Δημιουργία και Διατήρηση Ενημε
Αντιγράφων σε Αδόμητα Συστή
Ομότιμων Κόμβων με
Power-Law Τοπολογίας

Μυρτώ Ντέτσικα

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕ

— ♦ —

Ιωάννινα, Ιούλιος 2009

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF IOANNINA

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΔΙΑΤΗΡΗΣΗ ΕΝΗΜΕΡΟΤΗΤΑΣ ΑΝΤΙΓΡΑΦΩΝ ΣΕ ΑΔΟΜΗΤΑ ΣΥΣΤΗΜΑΤΑ ΟΜΟΤΙΜΩΝ ΚΟΜΒΩΝ ΜΕ ΤΟΠΟΛΟΓΙΑ POWER-LAW

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από την

Μυρτώ Ντέτσικα

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιούλιος 2009

DEDICATION

To my family

ACKNOWLEDGMENTS

I would like to thank my supervisor, Prof. Evaggelia Pitoura, for her time and the valuable comments during the elaboration of this thesis. I would also like to thank all the members of the DMOD laboratory and especially my friends Marina Drosou, Kostas Stefanidis, Eftychia Baikoussi and Dimitris Souravlias for their valuable advices, understanding and encouragment throughout the years of my studies. At last, I would like to thank my friends and family for supporting me in any possible way.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Ntetsika A. Mirto. MSc, Computer Science Department, University of Ioannina, Greece. July, 2009. Replication and Consistency Maintenance in Unstructured Peer-to-Peer Systems with Power-Law Topology. Thesis Supervisor: Pitoura Evaggelia.

Peer-to-peer communication model has been widely used over the past few years for data or resource sharing. In p2p applications such as Kazaa, each user is connected to a number of other users, thus forming a logical overlay network. In decentralized, unstructured p2p systems, peers that join the network randomly choose a number of other participating peers to connect with and there is no precise control over the network topology or where the data items are placed. It has been observed that this kind of networks tend to a power-law topology, where there are few peers that are very popular and the majority of peers have only a few connections to those popular peers. A peer that wishes to retrieve a particular data item poses a look-up query. Such queries are forwarded through the overlay network until a peer that offers the data item is located. Maintaining multiple replicas of data items has been widely used for speeding up the look-up process. This thesis summarizes some replication methods that exist in bibliography for creating replicas of data items and maintaining the consistency of different replicas in case of updates on the content of the data items. We also investigate through experimental study how the performance of those methods is affected by the topology of the overlay network. Based on our observations, we propose alternative replication and update strategies that consider the topological properties of the overlay network.

ΕΚΤΕΝΗΣ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

Ντέτσικα Μυρτώ του Αντωνίου και της Βασιλικής. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούλιος 2009. Δημιουργία και Διατήρηση Ενημερώτητας Αντιγράφων σε Αδόμητα Συστήματα Ομότιμων Κόμβων με Τοπολογία Power-Law. Επιβλέπουσα: Ευαγγελία Πιτουρά.

Τα μη-κεντρικοποιημένα, αδόμητα συστήματα ομότιμων κόμβων όπως το Kazaa έχουν επικρατήσει τα τελευταία χρόνια ως ένα από τα κυριότερα μέσα για τον διαμοιρασμό ενός μεγάλου όγκου δεδομένων, πόρων ή υπηρεσιών μεταξύ ενός μεγάλου αριθμού χρηστών. Στα συστήματα αυτά, οι χρήστες συνδέονται μεταξύ τους σχηματίζοντας ένα λογικό δίκτυο επικάλυψης. Ο τρόπος με τον οποίο οι χρήστες που εισέρχονται στο σύστημα επιλέγουν με ποιούς από τους ήδη συνδεδεμένους κόμβους θα συνδεθούν είναι τυχαίος, με αποτέλεσμα να μην υπάρχει κανένας έλεγχος πάνω στην τοπολογία του λογικού δικτύου ή στο πως είναι τοποθετημένα τα δεδομένα στο δίκτυο. Στην πραγματικότητα, έχει αποδειχθεί ότι η τοπολογία στην οποία τείνουν αυτά τα δίκτυα είναι η *power-law*, στη οποία υπάρχουν λίγοι δημοφιλείς κόμβοι με τους οποίους οι περισσότεροι κόμβοι επιλέγουν να συνδεθούν. Για τον λόγο αυτό, οι power-law γράφοι αποτελούν κατάλληλα μοντέλα για το δίκτυο επικάλυψης ενός αδόμητου συστήματος ομότιμων κόμβων. Τα δίκτυα αυτά βασίζονται σε αλγόριθμους πλημμύρας προκειμένου να εντοπιστούν κόμβοι που διαθέτουν δεδομένα για τα οποία ενδιαφέρεται ένας χρήστης. Τα power-law δίκτυα έχουν κάποια ιδιαίτερα τοπολογικά χαρακτηριστικά (μικρή διάμετρο, μικρή μέση απόσταση μεταξύ δύο κόμβων) τα οποία επηρεάζουν την διαδικασία εντοπισμού δεδομένων. Η διατήρηση πολλαπλών αντιγράφων των διαμοιραζόμενων δεδομένων (*replication*) χρησιμοποιείται ευρέως για την επιτάχυνση της διαδικασίας αναζήτησης δεδομένων. Η εργασία αυτή αρχικά συνοψίζει διάφορες τεχνικές που έχουν προταθεί τόσο για την δημιουργία αντιγράφων όσο και για την διατήρηση της ενημερότητας των διαφορετικών

