorithm does not preserve the Gnutella parameters (like degree distribution, average peer distance, diameter, etc.), hence robustness of the evolved network is not maintained. In our work, we take into consideration all the aspects like control overhead, network coverage, and robustness, and propose a holistic yet simple approach to topology formation. The algorithm is initiated as soon as a peer enters in the network, rather than having it demand driven. Thus the algorithm works during the bootstrap phase when the network is forming so that less overhead is involved afterwards. It further eliminates the inconsistencies that might occur during the bootstrap phase. We later compare the performance of our algorithm with DCMP, in terms of message overhead and network coverage, using simulations. It is observed that for large network size, our algorithm outperforms DCMP in both these aspects.

Yet another class of algorithm exists that works on the margins of community formation depending upon similar file interests. In these algorithms, the network topology is restricted to certain clusters, where intra-cluster traffic can be high but inter-cluster traffic very low.

Chen et al. [6] proposed a distributed technique to identify peers sharing similar file interests and form overlay connection among them. In this technique, each file f is identified by a set of attributes, say $\alpha_1, \alpha_2, \ldots, \alpha_n$, based on its specific application domain. For a pair of files, say $f_i = \alpha_1^{(i)}, \alpha_2^{(i)}, \ldots, \alpha_n^{(i)}$ and $f_j = \alpha_1^{(j)}, \alpha_2^{(j)}, \ldots, \alpha_n^{(j)}$, a feature function $F(\alpha_1^{(i)}, \alpha_1^{(j)})$ measures the correlation between the attributes $\alpha_1^{(i)}$ and $\alpha_1^{(j)}$, i.e. whether the two files f_i and f_j are related through these attributes. This correlation is used to measure a conditional probability, $Pr(f_j|f_i)$, that represents the conditional probability that a peer will request for a file f_j such that it had earlier requested for a file f_i. When a new query for a file f arrives to a peer p, it returns f if the file is present. Otherwise, it forwards the query to a new peer q that had earlier requested a file f_e that have the highest conditional probability, $Pr(f|f_e)$ among all other file requests. The idea is that, since q has requested for file f_e, the probability that it has queried for the file f and obtained it is very high. However, there are two major drawbacks of this approach; finding an appropriate feature function is difficult, and secondly a large number of file-sharing information needs to be disseminated that induces high control overhead.

1.2. Organization of the paper

A model of our network environment and basic handshake protocol is presented in Section 2. In Section 3 we have described our handshake protocol. The problems and compatibility of implementing our protocol in Gnutella networks are described in Section 4. In Section 5, we discuss the effects of inconsistency on HPC5 and state suitable algorithms to reduce its effects. We evaluate the performance and analyze the robustness of the evolved networks through simulation in Section 7. Moreover, we also draw comparisons of our algorithm with DCMP. In Section 8 we conclude and draw directions for future work.

A list of main notations that will be used throughout the paper is summarized in Table 1 for ready reference.

2. Basic system model of Gnutella 0.6

In order to carry on experiments, a basic version of *Gnutella 0.6* [1–3] has been implemented. The basic Gnutella consists of a large collection of nodes that are assigned unique identifiers and which communicate through message exchanges.

2.1. Topology

Gnutella 0.6 is a two-tier overlay network, consisting of two types of nodes: *ultra-peer* and *leaf peer* (the term peer represents both ultra and leaf-peer). An ultra-peer is connected with a limited number of other ultra-peers and leaf-peers. A leaf-peer is connected with some ultra-peers. However, there is no direct connection between any two leaf-peers in the overlay network. Yet another type of peer is called legacy-peers, which are present in ultra-peer level and do not accept any leaves. Since legacy-peers are negligible in the network (they constitute approximately 0.3% of the total peers [24]), hence we are not considering legacy-peers in our further analysis.

2.2. Basic search technique

The network follows a limited flood based query search. A query of an ultra-peer is forwarded to its leaf-peers with $TTL(0)$ and to all its ultra-neighbors with one less TTL only when $TTL > 0$. A leaf-peer does not forward a query received from an ultra-peer. On the other hand ultra-peers perform query searching on behalf of their leaf-peers. The

Table 1
Notations.

$TTL(r)$	Query search with $TTL = r$
Cycle-r network	A network which does not have any cycle up to length $(r-1)$
Cycle-3 network	Gnutella network
N	Total number of peers in the network
U	Total number of ultra-peers in the network
L	Total number of leaf-peers in the network
d_{uu}	Avg. no. of ultra-neighbors of an ultra-peer
d_{ul}	Avg. no. of leaf-neighbors of an ultra-peer
d_{lu}	Avg. no. of ultra-neighbors of a leaf-peer
H_k	Hit ratio to select kth ultra-neighbor
$\langle H \rangle$	Average hit ratio of a peer
$\langle H_{ev} \rangle$	Average evolved hit ratio of a peer
rth neighbor	A peer at a distance of r hops. All immediate neighbors are 1st neighbors, all neighbors of 1st neighbors are 2nd neighbors and so on

query of a leaf-peer is initially sent to its connected ultra-peers. All the connected ultra-peers simultaneously forward the query to their neighbor ultra-peers up to a limited number of hops. Since multiple ultra-peers are initiating flooding, a leaf-peer's query will produce more redundant messages if the distance between any two ultra-neighbors is not enough. Gnutella 0.6 incorporates *dynamic querying* [5] over limited flooding as query search technique. In dynamic querying, an ultra-peer incrementally forwards a query in three steps ($TTL(1)$, $TTL(2)$, $TTL(3)$, respectively) through each connection after measuring the responsiveness in each step. The ultra-peer can stop forwarding a query at any step if it gets sufficient number of query hits. Consequently dynamic querying uses $TTL(3)$ only for rare searches. Modern Gnutella protocol uses *QRP* technique over dynamic querying in which a leaf-peer creates a hash table of all the files it is sharing and sends that table to all the immediate ultra-neighbors. As a result, when a query reaches an ultra-peer it is forwarded to only those connected leaf-peers which would have query hits [1,2].

2.3. Basic handshake protocol

Many software applications (clients) are used to access the Gnutella network (like *Limewire, Bearshare, Gtk-gnutella*). The most popular client software, Limewire's handshake protocol is used in our simulation as a base handshake protocol. Through handshaking, a peer establishes connection with any other ultra-peer. To start a handshake protocol, a peer first collects the address of an online ultra-peer from a pool of online ultra-peers. A peer can collect the list of online peers from *GWebCache* systems [4] and/or through *pong-caching* and/or from its own hard-disk which has obtained a list of online ultra-peers in the previous run [8]. The handshake protocol is used to make new connections. A handshake consists of three groups of headers [1,2]. The steps of handshaking are elaborated next:

1. The program (peer) that initiates the connection sends the first group of headers, which tells the remote program about its features and the status to imply the type of neighbor (leaf or ultra) it wants to be.
2. The program that receives the connection responds with a second group of headers which essentially conveys the message whether it agrees to the initiator's proposal or not.
3. Finally, the initiator sends a third group of header to confirm and establish the connection.

This basic protocol is modified in this paper to overcome the problem of message overhead.

2.4. Simulated Gnutella

To generate an existing Gnutella network, we have simulated a strip down version of Gnutella 0.6 protocol which follows the parameters of Limewire [1]. Our simulated Gnutella network exhibits all features (like degree distribution, diameter, average path length between two peers, proportion of ultra-peers, etc.) exhibited by Gnutella network. These features are obtained from the snapshots collected by crawlers [3,19,22]. The properties of the Gnutella [1,22] and simulated Gnutella networks are given in the Table 2. We next highlight the functional features of our simulated Gnutella.

1. The simulator is a multi-threaded parallel execution simulator in which the Gnutella processes running at each nodes are implemented using separate threads. Each peer arrival spawns a new thread that follows the required bootstrap protocol to form neighbors.
2. The network grows from an initial set of 20 pre-existing ultra-peers that are connected in the form of an one-dimensional lattice of degree 2 and forms the initial topology. Each of these peers initially constitutes the GWebCache system.
3. As peers join, the GWebCache system is updated as some proportion of ultra-peer id's are recorded in the GWebCaches. The entries in the GWebCache system provide a gateway to the Gnutella network for new incoming nodes, as new nodes can get some initial peer addresses to connect.
4. Certain number of peers arrive in bursts (we have used a burst size of 25,000); arriving peers are randomly assigned an ultra or leaf-peer status in a ratio as stated in Table 2. These peers randomly connect to any one of the existing ultra-peers, the information of which is retrieved from a random peer in the GWebCache system.
5. Peers deploy the *pong-caching* [5] mechanism to maintain information about other known peers; peers then use a gossip based mechanism, named *ping-pong* mechanism, to obtain these cached peer addresses and subsequently connect to them until the maximum neighbor count is reached.
6. Nodes send connection requests to a random set of peers, whose addresses they have obtained using the ping-pong mechanism. Connection requests are sent sequentially, in a non-concurrent manner. We define the time difference between sending of two consecutive connection requests by a peer as a step. Thus, at each step, a peer sends a connection request to exactly one peer.

Table 2
Properties of Gnutella and simulated Gnutella network

Property	Gnutella	Simulated Gnutella
No. of peers	2000k	100k
Ultra-peer ratio	15–16% of total peers	15% of total peers
Avg. diameter of ultra-layer	6–7	4–5
Maximum connections	Ultra–ultra 32 Ultra-leaf 30 Leaf-ultra 3 (Applicable to Limewire)	Ultra–ultra 32 Ultra-leaf 30 Leaf-ultra 3
Average Connections	d_{uu}: 25–26 d_{ul}: 20–22 d_{lu}: 3–4	d_{uu}: 22–23 d_{ul}: 17–18 d_{lu}: 3

7. To study the effect of node removal on the Gnutella Network, we also implemented a node removal functionality in which a desired number of nodes is randomly removed from the network. However, since this reduces the degree of the existing peers, peers attempt to regain their lost degree by reconnecting with other peers. To simulate this effect, we assume that at each step, a peer that has lost a neighbor sends connection request to exactly one peer. If the connection request is from a valid peer, i.e. the node can be accepted as a neighbor, then the connection request is accepted. At the end of a step, all peers connect to valid peers from whom a connection request has arrived. We assume that at each step, a peer that has lost an edge connects to at least one new node; hence if it has lost $\mathscr{D}$ degrees, then it requires a maximum of $\mathscr{D}$ steps to restore its lost degree.

The peer selection strategy during bootstrapping can play an important role in the search performance. Several gossip based strategies exist for random peer sampling [7,21,23]; however, to maintain parity, we implement the pong-caching mechanism that is actually proposed in Gnutella. We ran our simulator on a server having DP Intel Dual Core Xeon 3.2 GHz 5060 processor with 4GB DDR2 RAM, and could simulate for a network size of 1 million nodes.

In the next section we discuss the proposed HPC5 mechanism in details.

3. HPC5: Handshake protocol for cycle-5 networks

Fig. 2 illustrates the proposed HPC5 graphically. In Fig. 2, peer-1 requests other online ultra-peers to be its neighbor, given that, peer-2 is already a neighbor of peer-1. In Fig. 2a and b, the possibility of the formation of triangle and quadrilateral arises if a 1st or 2nd neighbor of peer-2 is selected. However, this possibility is discarded in Fig. 2c and a cycle of length 5 is formed.