αντιγράφων και ερευνά την επίδραση που έχει η τοπολογία του δικτύου επικάλυψης στην απόδοσή τους. Από την πειραματική μελέτη προκύπτει ότι στην τοπολογία power-law ανταλλάσσεται μεγάλος αριθμός μηνυμάτων τα οποία μπορούν να αποφευχθούν. Μελετώνται έτσι εναλλακτικές τεχνικές δημιουργίας αντιγράφων και διατήρησης της ενημερότητάς τους στις οποίες λαμβάνονται υπόψην οι ιδιότητες της power-law τοπολογίας.

CHAPTER 1. INTRODUCTION

1.1. Introduction

Peer-to-Peer (p2p) systems have gained a lot of attention in the social, commercial and academic communities. Millions of people all over the world use p2p applications on a daily basis for data sharing and communication. P2p systems rely on a symmetric communication model where participating peers are both servers and clients. They are fully decentralized, thus avoiding the bottleneck imposed by the presence of a server in traditional systems and they are highly resilient to peers' arrivals and departures.

Each participating peer in a p2p system is connected to a number of other peers, thus forming an *overlay network*. A peer is connected to another peer in the overlay network if it knows its location in the p2p network. Connections between a pair of peers are built over the physical TCP/IP network.

The overlay network is built to facilitate the operation of a p2p system. In data sharing p2p systems, a basic functionality is discovering the data of interest. A look-up query for data items may be posed at any peer in the overlay. The query is then routed through the overlay to efficiently discover the peers that hold the requested data items. For such a query routing, it is important that the number of peers in the overlay network that need to be contacted for locating a data item is minimized and the number of messages that are exchanged is as "small" as possible.

Based on how the peers are linked to each other in the overlay network, we can classify the p2p networks as structured or unstructured. In *structured p2p systems*, peers in the overlay are organized in rigid topologies, such as ring, grid or a multidimensional cube and the data items are placed at specific peers according to some rules. In structured overlays, lookup reduces to locating the peer in the overlay that is responsible for the corresponding data item. *Unstructured p2p systems* are formed when the peers in the overlay are linked in an arbitrary, ad hoc manner. The topology of the resultant overlay network is not rigid, although it may have some properties, and there is no correlation between a peer and the data items managed by it. To locate data of interest, a peer queries its neighbors in the overlay, which in turn query their neighbors, and so on, until the query hits on a peer holding the requested data item. However, this procedure provides no guarantees on the complexity of search.

In many existing systems, upon joining the network, a peer selects to connect to another peer essentially at random. In these systems, topologies often tend towards a power-law degree distribution, where some long-lived peers have many connections, while most other peers have a few connections. For this reason, power-law graphs are used for modeling the overlay network of an unstructured p2p system. Some topological properties of power-law graphs, such as the diameter of the graph or the degree of each peer, can help us address some crucial questions regarding the centrality of each peer or the connectivity of the graph.