Each peer maintains a list of its 1st and 2nd neighbors, which contains only ultra-peers (because a peer only sends request to an ultra-peer to make neighbor). The 2nd ultra-neighbors of a leaf-peer thus represent the collection of neighboring ultra-peers of its adjacent ultra-peers. To keep an updated knowledge, each ultra-peer exchanges its list of 1st neighbors periodically with its neighbor ultra-peers and sends the list of 1st neighbors to its leaf-peers. To do this with minimal overhead, piggyback technique can be used in which an ultra-peer can append its neighbor list to the messages passing through it.

The two steps of modified handshake protocol (HPC5) is described below.

1. The initiator peer first sends a request to a remote ultra-peer which is not in its 1st or 2nd neighbor set. The request header contains the type of the initiator peer. The presence of remote peer in 2nd neighbor set implies the possibility of 3-length cycle. In Fig. 2, peer-1 cannot send request to peer 2 or 3, on the other hand peers 4 and 5 are eligible remote ultra-peers.
2. The recipient replies back with its list of 1st neighbors and the neighbor-hood acceptance/rejection message. If the recipient peer rejects the connection in this step, the initiator closes the connection and keeps the record of neighbors of the remote peer for future handshaking process. On acceptance of the invitation by the recipient peer, the initiator checks if at least one common peer between its 2nd neighbor set (say, A) and the 1st neighbor set of the remote peer (say, B) exists. A common ultra-peer between sets A and B indicates the possibility of 4-length cycle.
   If no common peer is present between sets A and B then the initiator sends *accept connection* to remote peer.
   Otherwise the initiator sends *reject connection* to remote peer.

HPC5 prevents the possibility of forming a cycle of length 3 or 4 and generates a cycle-5 network.

4. Hurdles in implementing the scheme

Before embedding the simple HPC5 in Gnutella network, several important questions have to be taken into consideration to assess its viability. The most important of them are listed below.

1. Is this scheme compatible with the current population of Gnutella network?
2. On average, how many trials are required to get an ultra-neighbor?
3. Is there any possibility of an inconsistency and if so, how can such inconsistency be removed?

Each of the questions are discussed one by one.

4.1. Compatibility with the current population of Gnutella

From ultra-peer point of view, the total number of ultra-leaf connections is $U \cdot d_{ul}$ and from leaf-peer point of view it is $L \cdot d_{lu}$. By equating both, we get

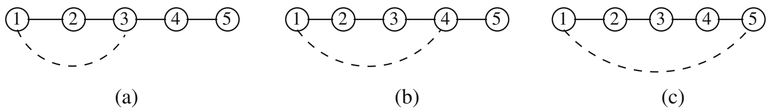


Fig. 2. Selection of neighbor by peer-1 after making peer-2 as a neighbor.

$$\begin{aligned} U \cdot d_{ul} &= L \cdot d_{lu} \\ U \cdot d_{ul} &= (N - U) \cdot d_{lu} \\ N &= U \cdot \frac{(d_{ul} + d_{lu})}{d_{lu}} \end{aligned} \tag{1}$$

Fig. 3 represents a part of an ultra-peer layer where P has immediate neighbors at level Q. Suppose, P is already connected with $(d_{uu} - 1)$ number of ultra-peers at Q level and wants to get d_{uu}th ultra-neighbor. According to HPC5, P should not connect to any ultra-peer from R or S level as its next neighbor. However, T can be a neighbor of P. Thus, we can say that if P wants to make a new ultra-neighbor then P has to exclude at most $(d_{uu}-1)+(d_{uu}-1)^2+(d_{uu}-1)^3$ number of ultra-peers from Q, R and S level, respectively. So, total $[(d_{uu}-1)+(d_{uu}-1)^2+(d_{uu}-1)^3] \approx d_{uu}^3$ number of peers cannot be considered as next neighbor(s) of P. Therefore the number of ultra-peers in the network needs to be at least

$$U \approx d_{uu}^3 \tag{2}$$

From Eqs. (1) and (2) we get

$$N \approx \frac{(d_{ul} + d_{lu}) \cdot d_{uu}^3}{d_{lu}} \tag{3}$$

Presently Gnutella network is having the population of almost 2000k of peers at any time [1]. From Eq. (3) it can be seen that for the present values of d_{uu}, d_{ul} and d_{lu} (Table 2), 120–130k peers are sufficient to implement HPC5 protocol. However, to form cycle-6 networks (HPC6) the number of peers

$$N \approx \frac{(d_{ul} + d_{lu}) \cdot d_{uu}^4}{d_{lu}}$$

required is more than 2000k. Hence the current population will not be able to support any such attempts.

4.2. Hit ratio

Hit ratio is defined as the inverse of the number of trials required to get a valid ultra-peer neighbor. As our protocol puts some constraints on neighbor selection, a contacted agreeing remote ultra-peer may not be selected as neighbor. Mathematically, on an average if a peer (say, P) is looking for its kth ultra-neighbor and only the m_kth contacted ultra-peer satisfies the constraints and becomes kth ultra-neighbor of P, then the hit ratio for kth neighbor will be $H_k = \frac{1}{m_k}$. We first make a static analysis of hit ratio, then fine tune it considering that the network is evolving.

At the time of the kth ultra-neighbor selection in HPC5, a peer (say, P) does not consider its 1^{st} $((k-1)$ ultra-peers) and 2nd $((k-1)(d_{uu}-1)$ ultra-peers) ultra-neighbors as a potential neighbor and this exclusion is locally done by checking the 1st and 2nd neighbors lists of P. The number of ultra-peers excluded (U') is then $[(k-1)+(k-1)(d_{uu}-1)]$. According to step-3 of HPC5, P cannot make neighbor from any ultra-peer of level S (3rd ultra-neighbors of P) of Fig. 3 as its neighbor which are $U'' = [(k-1)(d_{uu}-1)^2]$ in number.

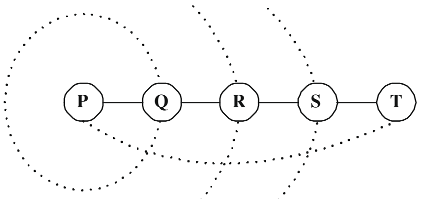


Fig. 3. A part of an ultra-peer layer, where a node represents all nodes that are present in that level. Like, Q represents all 1st neighbors of P.

Therefore, total $[U' + U'']$ number of ultra-peers are excluded. So hit ratio can be given as,

$$H_k = \frac{U - (U' + U'')}{U - U'}$$

Assuming $U' \ll U, U' \ll U''$ and $U'' \approx d_{uu}^2 \cdot (k-1)$

Therefore H_k becomes

$$H_k \approx \frac{U - d_{uu}^2 \cdot (k-1)}{U} \tag{4}$$

The upper bound of k and consequently average ultra-degree differs in leaf-peer and ultra-peer. To generalize further calculations, let m be the average ultra-degree of a peer. So, average hit ratio is

$$\begin{aligned} \langle H \rangle &= \frac{1}{m} \cdot \sum_{k=1}^{m} H_k = \frac{1}{m} \cdot \sum_{k=1}^{m} \left[1 - \frac{d_{uu}^2 \cdot (k-1)}{U} \right] \\ &= 1 - \frac{d_{uu}^2 \cdot (m-1)}{2 \cdot U} \end{aligned} \tag{5}$$

Eq. (5) shows the average hit ratio of peer joining the network when the population of ultra-peers in the network is U. It also reflects that $(1 - \langle H \rangle)$ is inversely proportional to the number of ultra-peers (U) in the complete network. Now as each node joins, the network grows. As a result the average hit ratio changes with the network growth. Therefore evolved hit ratio is the average value of all average hit ratios which are calculated at each growing stages of the network. Let U_0 and U_n be the number of ultra-peers in the initial and final networks. So, evolved hit ratio is

$$\langle H_{ev} \rangle = \frac{1}{U_n - U_0} \cdot \sum_{U_i=U_0}^{U_n} \langle H \rangle = 1 - \frac{d_{uu}^2 \cdot (m-1)}{2 \cdot (U_n - U_0)} \cdot \sum_{U_i=U_0}^{U_n} \frac{1}{U_i} \tag{6}$$

Now,

$$\sum_{U_i=U_0}^{U_n} \frac{1}{U_i} \approx \log\left(\frac{U_n}{U_0}\right)$$

Therefore, Eq. (6) becomes

$$\langle H_{ev} \rangle \approx 1 - \frac{d_{uu}^2 \cdot (m-1)}{2 \cdot (U_n - U_0)} \cdot \log(U_n/U_0) \tag{7}$$

From Eqs. (5) and (7) we get,

$$\langle H_{ev} \rangle \leqslant \langle H \rangle$$

As d and the maximum value of m are bounded, the value of $\langle H_{ev} \rangle$ increases with U. Again we have tested this phenomenon through our simulation and plotted the evolved hit ratio against the network size varying from 200k to 1000k in Fig. 4 and observed the similarity between them. The similarity is not pronounced in the beginning as the approximations made to develop Eqs. (4) and (7)

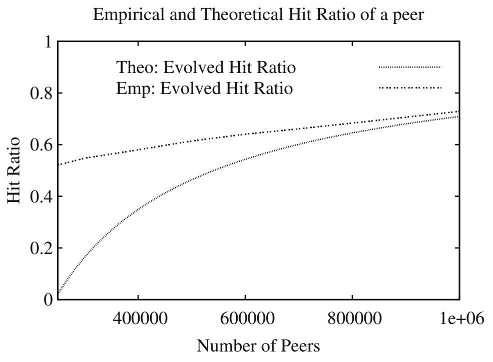


Fig. 4. Hit ratio of a peer against the number of peers in the network.

are significant in smaller networks. However, as the number of peers increases, both $\langle H\rangle$ and $\langle H_{ev}\rangle$ converges to the same value.

5. Consistency problems in HPC5

For HPC5 to work correctly, each peer must send correct information about its neighbors to other peers that want to connect. To facilitate this process, peers can periodically exchange the list of their neighbors. Periodically exchanging the list of neighbors facilitates the peers to get up-to-date information about their neighbors. However, this leads to a situation when parallel update might occur, where in between two successive updates, a peer may possibly have erroneous knowledge about it's neighbors. As a result, this inconsistency of the network leads to the presence of 3-length or 4-length cycles. Parallel update is possible when many peers enter simultaneously, like during the bootstrap phase [20], or when there is a huge failure/attack in the network whereby many nodes have lost their neighbors and would now like to quickly gain some.

In parallel update, due to inconsistency, short length cycles are formed as multiple peers in the same cycle handshake in parallel with a third common ultra-peer and become each other's neighbor. An instance of a parallel update situation is illustrated through an example (Fig. 5a) where peer-1 and peer-5 execute the following actions according to steps of HPC5 and form cycle-3.

1. Both peer-1 and peer-5 find that peer-P is a valid remote peer to contact and both send request to P.
2. Peer-P gets their request more-or-less at the same time and sends back the neighbor-hood status to them.
3. As peer-1 and peer-5 do not know each other's activity or updated status, they make P as their new neighbor, therefore a cycle-3 is formed due to this inconsistency.