Maintaining multiple copies (*replicas*) of data items is a commonly used mechanism for improving the *performance* and *fault-tolerance* of any distributed system. By placing copies of data items closer to their requesters, the *response time* of queries can be improved. In addition, replication improves *load balancing*. If highly demanded data items are replicated, the query load can be evenly distributed among the peers that hold these copies. Similarly, by eliminating hotspots, replication can lead to a better distribution of the communication load over the network links. Besides performance-related reasons, replication improves system *availability*, since the larger the number of copies of an item, the more site failures can be tolerated. Some of the questions that need to be resolved in replication are: how many replicas should be created, when replicas are created and where are they placed. In case of systems with dynamic content, an extra issue is how the different replicas of each data item remain consistent with each other, so that accessing stale data items is avoided.

1.2. Scope of Thesis

In this thesis, we focus on replication in unstructured p2p systems. Since the topology of the overlay network in unstructured p2p systems seems to follow a power-law distribution pattern, our first objective is to examine what are the structural properties of power-law graphs that make them suitable for modeling an overlay network topology and how can we generate power-law graphs for simulation studies. We then present various approaches that have been proposed for creating replicas that aim at achieving *optimal replication*, whereby the expected number of peers that are probed during each search (*expected search size*) is minimized. The scope of this thesis is to investigate the effects of the overlay network topology (especially regarding random and power-law topology) on those methods. Based on our observations, we next propose a new replication method that considers the power-law property of the overlay network, in order to reduce the communication cost. We also focus on the problem of consistency maintenance in case of updates. Particularly, our aim is to investigate the effects of the overlay network topology on some known consistency maintenance protocols and propose a new update policy that is intended for p2p systems with power-law overlay network topology. Finally, we propose a different approach for maintaining consistency which adjusts the traditional quorum consensus to the distributed, dynamic environment of p2p systems.

1.3. Thesis outline

The remainder of this work is structured as follows. Chapter 2 summarizes some basic concepts regarding the topology of an overlay network. Particularly, the power-law topology observed in real p2p systems is described along with a theoretical analysis that is followed by description of methods for generating power-law graph topologies for simulation study. Chapter 3 focuses on the basic operation of a p2p network: search. We describe the alternative flood-based techniques that have been proposed for locating data items and provide a brief summary of the main issues about replication in unstructured p2p systems. We also present a few approaches that are proposed for creating a data item. In Chapter 4, we compare the performance of the described replication methods on networks with random and power-law topology and present experimental results to investigate whether they result in optimal replication. We then present a degree-based replication strategy that achieves lower communication cost. In Chapter 5, we discuss the problem of maintaining the consistency of replicas and investigate how existing update policies perform on networks with random and power-law topology. We also propose an alternative update policy for p2p networks with power-law overlay topology that decreases the massage overhead caused by peers with high degree. Based on quorum consensus traditional consistency maintenance technique, we present an adaptive update policy whereby the cost of updating a data item is decreased at the expense of making data item accesses more "expensive". In Chapter 6, we present the related work concerning replication and consistency maintenance in unstructured p2p systems. A brief summary of different approaches with different goals for creating and updating data items is provided. Finally, Chapter 7 concludes this thesis and presents the open issues for future work.

CHAPTER 2. MODELING AN OVERLAY NETWORK

In p2p networks, autonomous peers, who may join and leave the network at any time, share data with each other. Since these networks are usually very large and highly dynamic, each peer only stores the IP addresses for a selected subset of peers and other peers are reached via these neighbors. This way, peers form an *overlay network* that is built on top of the physical one.

In unstructured p2p systems, peers join the network by selecting a peer from a known list of participating peers. The selection of a peer from the list can either be random or based on some loose rules. The overlay network is formed in a decentralized manner as peers join and leave the network and there is no precise control over the topology of the resulting overlay network or over data placement. A look-up query for data items posed at any peer is routed through the overlay network according to the strategies that will be discussed in Chapter 3. The performance of a search strategy highly depends on the topology of the overlay network. Thus, an appropriate model of the overlay network topology is necessary for evaluating a search method. The choice of a model for the overlay network may also have strong implications on the analysis of some replication objectives (e.g. determining an optimal number of replicas).

The overlay network is represented as an undirected graph $G = (V,E)$. The set V of vertices contains one vertex for each participating peer. A pair of vertices belongs to the set E of edges if and only if the two corresponding peers are neighbors in the overlay network.

Some structural properties of the graph used to model the overlay network can help us predict the behavior of a network under certain assumptions. For example, the number of links with other nodes, the so-called *degree* of a node, is an indication of how frequently this node tends to be visited. Several measures regarding the graph's connectivity or the centrality of each vertex or the graph's diameter can be used to address the following questions:

- Is the network connected?
- Is it resilient to link or node failures?
- How easy is it for the network to break down to smaller pieces and which links would damage the connectivity of the network if they were removed?
- Are there any peers that receive more messages during a look-up process than others?
- What is the expected average number of hops that a look-up query is forwarded until the desired data item is located?

The rest of this chapter is organized as follows: Section 2.1 analyses some topological measures of graphs that are used to estimate the centrality of nodes in a network graph. In Section 2.2, a solution to the problem of finding the shortest paths between any pair of nodes is presented. In Section 2.3, we explain why power-law graphs are useful for modeling the topology of unstructured p2p overlay networks and define some power-law distributions that a power-law graph should follow. In Section 2.4, we present three known algorithms for generating undirected graphs with the defined power-law distributions. In Section 2.5, we present variations of two of these algorithms that generate connected graphs. In Section 2.6, simulation results are shown for estimating the suitability of these algorithms for generating graphs that obey power-laws.

2.1. Centrality measures

Usually, a graph consisting of n vertices is represented by an nxn matrix A, named the *adjacency matrix*, which is defined as:

$$A_{ij} = \begin{cases} 1 \text{ if there is an edge between vertices } i \text{ and } j \\ 0 \text{ if there is no edge between vertices } i \text{ and } j \end{cases} \quad (2.1)$$

Information about the relative importance of a vertex or an edge in a graph is obtained through some centrality measures.

The most common centrality measure is the degree d_i of a vertex i, which is defined as the number of links that i has. The *degree centrality* can be easily measured for a vertex i from the adjacency matrix as:

$$d_i = \sum_{j=1}^{n} A_{ij} \quad (2.2)$$

The idea behind the degree centrality is that the more connections a node has the more central and important is considered for the network.

Based on the same idea, there is another centrality measure, called *eigenvector centrality*. The difference is that in this measure the centrality of a vertex also depends on the centrality of its *neighbors* (the set of vertices for which an edge exists). In general, vertices that are connected to a central vertex are considered more central than vertices that are connected to less important vertices.

If we assign each vertex i a centrality x_i, this centrality is taken to be the average of the centralities of the neighbors of i:

$$x_i = \frac{1}{\lambda} \sum_{j=1}^{n} A_{ij} \, x_j \quad (2.3)$$

where λ is a constant. Defining the vector x as the vector of the centralities for all vertices, equation (2.3) becomes:

$$\lambda x = Ax \quad (2.4)$$

By the equation (2.4), it is clear that x is an eigenvector of the adjacency matrix A corresponding to the eigenvalue λ. Especially for graphs that represent a network topology, x should contain non-negative centralities. Thus, it is proved by the *Perron-Frobenius theorem* [13] that λ must be the largest eigenvalue of A and x the corresponding eigenvector.

In a graph, a *path* between two vertices is defined as a sequence of vertices such that for each vertex in the path there is an edge to the next vertex in the sequence. The *length of a path* is defined as the number of vertices (edges) it contains. The *shortest path* between a pair of vertices is the path between the two vertices with the smallest length. The length of this shortest path is known as the *distance* between them.

The centrality of a vertex can also be measured based on the shortest paths from every vertex to every other vertex. The *closeness centrality* of a vertex i is defined as the average length of every shortest path from vertex i to every other vertex j. Intuitively, the average distances of a vertex with high centrality to every other vertex should be lower compared to vertices with lower centrality. Accordingly, the *betweeness centrality* measures the importance of a vertex by counting the number of shortest paths between all pairs of vertices that it is part of.

2.2. All pairs shortest paths problem

As mentioned before, finding the shortest paths between all pairs of vertices in a graph that represents an overlay network topology can be very useful for describing the behavior of a network. The length of the maximum shortest path among the shortest paths of every pair of vertices is called the *diameter* of the graph.

The calculation of the diameter of a graph requires the solution of a more general problem, known as the *all pairs shortest path problem*, in order to obtain the set of distances between all pairs of vertices and find the maximum among them.

More formally, given a graph G with a set V of n vertices and a set of E of edges, the all pairs shortest paths problem is defined as finding the length of the shortest path $dist(i,j)$ between every pair (i,j) of vertices in V.