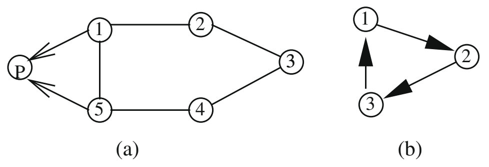


Fig. 5. A part of cycle-5 network, representing parallel update inconsistency.

Similarly smaller cycles may be created when multiple peers contact each other as a directed cycle (as in Fig. 5b) within the period of two successive updates.

In the next section, we provide a detailed analysis of the topological structure arising due to node failure; however, the analysis will be analogous for inconsistencies arising due to bursty peer arrival. We analytically derive the average number of short length cycles formed in the ultra-peer level due to inconsistencies arising from node failure and validate our results using simulation. We then propose an algorithm to handle these inconsistencies, supported by extensive simulation results to validate the efficiency of our algorithm.

5.1. Inconsistency arising in the face of failure/attack

Here, we discuss about the topological status of the network when x fraction (where $x \ll 1$) of the nodes leave the network. Our analysis deals with the topological structure at the ultra-peer level; this is because, with *QRP*, search queries are not forwarded to the leaf-peers, hence the number of short cycles at the ultra-peer level affect the traffic redundancy in the network. Thus, in all our derivations in this section, the term, peer, refers to an ultra-peer in the network. We derive the distribution of the number of 3 and 4-length cycles formed around an ultra-peer, when a small fraction, x, of the peers leave the network. Then we go on to derive the total number of such cycles formed in the network. We provide simulation results to test and compare the validity of our models. We assume that the nodes have left uniformly from different parts of the network. So each peer loses a fraction of its neighbors and in effect the average degree of a peer in the network becomes less. To maintain the degree distribution of the network, each peer contacts other remote ultra-peers to fulfill neighbor deficiency. During this process, 3-length and 4-length cycles are created temporarily due to inconsistency between two successive updates. As defined earlier, we consider the time difference between sending two consecutive connection requests, by a peer to other peers, as a *step*; we assume that at each step a peer sends a connection request to exactly one another peer, however, it receives connection requests from many peers. After the removal process, $U_{rem} = (U - xU)$ number of ultra-peers remain in the network. Since, we have assumed that the peers have been removed uniformly throughout the network, the average ultra-neighbors of an ultra-peer will get reduced to $\hat{d}_{uu} \approx d_{uu}(1-x)$. To restore the ultra-neighbor count, each ultra-peer tries to connect to new ultra-peers. We assume that, on average, each peer connects to xd_{uu} new ultra-peers so as to restore the average number of ultra-neighbors to d_{uu}. According to HPC5, a peer (say peer-1) cannot make any ultra-peers at level Q, R or S in Fig. 3 as its neighbor. With an average ultra-neighbor of $\hat{d}_{uu}$, the number of ultra-peers from which a peer has to choose a neighbor is

$$\begin{aligned} N_{rem} &= U_{rem} - [\hat{d}_{uu} + \hat{d}_{uu}(\hat{d}_{uu}-1) + \hat{d}_{uu}(\hat{d}_{uu}-1)^2], \\ &= U_{rem} - \hat{d}^3_{uu} + \hat{d}^2_{uu} - \hat{d}_{uu}. \end{aligned}$$

We define this set from which a peer can choose a neighbor as the *potential neighbor set* of the peer. Hence N_{rem} represents the size of the potential neighbor set of a peer; the potential neighbor set of every peer can be assumed to be approximately same in the limit of large N_{rem}. We next find the average number of new connections made by a node at each step, i.e. the average degree that is restored. At each step, a connection is made to peer-1 if peer-1 sends a connection request and finds a valid peer, or any valid peer sends a connection request to peer-1. The probability that peer-1 finds a valid peer at a step is $\frac{N_{rem}}{U_{rem}}$ and the probability that any valid peer connects to peer-1 is $N_{rem} \times \frac{1}{N_{rem}} \times \frac{N_{rem}}{U_{rem}} = \frac{N_{rem}}{U_{rem}}$. Thus the average number of connections formed by peer-1 at each step is $C = 2\frac{N_{rem}}{U_{rem}}$ and the average number of steps required to restore the average is approximately

$$\mathscr{S} = \frac{xd_{uu}}{C} = \frac{U_{rem}xd_{uu}}{2N_{rem}} \tag{8}$$

We now derive the distribution of the number of 3 and 4-length cycles formed around a peer at each step and then find the average number of cycles formed in the network. It is similar to finding certain prohibitive network motifs [14] that might arise due to inconsistent updates.

5.2. 3-Length cycle

A 3-length cycle is created if three peers get involved in HPC5 as in Fig. 5. The initiation of handshake protocol in different combinations among three peers may create a triangle. We attempt to model the average number of 3-length cycles formed around a peer, and the average number of such cycles in the whole network. Considering 3 peers named, peer-1, peer-5 and P, as shown in Fig. 5a, at each step, a 3-length cycle can be formed around peer-1 by various combination of connection requests made by each of the three peers. If we assume that no two peers simultaneously send connection requests to each other at any step, the various ways of forming a cycle among the three peers can be represented by connecting the peers using directed edges as shown in Fig. 5.

A directed edge from peer-1 to peer-5 implies that peer-1 has sent a connection request to peer-5. We consider a combination of directed edges as invalid (Fig. 6b), if one peer sends connection requests to two different nodes at one single step, i.e. no two directed edges go out from the same peer. Two combinations are considered as isomorphic with respect to peer-1 (Fig. 6a), if removing the labels of the other peers, (i.e. peer-5 and P in this case), makes both the graphs similar to each other. Thus a 3-length cycle can be formed due to three major cases.

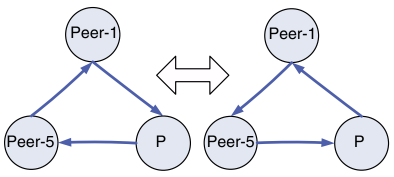
1. Each of the three peers, peer-1, peer-5 and P sends connection request to each other. Thus we have $2^3 = 8$ combinations of directed edges, out of which six possible combinations are invalid and the rest two are isomorphic cases; hence there is one possible way in which a 3-length cycle can be formed due to this case. This case is shown in Fig. 6a. Some of the invalid cases are shown in Fig. 6b.
2. There is an existing edge between peer-5 and P, and these two nodes simultaneously connect to peer-1. The number of such combinations of directed edges are 4, out of which one is invalid and two are isomorphic. Thus a cycle can be formed due to this case in two possible ways (Ref. Case-2 in Fig. 7a).
3. An edge exists between peer-1 and either P or peer-5. Thus 3-length cycles are possible due to four possible combinations of directed edges out of which one is invalid, leading to three possible ways (Ref. Case-3 in Fig. 7a).

We now find the probability that a 3-length cycle forms around peer-1 for each of the above three cases.

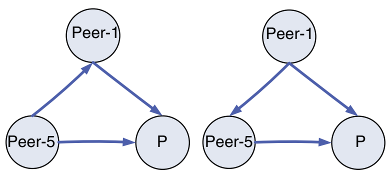
Case 1. In case 1, the peers connect to each other in a cyclic order. The probability that peer-1 connects to a random node (say peer-5), which in turn connects to another random peer (say P) that connects back to peer-1 is approximately

$$A_1^{(3)} = (N_{rem})(N_{rem} - 1)\left(\frac{1}{N_{rem}}\right)^3 \approx \frac{1}{N_{rem}} \tag{9}$$

Case 2. In case 2, the probability that two given peers, which are themselves connected, both can connect to peer-1 in any possible way is $\left(\frac{1}{N_{rem}}\right)^2$. Since each peer has an average degree of $\hat{d}_{uu}$, the average number of edges shared between N_{rem} peers is $\frac{N_{rem}\hat{d}_{uu}}{2}$, which is the possible number of neighboring peers that can simultaneously attempt to connect to peer-1 forming 3-length cycles. Assuming, the probability that more than one pair of adjacent peers connects to peer-1 is negligible, the probability that a 3-length cycle formed due to case 2 is

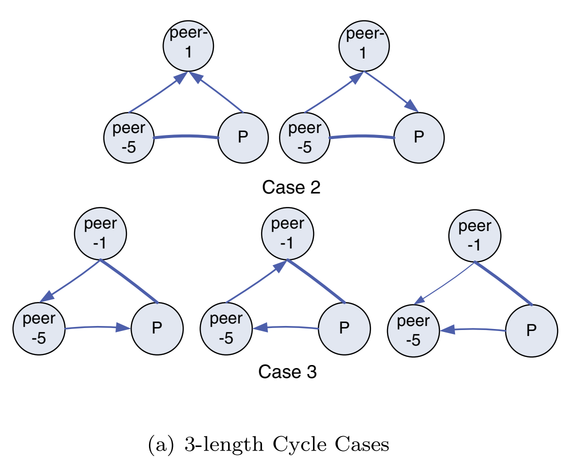


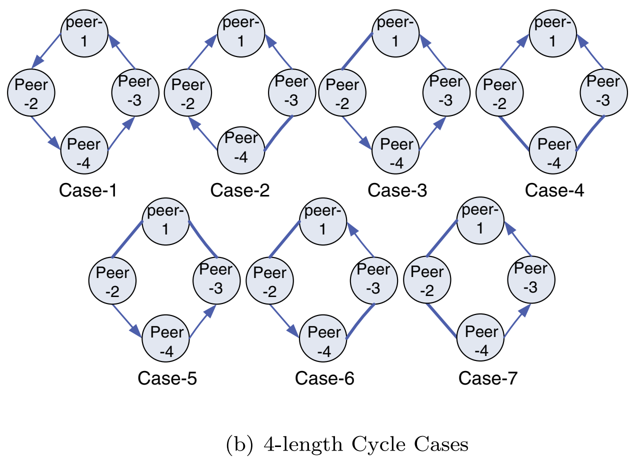
(a) Isomorphic Combinations



(b) Invalid Combinations

Fig. 6. Case 1. Graphs in (a) are isomorphic with respect to peer-1, removing the labels peer-5 and P from both these graphs makes the two graphs absolutely similar. Isomorphic combinations can be considered as single combination with respect to peer-1. This combination also represent case 1 of 3-length cycle formation. Graphs in (b) are invalid as in first case peer-5 sends connection requests to two peers simultaneously (represented by the directed edges from peer-5), whereas in the second figure, peer-1 sends two connection requests at the same time.





(a) 3-length Cycle Cases (b) 4-length Cycle Cases

Fig. 7. Possible cases of 3 and 4 length cycles.

$$A_2^{(3)} = \frac{N_{rem}\hat{d}_{uu}}{2(N_{rem})^2} \approx \frac{\hat{d}_{uu}}{2N_{rem}} \tag{10}$$

There are two possible valid combinations for formation of a cycle due to case 2, and thus each will have an occurrence probability equal to $\frac{\hat{d}_{uu}}{2N_{rem}}$.

Case 3. For each possible valid combination in case 3, the probability that a random node connects to peer-1 is $\frac{1}{N_{rem}}$ and also connects to any neighbor of peer-1 is $\frac{\hat{d}_{uu}}{N_{rem}}$. Thus the probability of forming a 3-length cycle around peer-1 due to any valid combination of case 3 is

$$A_3^{(3)} = \frac{\hat{d}_{uu}}{N_{rem}} \tag{11}$$

Considering each of the probabilities, $A_1^{(3)}$, $A_2^{(3)}$ and $A_3^{(3)}$ to be small in the limit of large N_{rem}, the average number of 3-length cycles formed around peer-1 after it has restored its average degree is

$$A^{(3)} = \mathscr{S} \cdot \left(A_1^{(3)} + 2A_2^{(3)} + 3A_3^{(3)}\right) \tag{12}$$

and the total number of 3-length cycles in the whole network is given by,

$$L_3 = \frac{1}{3} \cdot A^{(3)} \cdot U_{rem} \tag{13}$$

5.3. 4-Length cycle

Similar to the case of 3-length cycle, formation of a 4-length cycle around a peer involves connections among 4 peers, namely, peer-1, peer-2, peer-3, and peer-4. The definitions for invalid combinations and isomorphic combinations remain same as defined previously. We enumerate the possible ways in which a 4-length cycle can be formed around peer-1. For each of these cases, we represent one of the possibilities in Fig. 7b.

1. Each of the four peers have no connection among themselves and randomly select each other so as to form a cycle. Similar to the case of 3-length cycles, we assume that the probability of two peers sending request to each other in the same step is zero, thus in this case, a 4-length cycle can be represented by various combinations of the four directed edges among these peers. Out of the 16 possible combinations, 14 combinations are invalid, and two combinations are isomorphic to each other with respect to peer-1.
2. An edge exists between peer-3 and peer-4. 4-Length cycles will be formed if one of these peers connects to peer-1 and another peer connects to a peer, say peer-2, that in turn connects to peer-1. However, there can be several possible combinations by which these connections can be made. The number of such possible combinations are $2^3 = 8$, out of which four are invalid. Thus 4-length cycles can be formed due to this case in four possible ways.
3. There is an existing edge between peer-1 and any of the other peers, say peer-2. Similar to the previous case, the number of such possible combinations are $2^3 = 8$, out of which four are valid, leading to four possible ways of forming a 4-length cycle.
4. Two edges are existing among three peers and none of the edge belongs to peer-1, i.e the peers – peer-2, peer-3 and peer-4 – are already connected and nodes peer-2 and peer-4 attempt connecting to peer-1. The number of such possible combinations are $2^2 = 4$, out of which one is invalid and two are isomorphic. Thus a 4-length cycle can be formed due to this case in two possible ways.
5. Two edges are existing, both of which are from peer-1 that connects to peers, say peer-2 and peer-3. A 4-length cycle will be formed if both peer-2 and peer-3 connect to the fourth peer, i.e. peer-4 in any possible combination of directed edges. Similar to the previous case, the number of such possible combinations are $2^2 = 4$, out of which one is invalid and two are isomorphic, leading to two possible ways of forming a 4-length cycle.
6. Two edges are existing, one of which belongs to peer-1 and another does not, i.e an edge connects peer-1 and another peer, say peer-2 and another edge connect peer-3 and peer-4. The number of possible edge combinations in which a 4-length cycle is formed is $2^2 = 4$, with no invalid or isomorphic cases.

7. Two edges are existing, one belonging to peer-1, connecting to any of its neighbor, say peer-2, and another connecting peer-2 to any other peer, say peer-4. The possible number of edge combinations in which a 4-length cycle is formed is $2^2 = 4$, out of which one is invalid, thus leading to three possible ways of cycle formation.

We derived the probability of forming a 4-length cycle around peer-1 for each of these cases in the limit of large N_{rem}, as stated in Table 3. We find the probabilities of occurrence of 4-length cycles due to cases 1, 2 and 3 to be significantly smaller in the limit of large N_{rem}. Thus, there are 11 major possibilities of occurrence of 4-length cycles; considering these occurrence probabilities as equal to $\frac{\hat{d}^2}{2N_{rem}}$, the number of 4-length cycles formed around a peer at each step can be modeled as a binomially distributed random variable with the probability mass function given as $\binom{11}{i}\left(\frac{\hat{d}^2}{2N_{rem}}\right)^i\left(1-\frac{\hat{d}^2}{2N_{rem}}\right)^{11-i}$. The average number of cycles thus formed at each step equals $\frac{11\hat{d}^2}{2N_{rem}}$. However, according to Eq. (8), the number of steps required for a peer to restore its average degree is represented as $\mathscr{S}$, hence the average number of cycles around peer-1 after it has restored its average is

$$A^{(4)} = \frac{11\mathscr{S}\hat{d}^2}{2N_{rem}} \tag{14}$$

Thus the average number of cycles in the whole network equals

$$L_4 = \frac{1}{4}\cdot U_{rem}\cdot\frac{11\mathscr{S}\hat{d}^2}{2N_{rem}} = U_{rem}\cdot\frac{11\mathscr{S}\hat{d}^2}{8N_{rem}} \tag{15}$$

and the total number of short length cycles formed in the network is,

$$L = L_3 + L_4 \tag{16}$$

We simulated the HPC5 protocol to find the total number of cycles in the network, and compared the results with our model as shown in Fig. 8. The number of peers considered for our simulations was 200k, having nearly 30k ultra-peers. We found that, as per our assumptions, for low to moderate values of x, the simulation results match well with the model values and deviates when the values of x are large. This is due to an effect that we term as the *overlapping neighbor set effect*. In our model we assumed that the potential neighbor set of two random nodes is approximately same, which is true when N_{rem} is large. With increasing values of x, the assumption of N_{rem} being very large, does not hold. A lower N_{rem} value implies a large overlap of the neighbor sets of any two random peers; this results in formation of more number of short-length cycles than is actually predicted by our model. Similarly, for very low values of x, we find a small variation of the simulation results with our model. This is due to an effect that we term as the *interconnection effect*. In practice, the average degrees actually lost by the remaining peers is slightly lower than that assumed in the model. This is because, we have assumed that the nodes which leave have no connection within themselves, which is not strictly true. However, as value of x increases, the impact of overlapping neighbor set effect becomes more prominent, the balance is noticed for moderate values of x (4–6%), where simulation results match perfectly with the model values.

Table 3
The table states the derivation results of the probability of occurrence of a 4-length cycle around a peer due to each of the above stated cases. The third column indicates the number of ways of forming a cycle for a particular case.

Case #	Prob. of occurrence	# of ways
1	$\frac{1}{N_{rem}}$	1
2	$\frac{\hat{d}_{uu}}{2N_{rem}}$	4
3	$\frac{\hat{d}_{uu}}{N_{rem}}$	4
4	$\frac{\hat{d}^2_{uu}}{2N_{rem}}$	2
5	$\frac{\hat{d}_{uu}(\hat{d}_{uu}-1)}{2N_{rem}}$	2
6	$\frac{\hat{d}^2_{uu}}{2N_{rem}}$	4
7	$\frac{\hat{d}^2_{uu}}{N_{rem}}$	3

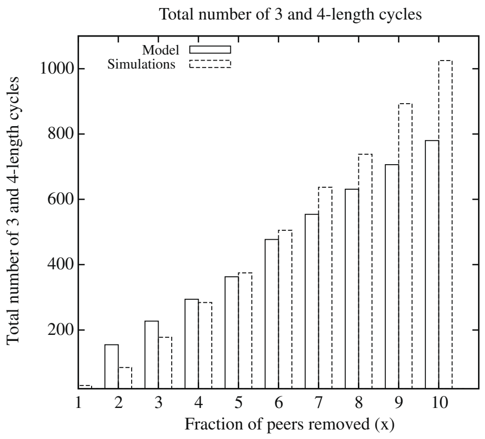


Fig. 8. The total number of cycles formed in the network in HPC5 due to inconsistent updates for a network with 200k total peers and 30k ultras.

We find that the total number of short-length cycles formed when 10% of the peers have been removed is around 1025, which is quite small compared to the size of the network. Although the number of cycles formed due to inconsistency in HPC5 is small, however, in the next section, we provide an outline of detecting and removing small length cycles arising due to inconsistency.

6. Handling consistency problems

Theoretically, as seen from Eq. (16), in HPC5 the probability that a peer will receive duplicate queries due to inconsistent updates is quite small. For a network with 200k peers and 30k ultra-peers, the simulation results show that the total number of short length cycles formed in the network, when 10% of the total peer departs, are only around 1025. This implies that the number of redundant messages generated by any peer in HPC5 due to inconsistency is quite less, and this level of inconsistency can arise only if all the 10% of the nodes attempt to regain their links simultaneously. However, to handle such rare flash crowd type situation, we propose an inconsistency handling procedure to handle these inconsistencies. Every ultra-peer maintains a record of the number of duplicate queries received over the total number of received queries in a particular time interval, a ratio that we term as *duplicate query ratio*. When the duplicate query ratio at an ultra-peer P (say d_P) crosses a threshold value (say d_t), the peer considers that inconsistency may have occurred in the network. Hence it becomes a coordinator and initiates an inconsistency handling protocol. Major objectives of the inconsistency handling protocol are to predict whether the extent of inconsistency has crossed a limit; in that case, detect the short length cycles that may have arisen due to inconsistency, and subsequently remove them by dropping a suitable link. The inconsistency handling protocol is divided into three parts: Inconsistency estimation, cycle detection and cycle removal. The algorithm is inherently distributed in nature and each node runs the algorithm separately. We discuss each of the parts precisely.

6.1. Inconsistency estimation

The initial task of the coordinator – i.e. the peer that realizes that its duplicate query ratio has crossed the threshold, d_t – is to predict, whether the inconsistency in the network has also crossed a particular threshold. We refer percentage inconsistency in the network as the percentage of peers that are part of any short length cycle. As observed from the simulation results, the average duplicate ratio of the network (say μ) is highly correlated to the percentage of inconsistency in the network (refer Appendix A). Thus, locally estimating the value of μ helps in predicting the percentage inconsistency in the entire network. Since it is difficult for the coordinator to obtain the average duplicate ratio of the whole network, μ, the coordinator predicts μ from the duplicate query ratio of the neighboring peers. Thus to obtain this estimate, the peer queries the neighbors about their duplicate query ratio, calculated over a specific interval of time, by sending a query message. The neighboring peers respond by sending their individual duplicate query ratios, and subsequently, an estimated average duplicate ratio, $\bar{x}$, is calculated. Using $\bar{x}$, the average duplicate ratio, μ, is tested with a given level of significance, $\alpha = 0.05$, to determine whether μ is less than a threshold value μ_c. If the average duplicate query ratio, μ, crosses the threshold, μ_c, the cycle detection and cycle removal algorithms are initiated to detect and remove the three or four length cycles, respectively.

Algorithm 1. Algorithm *INCONSISTENCY_ESTIMATION*. This algorithm estimates the average duplicate ratio of the network and checks whether it exceeds a threshold. Depending upon the results, it initiates *CYCLE_DETECTION* algorithm

```
$d_P$ = duplicate query ratio at peer $P$
$d_t$ = threshold duplicate query ratio
$\mu_c$ = threshold average query ratio
if $d_P > d_t$ then
  Send query to all neighbors requesting their duplicate query ratio $d_i$
  Estimate the mean $\bar{x}$ and variance $S^2$ of the duplicate query ratio of the network from the values of $d_i$
  Test the hypotheses $H_0 : \mu = \mu_c, H_1 : \mu < \mu_c$ for a given level of significance $\alpha$
  if $H_1$ == FALSE then
    Initiate CYCLE_DETECTION algorithm
  else
    Do Nothing
  end if
end if
```

Algorithm 2. Algorithm *CYCLE_DETECTION*. This algorithm detects the short length cycles through any peer P. Although this algorithm is initiated by the peer P, but this is a distributed algorithm and is run by each peer through which the *CYCLE-DETECT* control message passes

```
$SeqNo$ := Message Sequence Number
$ID(i)$ := ID of peer $i$
$T$ := Message Time Stamp
$N(i)$ := Neighbor set of peer $i$
Send a CYCLE_DETECT control message with $TTL(2)$ to neighbors $N(P)$.
if A peer $Q$ receives a CYCLE_DETECT message created by some other peer then
  Stamp $ID(Q)$ in the message
  Stamp $N(Q)$ in the message
  Save $SeqNo$, $T$, and ID of adjacent peer (:= $arrID$) from whom the message has arrived
  if the CYCLE_DETECT message is NOT a DUPLICATE message then
    If $TTL == 1$ then set $TTL := 0$ and forward to neighbors $N(Q)$, else do nothing
  else
    if $TTL == 0$ then
      Set $TTL := 2$ and send the duplicate message through $arrID$ from where the message arrived
    else
      Set $TTL := 1$ and forward the duplicate message through $arrID$ from where the message arrived and also to other neighbors $N(Q)$
    end if
  end if
end if
if Peer $P$ receives duplicate CYCLE_DETECT messages sent by itself then
  Initiate CYCLE_REMOVE algorithm
end if
```

6.2. Cycle detection

The major objective of this algorithm is to detect the edges of the cycles that have arisen due to inconsistency. If $\mu \geqslant \mu_c$ for significance level α, then the coordinator attempts to detect those nodes that are involved in a short length cycle with it by initiating this algorithm. For this, the coordinator sends a special *TTL*(2) control message, which we name as *CYCLE-DETECT* message, to its adjacent peers along with a time-stamp. On receiving this message, each adjacent peer records in its own table, the message sequence number, the time-stamp, and the id of the peer from which the message has arrived. This record helps in identifying duplicate messages. Further, each adjacent peer attaches in the message, its id and the number of its first neighbors, and forwards the message to its adjacent neighbors – other than the one from whom it has received the message – reducing the TTL by 1. If a peer receives a duplicate *CYCLE-DETECT* message, it implies the existence of a short length cycle. The information about the nodes in the cycle has to be sent to the coordinator. Hence, the peer receiving the duplicate control message attaches its id and the number of its first neighbors in each of the received message, and sends back each message through the same path from which the message had arrived. Since each peer has recorded the sending peer, they forward the duplicate control message through the same path in the reverse direction towards the coordinator peer. The messages follow the reverse path and reach the coordinator who thereby obtains the topology information. It then runs the short length cycle removal algorithm to identify those links that should be dropped to remove all short length cycles.

6.3. Cycle removal

The cycle removal algorithm is simple; the coordinator decides to drop the link that will affect the minimum number of nodes. When a coordinator receives back the duplicate *CYCLE-DETECT* control messages sent by itself, the coordinating peer extracts the information about the number of first hop neighbors of each node from the control messages. To decide which link to drop, for every link it calculates the number of peers that will use the link by taking the sum of the first hop neighbor count of the two nodes of the link, and then drops the link with the minimum sum value. However, if the link connects to a leaf-peer, then it does not drop that link and considers the next best link that connects two ultra-peers. Links to leaf-peers are not disconnected as leaf-peers are connected to only three or four ultra-peers, and reducing their degree by one will have a big impact on their performance. The coordinator peer sends a message to all the peers involved in the cycle informing the link to be dropped. All the peers update this information and the peers corresponding to the link propagate this information to all their neighbors. The pseudocode is presented in Algorithm 3. The performance of the algorithm depends upon the proper choice of the threshold average duplicate query ratio μ_c. If the threshold is kept too high, the number of redundant messages in the network might increase enormously, whereas, a lower value might lead to unnecessary deletion of links leading to a degradation in performance. Thus we need to have a tradeoff. In Section 7.6, we experimentally derive an idea of μ_c for certain request patterns. It can be seen that the asymptotic bound on the number of messages transmitted for handling an inconsistency is $O(d_u^2)$, where d_u is the sum of the maximum number of ultra and leaf-neighbors of an ultra-peer.

Algorithm 3. Algorithm *CYCLE_REMOVE*. This algorithm is initiated by a coordinator peer P to remove a suitable cyclic link of an already detected short length cycle in order to break the cycle

```
Extract the cyclic path $p$ from the message through which the peer is traversed
Find the sum of the neighbor count of the endpoint peers of every link of cycle $p$
Select the link $l_{ij}$ with minimum sum that connects peers $i$ and $j$
**if** $l_{ij}$ connects to a leaf-peer **then**
  **repeat**
    Select the link $l_{ij}$ in path $p$ with next minimum sum
  **until** $i$ and $j$ are both ultra-peers
**end if**
Send message to peers $i$ and $j$ to drop the link
```

7. Evaluation by simulation

To validate our approach, we have performed numerous experiments. We have taken different sizes (up to 100k nodes) of networks and performed experiments on those networks several times to obtain the average behavior. Through these experiments, we have seen that HPC5 performs better than the existing protocols.

7.1. Search performance

The efficiency of search algorithms can be measured using various metrics such as success rate, average number of hops required to get results (response time), message complexity, network coverage (number of nodes discovered), percentage of message duplication, etc. [12]. In our simulations, we use message complexity and network coverage as performance metrics. *Message complexity* is defined as the average number of messages required to discover a peer in the overlay network. *Network coverage* implies the number of unique peers explored during query propagation in limited flooding. We plot the network coverage and message complexity (y-axis) with *TTL*(2) and *TTL*(3) flooding against the size of the network (x-axis). We also show the performance separately for the query originating from leaf-peers and ultra-peers. To get the overall performance of the network, we choose the number of ultra-peers and leaf-peers for query flooding in the same $\frac{U}{L}$ ratio. The performance of the network is greatly influenced by the value of *TTL* used in search and thus we discuss the performance metrics based on *TTL*(2) and *TTL*(3) separately. The search performance (specially message complexity) also depends on the implementation of *QRP* technique [1]. Thus we discuss the search performance without and with QRP.

7.2. TTL(2) without QRP

It is clear from Fig. 9a and b that with *TTL*(2), cycle-5 networks are better than cycle-3 networks in both message complexity and network coverage. In cycle-5 networks, the network coverage is approximately doubled and message complexity is almost 20% less than that of cycle-3 networks. With *TTL*(2), a search query covers a significant portion (in our simulation it is more than 30%) of the cycle-5 network with lesser number of redundant messages. From the results we see that the message complexity is not as close to 1 as expected. This is because the message complexity of the leaf-peer generated query is particularly high (Fig. 9b). In cycle-5 networks, a leaf-peer can be connected with two ultra-peers which are themselves 3rd or 4th neighbors of each-other and becomes part of cycle-5 or cycle-6 (Fig. 9). From Fig. 9 we see, a leaf-peer search is initiated by its ultra-peers; both ultra-peers 1 and 4 (in Fig. 9a) {1 and 5 (in Fig. 9b)} start a *TTL*(2) flooding. Consequently redundant messages are produced at ultra-peers 2 and 3. However, hardly any redundancy is generated in ultra-peer initiated query (Fig. 10b).

As a result, cycle-5 networks generate a large number of redundant messages which get reflected in Fig. 10b.

7.3. TTL(3) without QRP

Fig. 11a shows that in our simulation, the entire cycle-5 networks is covered with *TTL*(3) search which is almost double of the coverage attained in cycle-3 networks. If any pair of ultra-neighbors of a particular leaf-peer is not more than 6 hops apart from each other (in case of *TTL*(2) it was 4 hops), then query generates redundant messages. In cycle-5 networks the probability of forming cycle-5 and cycle-6 is very high. As a result, the message complexity of cycle-5 networks becomes higher than cycle-3 networks (Fig. 11b). As mentioned earlier, Gnutella (Limewire etc.) uses *TTL*(3) in dynamic querying only for rare searches [1,2]. Therefore larger network coverage in this case, which increases the query hit probability, is more essential than slight increase of message complexity.

7.4. TTL(2) and TTL(3) with QRP

With QRP technique, searching is performed only at the ultra-peer layer, since ultra-peers contain the indices of their children [1,2]. So, the measurement of message complexity at the ultra-peer layer is more appropriate to compare results with Gnutella networks. The ultra-peer layer message complexity is shown in Fig. 12. Simulation reflects that the message complexity in *TTL*(2) of cycle-3 networks is almost 2–2.5 times than that of cycle-5 networks. Even in *TTL*(3) search, cycle-3 networks generate 25% more messages than that of cycle-5 networks. So HPC5 protocol will be more effective in the Gnutella network in the presence of QRP protocol.

7.5. Comparison of robustness

In this section we compare the robustness of the evolved networks with that of the base network. In order to test the robustness of the network evolved through HPC5, we have considered networks with 50k peers and plotted the percentage of existing peers belonging to the largest component against the percentage of peers removed. We have removed peers in two ways: (i) random removal and (ii) pathologically removing the highest-degree nodes first [22]. The upper (right) two curves in

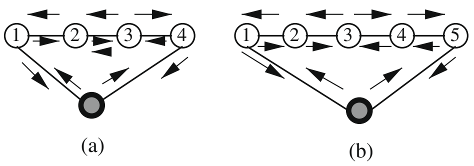
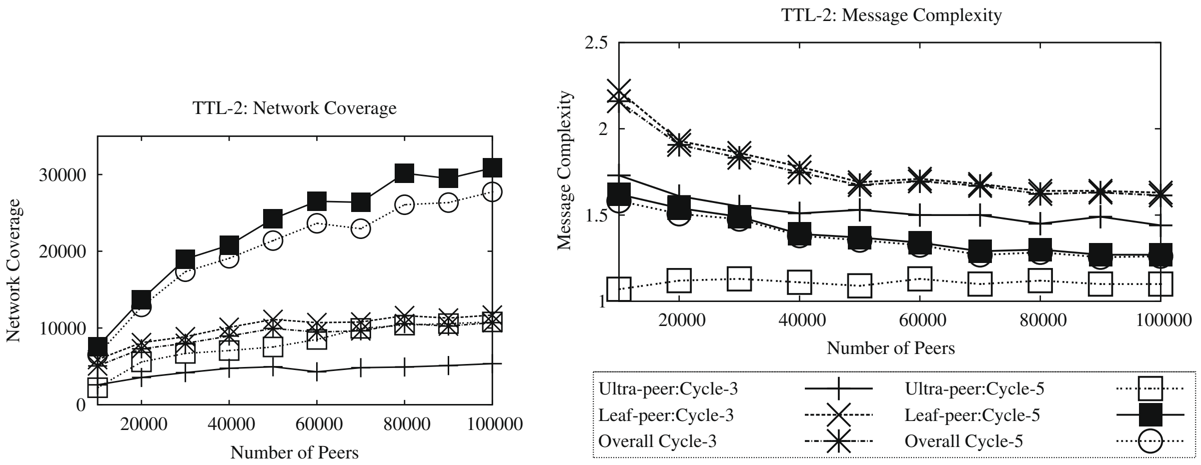
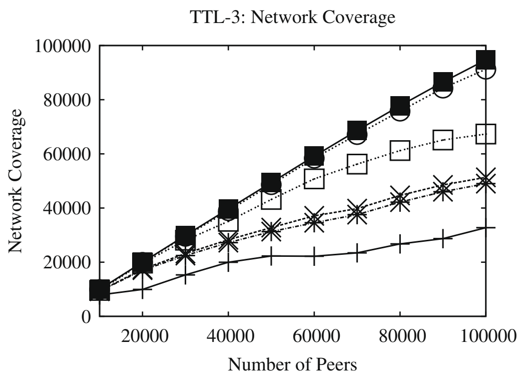


Fig. 9. Effect of leaf-peer layer with *TTL*(2) search. The arrows inside and outside of polygons indicate the directions of search by a leaf-peer and an ultra-peer, respectively.

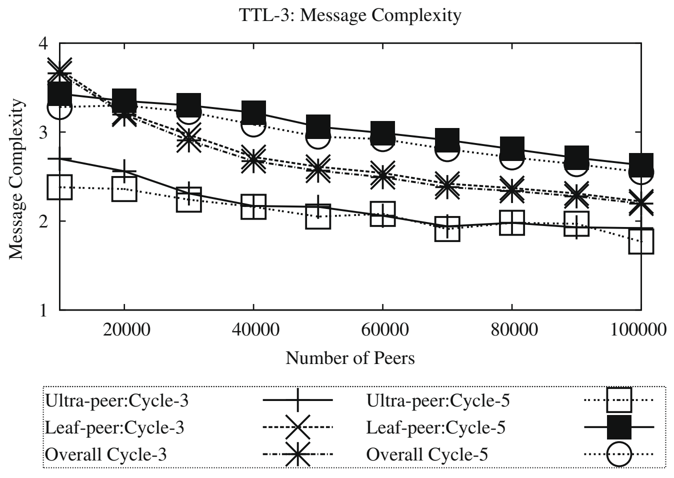


(a) Network Coverage (b) Message Complexity

Fig. 10. Network coverage and message complexity with TTL 2 for cycle-3 and cycle-5 networks.



(a) Network Coverage



(b) Message Complexity

Fig. 11. Network coverage and message complexity with TTL 3 for cycle-3 and cycle-5 networks.

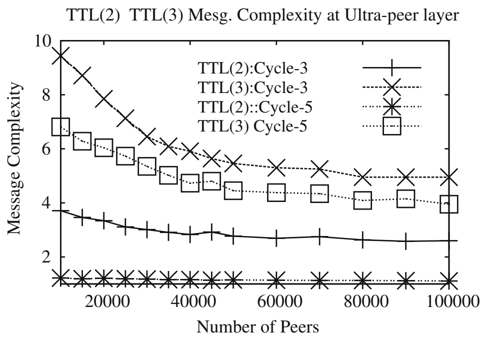


Fig. 12. Message complexity with *TTL*(2) and *TTL*(3) at the ultra-peer layer.

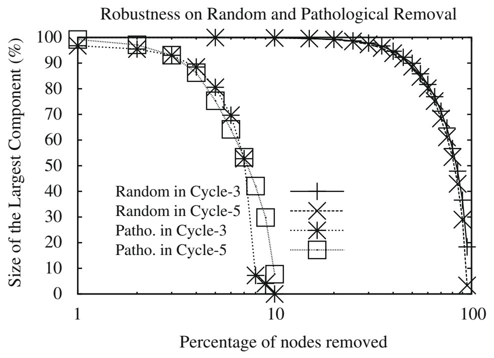


Fig. 13. Comparison of robustness between cycle-3 and cycle-5 networks.

Fig. 13 represent the effect of random removal, whereas lower (left) two curves represent pathological removal. Both cycle-3 and cycle-5 networks are extremely resilient to random removal and the largest connected component still contains 80% of existing peers even after removing almost 75–80% of nodes. The network gets fragmented after 85% peer removal. But in pathological removal, both cycle-3 and cycle-5 networks get fragmented only after removing 6–7% of total nodes. However, through our empirical study, we can conclude that robustness is similar for both cycle-3 and cycle-5 networks.

7.6. Inconsistency handling

We simulated the average duplicate ratio at the peers in the whole network for certain percentages of inconsistent peers in the network. A peer is termed as inconsistent if it is involved in a short length cycle. To simulate the effect of inconsistent peers in the network, we re-organized the cycle-5 topology of the simulated Gnutella network to form short length cycles without changing the parameters like, the number of ultra and leaf-neighbors, or the total number of neighbors of any peer. We denote the percentage of inconsistent peers in the network as the percentage of peers whose neighbor lists have been modified explicitly by us so as to form short length cycles. We initially state the simulation results showing the impact of inconsistent peers on the average duplicate ratio of the whole network and then discuss the effectiveness of our proposed inconsistency handling algorithm.

7.7. Impact of inconsistency on average duplicate ratio

We simulated the average duplicate ratio of the network with respect to the percentage inconsistency in the network for a total node count of 125k and 150k. Every peer in the network was considered as equally active, i.e. each peer had the same average query sending rate. Fig. 14 shows the results with and without inconsistency handling. As can be seen, when no inconsistency handling is applied, the average duplicate ratio of the network increases rapidly with increasing percentage inconsistency.

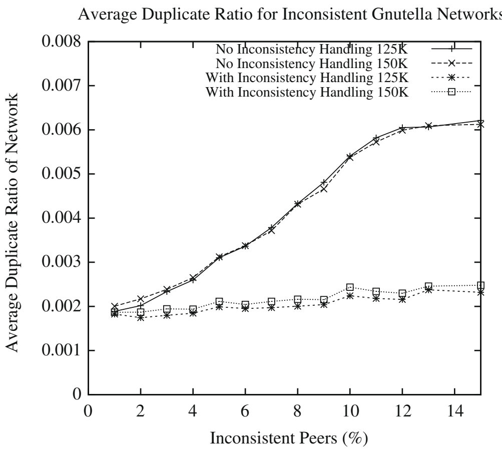


Fig. 14. The graph plots the change in average duplicate ratio of the peers μ in the network with increasing percentage of minimum inconsistent peers. The number of peers used in the simulation are 125k and 150k. The graph shows the results when no inconsistency handling is done and also for the case when inconsistency handling is applied beyond a threshold of average duplicate ratio $\mu_c = 0.002$.

From the *Pearson product-moment correlational coefficient* of the simulated results (refer Appendix A), we find that the average duplicate ratio and the percentage inconsistency of the network exhibit a very high linear dependency. Thus the average duplicate ratio of the network can be used to predict the percentage inconsistency in the network, i.e. we can claim that when the average duplicate ratio of the peers in the network, μ, exceeds a threshold μ_c, it implies that the percentage inconsistency in the network, ι, has also exceeded a threshold value of ι_c and vice versa. Hence, our proposed inconsistency handling algorithm estimates the average duplicate ratio of the network to obtain an idea about the percentage inconsistency in the network. We, now state the simulation results showing the efficiency of our proposed inconsistency handling algorithm in predicting and reducing the average duplicate ratio of the whole network.

7.8. Efficiency of the proposed inconsistency handling algorithm

We measure the efficiency of the proposed inconsistency algorithm by measuring the percentage correctness of inconsistency estimation by the peers and by the reduction in the average duplicate ratio of the network for different percentage of peer inconsistency. The inconsistency handling algorithm was divided into three parts: inconsistency estimation, cycle detection, and cycle removal. We simulated our inconsistency estimation algorithm and obtained the results for percentage correctness of the estimation – whether the percentage inconsistency ι has crossed a threshold value $\iota_c = 2\%$, when the average duplicate ratio of the network has crossed a threshold value $\mu_c = 0.002$, for 0.05 level of significance, and vice versa. In our simulations, we observed that the average duplicate ratio is 0.002, when the peer inconsistency is nearly 2%. Thus, the threshold value $\mu_c = 0.002$ predicts whether the peer inconsistency has crossed 2%. The percentage inconsistency of the network has been varied from 1% to 15% for a total network size of 125k peers. The results shown in Fig. 15 reveal that the percentage correctness of the estimations are nearly 100%, except for some inconsistencies very near to 2%. This is because, for low percentage inconsistency, the actual average duplicate ratio of the network is very close to 0.002; hence the number of incorrect estimations are higher compared to the case when the actual average duplicate ratio is much higher than the threshold.

The efficiency of the cycle detection and removal algorithms is shown in Fig. 14. As the figure shows, the cycle detection and removal algorithm successfully reduces the average duplicate ratio of the whole network to the threshold bound of 0.002 for all percentages of peer inconsistency. However, the cycle removal algorithm deletes

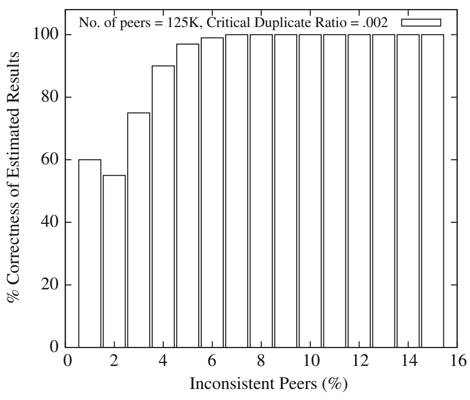


Fig. 15. Percentage correctness of the test, whether the percentage inconsistency ι has crossed a threshold value $\iota_c = 2\%$, when the average duplicate ratio of the network has crossed a threshold value $\mu_c = 0.002$ for 0.05 level of significance and vice versa. The percentage inconsistency was varied from 1% to 15%. The total number of peers used was 125k.

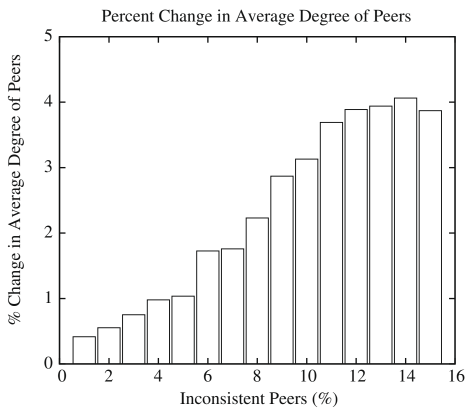


Fig. 16. The graph shows the change in the average degree of the peers when the inconsistency handling protocol is applied on the peers when the average duplicate ratio μ of the network crosses the threshold $\mu_c = 0.002$. The number of peers used in the simulation was 125k.

certain links in the network to remove inconsistencies. Hence we need to study the effect of link removal on the average degree of the peers. Fig. 16 shows the percentage change in the average degree of the peers following cycle removal algorithm. The maximum reduction of the average degree is around 4%. Since from Table 2, we find that the average degree of the ultra-peers is $d_{uu} + d_{ul} \approx 48$, hence the number of edges that might be removed per peer is expected to be around 1–2, at most.

We next compare the performance of the HPC5 protocol with the *Distributed Cycle Minimization Protocol* (DCMP) for p2p networks.

7.9. Comparison with DCMP

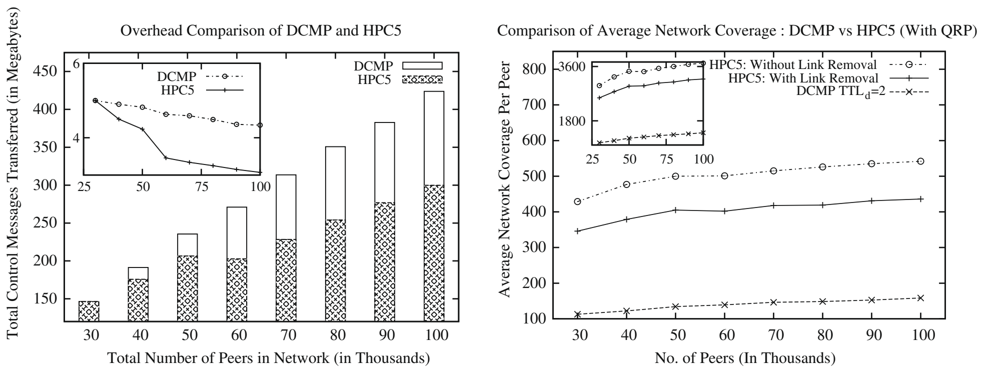
In this section, we compare the performance of DCMP with HPC5 in terms of message overhead and the network coverage of the peers. The DCMP protocol reduces traffic redundancy by detecting the short length cycles dynamically and deleting suitable edges to remove these cycles. In Algorithm 4 (Appendix C), we highlight the major steps of the DCMP algorithm for ready reference. The performance comparison is based on the simulation results that we obtain from implementing the DCMP protocol in our simulator. In the DCMP paper [25], results have been given for at most 10k nodes, without considering certain specific details of the Gnutella network (like average degrees of the ultra and leaf-peers, QRP, etc.). Hence to make the comparison with HPC5, we deemed fit to implement the DCMP protocol in the simulation platform.

The DCMP protocol follows a lazy approach in detecting and removing cycles in Gnutella networks; however, as the gatepeers and transitive peers constantly broadcast tagged messages to their *TTL*-hop neighbors, a fixed number of control messages will always flow through the network. Another problem with the DCMP protocol is that the deletion of edges to remove cycles might reduce the network coverage of the peers in the network. In contrast, HPC5 generates control messages only at the bootstrapping phase, when a node joins a network. Since HPC5 avoids short length cycle formation by selecting suitable neighbors, rather than removing edges, the network coverage is also higher as compared to DCMP. However, DCMP works for any *TTL* values and it has been shown to perform well for *TTL* values upto 8, whereas, HPC5 supports only *TTL*(2) flooding. We implemented the DCMP protocol on our simulator; the comparison of the message overhead and the network coverage with HPC5 is detailed next.

7.10. Comparison of message overhead

We measured the message overhead as the total number of control messages sent over the network for a certain period of time. We compare the message overhead for both HPC5 and DCMP with respect to the size of the network (number of nodes). The list of various simulation parameters that we have used for the simulations are as follows:

1. We considered that every peer sends an average of 3.6 query requests per hour distributed uniformly (as considered in the DCMP paper [25]), and run the simulation for two hours of simulation time and various network sizes.
2. For both HPC5 and DCMP, we used a growing network where peers arrive in a burst of 25k after every 10 min. The growth process in each case is similar and is discussed in Section 2.
3. The minimum Gnutella message size is 23 bytes [16]; we consider each *Node Information Vector* (NIV) field of a peer *A* of length equal to six times the *number of neighbors of A*. This is because, for each neighbor of *A*, 4 bytes is allocated to represent the address of *A* and 2 bytes for representing the bandwidth of *A*, making a total of 6 bytes.
4. In our experiments, we ignore the overhead in DCMP due to dissemination of gatepeer information in the form of tagged messages.
5. For the case of HPC5, we assume that a peer sends a control message of 23 bytes for a connection request to a peer. The peer responds with the list of addresses



(a) Control Overhead (b) Network Coverage

Fig. 17. (a) Shows the total number of control messages transferred (in megabytes) in the whole network. The figure in inset shows the average number of control messages sent per peer (in kilobytes). (b) Compares the average network coverage of both leaf and ultra-peers in DCMP and HPC5. The figure in inset shows the coverage of the leaf-peers, and the main figure shows the average networks coverage of the ultra-peers in a Gnutella network. The number of peers in the simulations is varied from 30k to 100k.

of its neighbors; each address along with the port number is assumed to be of 6 bytes. An additional overhead of 23 bytes, which is the minimum Gnutella message size is also assumed for the latter case.

As can be seen from Fig. 17a, for a large network size ($N = 100\text{k}$), in HPC5 the total number of control messages sent through the network (measured in bytes) is less by almost 30% compared to DCMP. This is because, in HPC5, the overhead is mainly due to the response messages sent by the peers in reply to a connection request. With increasing number of nodes, the probability of finding a valid peer at a particular attempt increases; thus peers require to send fewer connection requests, thereby reducing the number of control messages. Hence, with increasing network size, the average number of control messages sent per peer decreases much faster compared to DCMP (refer inset in Fig. 17a).

7.11. Comparison of network coverage

We also measured the average $TTL(2)$ network coverage of both ultra and leaf-peers, for both DCMP and HPC5, when *QRP* is followed. We assume that DCMP preserves all cycles of length greater than 4 and eliminates all 3 and 4-length cycles, i.e. $TTL_d = 2$ [25]. We compare the results of DCMP with two possible cases of HPC5.

1. In the first case, we assume that HPC5 does not remove any link, i.e. HPC5 does not initiate the inconsistency handling algorithm.
2. In the second case, to compare DCMP with the worst case performance scenario of HPC5, we remove 4% of the total links (as seen in Fig. 16), but from the ultra-peer level of the network. This simulates the possible scenario that a large number of links have been removed by the inconsistency handling algorithm of HPC5, due to inconsistency. Removing 4% of the total links reduced the average degree of the ultra-peers by as much as 4%.

As observed in Fig. 17b, in both cases, DCMP has much lower network coverage as compared to HPC5. This is because, Gnutella networks form a large number of 3 and 4-length cycles; hence in DCMP, a high number of duplicate messages are generated at each peer. Thus DCMP eliminates a large number of links (we observed that for $TTL_d = 2$, DCMP eliminates about 16% of the total links), thereby reducing the average network coverage of the peers.

8. Conclusion and future work

In this paper, we have presented a handshake protocol which is compatible with Gnutella-like unstructured two-tier overlay topology. The objective of the protocol is to enhance the network coverage of the peers, besides reducing redundant messages, in the existing Gnutella networks. Our protocol is based on the observation that network coverage of peers can be improved, and redundant messages can be reduced, if we suitably eliminate short length cycles from the network topology. We have shown that the protocol is far more efficient than existing protocols. We also studied the various hurdles in implementing our scheme and proposed methods to handle inconsistency in the network. The basic strength of our protocol lies in its simplicity and the ease with which it can be implemented on existing Gnutella networks. The modularity of the handshaking and the inconsistency handling protocol provides a suitable choice to the designers whereby they can decide on the use of any of these protocols without affecting the other one. Moreover, the inconsistency handling protocol can be tuned dynamically. Our protocol can be instrumental in improving the scalability of Gnutella networks; moreover, this concept of topology management can be customized for other unstructured p2p networks to improve blind search behavior, resulting it in performing comparably with many proposed intelligent search mechanisms that are generally more computation intensive.

Appendix A. Relation between average duplicate ratio and inconsistent peers

The following table indicates the result of the simulations. If X and Y denote two parameters of an experiment and we take n measurements of the parameters denoted by (x_i, y_i), where $i = 1, 2, \dots n$, then the *Pearson Correlation Coefficent* between X and Y is given by (see Table 4)

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}},$$

where all the summations are carried for $i = 1$ to n. For the given data, let X denote the percentage of inconsistent peers and Y denote the average duplicate ratio at the peers; we calculated the value of r_{xy} for 120k peers and

Table 4
The table shows the simulation results of the average duplicate ratio at the peers for a given percentage of inconsistent peers. The total number of peers used in the simulation are 125k and 150k.

120k		150k	
% Inconsistent peers	Average duplicate ratio	% Inconsistent peers	Average duplicate ratio
1	0.001889	1	0.002005
2	0.002019	2	0.002171
3	0.002336	3	0.002388
4	0.002597	4	0.002654
5	0.003103	5	0.003124
6	0.003366	6	0.003376
7	0.003786	7	0.003717
8	0.004328	8	0.004315
9	0.004812	9	0.004661
10	0.005406	10	0.005374
11	0.005820	11	0.005725
12	0.006051	12	0.005991
13	0.006066	13	0.006020
14	0.008082	14	0.005996
15	0.006214	15	0.006126

150k which is around 0.984 for both cases. Since the values of r_{xy} are near to 1 for both cases, so it shows that there is a very high linear dependency between the two parameters. Thus the peers can use the duplicate ratio of itself and the neighboring peers to estimate the average duplicate ratio of the whole network based on which it can decide whether to initiate the inconsistency handling protocol.

Appendix B. Importance of cycle-5 networks for gnutella

Gnutella networks use a $TTL(2)$ query for most searches; however, for rare searches it uses a TTL value of 3 (dynamic querying [5]). A $TTL(2)$ query from a peer reaches its first as well as its second neighbors. Thus, the network coverage of a peer may be defined as the sum of the set of its first and second neighbors. Suppose a peer, P, in a p2p network has $F(P)$ number of first neighbors, and let $L(P)$ denote the number of outgoing edges from $F(P)$ first neighbors of P. Moreover, suppose $L(P)$ edges connect to $S(P)$ unique peers other than P or any of its first neighbors. Then the network coverage of P is given by the sum $F(P) + S(P)$. We prove that, for given $F(P)$ and $L(P)$, the peer P, that uses $TTL(2)$ search queries, will have a maximum network coverage, if the peer P does not involve itself in any cycle of length less than five. Moreover, with the above stated condition, the peer P does not produce any redundant message at any of its first or second neighbors. We prove each of the statements using a lemma and a theorem as stated below. In each case, we represent the network as a graph $G\langle V, E\rangle$, where the vertex set V represents the peers in the network and the edge set E represents the overlay links between them.

Lemma 1. *Given a network, represented as, $G\langle V, E\rangle$ with a source vertex Q. A $TTL(2)$ flood query from the source vertex Q will reach a vertex $v \in V$ exactly once, if the minimum length cycle through Q and v has length 5, and the minimum distance of v from Q, given by $D_Q(v)$, is less than or equal to 2.*

Proof 1. Since the minimum distance of node v from Q, given as $D_Q(v) \leqslant 2$, so node v must be reachable from Q using a $TTL(2)$ flood query. We now prove that if there exists a cycle between Q and v of minimum length 5 and $D_Q(v) \leqslant 2$, then there will be no duplicate query at vertex v from source Q. We prove this by method of contradiction. Let us assume that node v receives a duplicate query from source Q through two separate paths, say p_1 and p_2. Then, the path length between Q and v through each of these paths, p_1 or p_2, is at most 2. Thus the cycle represented by $Q \rightsquigarrow_{p_1} v \rightsquigarrow_{p_2} Q$ is at most of length 4 which is a contradiction to our initial assumption that Q and v are connected through a cycle of minimum length 5. Thus no redundant messages can be produced at v. □

Theorem 1. *Given a network, represented as $G\langle V, E\rangle$, with a source vertex Q that uses $TTL(2)$ based flooding mechanism for searching. Let $F(Q)$ be the number of first neighbors of Q and $L(Q)$ denote the total number of outgoing edges from the $F(Q)$ first neighbors. If $L(Q)$ number of outgoing edges from the first neighbors leads to further $S(Q)$ unique second neighbors of Q, then for a given value of $F(Q) + L(Q)$, the network coverage of Q, given by $C(Q) = F(Q) + S(Q)$, is maximum, if and only if Q is not involved in any cycle of length less than 5.*

Proof 2. *If-part: If Q is not involved in any cycle of length less than 5, then $C(Q)$ is maximum.* The *if-part* of the proof is trivial and is directly obtained from Lemma 1. Since according to Lemma 1, if the minimal length cycle from Q through any node $v \in V$ has length 5, and $D(v) \leqslant 2$, a $TTL(2)$ flood message from Q must reach v exactly once. This implies that all the nodes within two hops from Q are reached exactly once from Q, i.e. all the $L(Q)$ edges from first hop neighbors of Q must reach to different and unique nodes. Hence, the number of unique second neighbors $S(Q)$ reached is given by $S(Q) = L(Q)$. Since $S(Q)$ cannot be greater than $L(Q)$, hence, for a given $F(Q) + L(Q)$, the coverage of node Q will be maximum and equal to $C(Q) = F(Q) + L(Q)$. Thus it is proved that if Q is not involved in any cycle of length less than 5, then $C(Q)$ is maximum.

Only-if part: If $C(Q)$ is maximum, then Q is not involved in any cycle of length less than 5. We prove the above statement by method of contradiction. Suppose $C(Q)$ is maximum for a given $F(Q) + L(Q)$, when Q is involved in some cycles of length less than 5. We note that, any pair of peers will be connected by a single edge only (no parallel edges occur), so there is no possibility of any cycle of length 2. Thus Q can be involved in cycle of either length 3 or 4. Let us initially assume that Q is involved in a cycle of length 3. If Q is involved in cycle of length 3, then any two of its first neighbors (say N_1 and N_2) must be connected by an edge E_{12} among themselves. We remove the end of edge E_{12} connected to N_1, and connect it to a new node, N_a, that is not presently a first or second neighbor of Q. Thus, for same value of $F(Q) + L(Q)$, the number of second neighbors of Q increases by 1; hence we have the new coverage $C'(Q) > C(Q)$, which is not possible as we have already assumed that $C(Q)$ is maximum. Thus, our assumption was wrong and Q cannot be involved in a cycle of length 3.

Now suppose we assume that Q is involved in a cycle of length 4. Then any two of its first neighbors (say again N_1 and N_2) must be connected to a common peer, say N_3. We remove the link connecting N_2 and N_3, and add a new link from N_2 to a new peer N_b that is not currently a first or second neighbor of Q. Thus, as in the previous case, for a given value of $F(Q) + L(Q)$, the number of second neighbor of Q increases by 1; hence the new coverage $C'(Q) > C(Q)$, which is not possible as we have already assumed $C(Q)$ as maximum. Hence Q cannot be involved in any cycle of length 4.

Thus for $C(Q)$ to be maximum, Q cannot be involved in any cycle of length less than 5. □

The generalization of the proof can be thought as, for a given number of edges in a p2p network, the network coverage of the peers can be enhanced, if we can increasingly eliminate short length cycles from the network.

Appendix C. Steps of algorithm of the DCMP protocol

We present an algorithmic outline (Algorithm 4) of the DCMP protocol for ready reference.

Algorithm 4. Algorithm *DCMP*. Each message in DCMP is assigned a globally unique ID (GUID), every peer A on receiving a message from B records the GUID and its direction of travel $(B \rightarrow A)$ in a history table. The algorithm is initiated by a peer A that receives a duplicate message (two messages with same GUID) from two different directions, say B and C

```
GUID := Globally Unique ID of the duplicate
    message
DetectionID := The direction of the duplicate
    message, (say B → A)
NIV(A) := Node Information Vector of peer A
    containing the IP address, bandwidth, cpu speed,
    degree and the neighbors of A
On receiving duplicate messages from a source Q
    through 2 different adjacent nodes, say B and C,
    peer A introduces a new message called,
    Information Collecting (IC) message with same
    GUID containing DetectionID, NIV(A) and
    forwards it to both B and C.
For each node i on receiving the IC message
repeat
   Record the GUID and DetectionID
   Add its own NIV, i.e. NIV(i)
   Propagate the IC message in reverse direction
    from which the original message arrived.
until A node i receives both IC messages
if peer X receives both IC messages then
   peer X checks the NIVs' of each peer in message
    and finds the most powerful peer (Gatepeer)
    depending upon the degree, bandwidth and CPU
    speed,
   determines the cut position by finding the link
    whose end nodes are maximal equidistant from
    Gatepeer,
   sends a Cut Message (CM) to the peers in the cycle
    containing the information about the link to cut
    and the IP address of the gatepeer.
end if
if the CM arrives at the Gatepeer then
   then the Gatepeer generates a tagged message
    containing the NIV of the gatepeer and the
    number of hops from the message origin to the
    gatepeer,
   piggybacks the tagged message with other
    messages that are forwarded by the gatepeer,
   periodically sends the tagged message; initially it
    is sent frequently, the rate gradually slows down
    with time.
end if
if a peer Y receives tagged messages from more than
    one direction then
   peer Y becomes a transitive peer,
   it also generates tagged messages and piggybacks
    them.
end if
```

References

[1] Gnutella and Limewire: <URL www.limewire.org>.
[2] Gnutella protocol specification 0.6: <http://rfc-gnutella.sourceforge.net>.
[3] Gnutella: <URL www.gnutellaforums.com>.
[4] Gwebcache system: <URL www.gnucleus.com>.
[5] How gnutella works, <URL wiki.limewire.org>.
[6] G. Chen, C.P. Low, Z. Yang, Enhancing search performance in unstructured P2P networks based on users' common interest, IEEE Transactions on Parallel and Distributed Systems 19 (6) (2008) 821–836.
[7] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, M. van Steen, Gossip-based peer sampling, ACM Transactions on Computer Systems 25 (3) (2007) 8.
[8] P. Karbhari, M.H. Ammar, A. Dhamdhere, H. Raj, G.F. Riley, E.W. Zegura, Bootstrapping in Gnutella: a measurement study, in: PAM, Lecture Notes in Computer Science, vol. 3015, Springer, 2004, pp. 22–32.
[9] Y. Liu, X. Liu, L. Xiao, L.M. Ni, X. Zhang, Location-aware topology matching in P2P systems, in: INFOCOM: The Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 2004.
[10] Y. Liu, L. Xiao, X. Liu, L.M. Ni, X. Zhang, Location awareness in unstructured peer-to-peer systems, IEEE-TPDS: IEEE Transactions on Parallel and Distributed Systems 16 (2005) 163–174.
[11] K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, A survey and comparison of peer-to-peer overlay network schemes, Communications Surveys & Tutorials, IEEE (2005) 72–93.
[12] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proceedings of the 2002 International Conference on Supercomputing (16th ICS'02), ACM, 2002, pp. 84–95.
[13] S. Merugu, S. Srinivasan, E.W. Zegura, Adding structure to unstructured peer-to-peer networks: the role of overlay topology, in: NGC 2003, and ICQT 2003, Proceedings, Lecture Notes in Computer Science, vol. 2816, Springer, 2003, pp. 83–94.
[14] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon, Network motifs: simple building blocks of complex networks, Science 298 (5594) (2002) 824–827.
[15] C. Papadakis, P. Fragopoulou, E. Athanasopoulos, M.D. Dikaiakos, A. Labrinidis, E. Markatos, A feedback-based approach to reduce duplicate messages in unstructured peer-to-peer networks, in: Integrated Research in GRID Computing, 2007.
[16] M. Portmann, A. Seneviratne, Cost-effective broadcast for fully decentralized peer-to-peer networks, Computer Communications 26 (11) (2003) 1159–1167.
[17] S. Saroiu, P.K. Gummadi, S.D. Gribble, A measurement study of peer-to-peer file sharing systems, Tech. rep., July 23 2002.
[18] S. Shaw, J. Chandra, N. Ganguly, HPC5: an efficient topology generation mechanism for gnutella networks, in: 10th International Conference on Distributed Computing and Networking – ICDCN, Hyderabad, 2009.
[19] D. Stutzbach, R. Rejaie, Capturing accurate snapshots of the gnutella network, IEEE INFOCOM, 2005, pp. 2825–2830.
[20] D. Stutzbach, R. Rejaie, Understanding churn in peer-to-peer networks, in: IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet Measurement, ACM, New York, NY, USA, 2006, pp. 189–202.
[21] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, W. Willinger, On unbiased sampling for unstructured peer-to-peer networks, IEEE/ACM Transactions on Networking 17 (2) (2009) 377–390.
[22] D. Stutzbach, R. Rejaie, S. Sen, Characterizing unstructured overlay topologies in modern P2P file-sharing systems., in: Internet Measurment Conference, USENIX Association, 2005, pp. 49–62.
[23] D. Gavidia, S. Voulgaris, M. van Steen, CYCLON: inexpensive membership management for unstructured P2P overlays, Journal of Networks and System Management 13 (2) (2005) 197–217.
[24] S. Zhao, D. Stutzbach, R. Rejaie, Characterizing files in the modern gnutella network: a measurement study, in: Proceedings of SPIE/ACM Multimedia Computing and Networking, vol. 6071, 2006.
[25] Z. Zhenzhou, K. Panos, B. Spiridon, DCMP: a distributed cycle minimization protocol for peer-to-peer networks, in: Parallel and Distributed Systems, IEEE Transactions, vol. 19, IEEE, 2008, pp. 363–377.

Joydeep Chandra completed B.Tech. (Computer Science and Engineering) from Haldia Institute of Technology, India in 2000 and M.Tech. (Computer Science and Engineering) from Indian Institute if Technology, Kharagpur in 2002. Presently, he is pursuing his Ph.D. from Indian Institute of Technology, Kharagpur. His research interests include p2p networks, distributed algorithms and complex networks.

Santosh Kumar Shaw completed BE (Computer Science and Engineering) from Bengal Engineering College(D.U), India in 2001 and M.Tech (Computer Science and Engineering) from Indian Institute of Technology, Kharagpur, India in 2008. He served as a faculty member at University Institute of Technology, Burdwan University, India. He also served in Magma Design Automation as Associate Member of Technical Staff. He is presently working with Berkeley Design Automation as a Member of Technical Staff. His research interests includes P2P networks and Distributed algorithms.

Niloy Ganguly is an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur. He has received his B.Tech from IIT Kharagpur in 1992 and his PhD from BESU, Kolkata, India in 2004. He has spent two years as Post-Doctoral Fellow in Technical University, Dresden, Germany before joining IIT, Kharagpur in 2005. His research interests are in peer-to-peer networks in particular and distributed dynamic networks in general. He has applied various bio-inspired techniques to solve problems in such networks. He also works on applying complex networks methodology in various information retrieval problems. He is presently leading a research group comprising of several research scholars and masters student as well as various collaborators from industry and academia. Visit the Complex Networks Research Group (CNERG) at www.cse-web.iitkgp.ernet.in/~cnerg/.