An obvious solution to the all pairs shortest paths problem is running one of the known shortest path algorithms, such as *Dijkstra's algorithm*, that computes the shortest path lengths from one vertex to every other vertex, n times, once for every vertex. At each run the distances of one vertex to every other vertex is computed. It is clear that the complexity of such an approach depends on the single-source shortest path algorithm that is used.

An alternative solution is using dynamic programming instead of a single-source shortest path algorithm. For computing the distance $dist(i,j)$ between all pairs (i,j) of vertices, this approach computes the distance $dist(i,z)$ between every possible intermediate vertex z and then add the last edge from z to j. The $dist(i,j)$ could then be defined as $\min_z(dist(i,z) + 1)$ for all vertices z for which an edge (z,j) exists. We define the $dist(i,j,t)$ as the length of the shortest path from i to j that uses at most t vertices. If there is no shortest path of length at most t between i and j, the $dist(i,j,t)$ is infinity. At each iteration t, all shortest paths of length at most t are computed. A shortest path between a pair of vertices cannot contain more than $|V|$ vertices (otherwise there would be a cycle). Thus, at iteration $|V|$ the distances between all pairs of vertices will have been computed.

The recursive relationship that calculates all distances $dist(i,j)$ at iteration t is:

$$dist(i,j,1) = \begin{cases} 0 \text{ if } i=j \\ 1 \text{ if } i \neq j \text{ and } (i,j) \in E \\ \infty \text{ if } i \neq j \text{ and } (i,j) \notin E \end{cases} \tag{2.5}$$

$$dist(i,j,t) = \min_{k \in V}\{dist(i,j,t-1), dist(i,k,t-1) + dist(k,j,1)\} \tag{2.6}$$

Algorithm 2.1 presents the algorithm used for determining the diameter of a graph.

Diameter calculation algorithm

```
Input:
      V  //the set of vertices
      E  //the set of edges
      n  //number of vertices
Output:
      int dist //the matrix that contains all distances from every
             node  to every other node
      int diam //the diameter of the network
Variables:
      prev_dist   //a matrix that contains that contains all
distances from every node to every other node that are known at
the beginning of each iteration t(distances found at iteration t-1)
//initially (within a distance of 1 hop)
  for every node i∈V
    for every node j∈V
     if(i == j)
         dist[i][j] = 0;
         prev_dist[i][j] = 0;
     else if (i,j) ∈ E    //there is an edge between i and j
         dist[i][j] = 1;
         prev_dist[i][j] = 1;
     else
         dist[i][j] = ∞
         prev_dist[i][j] = ∞
     end if
    end for
  end for

  t = 1
  while t<n-1  //find shortest paths of length t
     for every node i∈V
       for every node  j∈V
        dist[i][j] = ∞
        for every node k∈V
          if i ≠ j and dist[i][j] > prev_dist[i][k]+prev_dist[k][j]
               dist[i][j] = prev_dist[i][k]+prev_dist[k][j]
          end if
        end for
```

```
            prev_dist[i][j] = dist[i][j];
        end for
      end for
      t = t+1
  end while
  //dist now contains the minimum distances of each pair of vertices
 //the maximum between all minimum distances within diam hops
  diam = 0
  for every node i∈V
    for every node j∈V
         if(dist[i][j] > diam)
            diam = dist[i][j]
        end if
    end for
  end for
```

Algorithm 2.1: A dynamic algorithm for calculating the diameter of a graph

2.3. Power-law networks

2.3.1. Power-law networks as a model for the overlay network

Unstructured topologies evolve in more or less unpredictable ways, as nodes leave and join the overlay at arbitrary positions. Therefore, random graph, or *Erdős-Rényi random graph* is commonly used for modeling an overlay network. An Erdős-Rényi graph [10] with n nodes can be equivalently described in two ways: it is a graph where each of the possible $\frac{n(n-1)}{2}$ edges is present with some fixed probability p; or equivalently it is a graph selected uniformly at random among all possible graphs of n nodes and m edges. This is a simple but powerful model, where it can be shown that the degrees of participating nodes follow a Poisson distribution.

However, the Erdős-Rényi random graph is proved to be inappropriate for modeling real-world networks. There are some properties of several real-world networks such as Internet overlay network or the overlay network of an unstructured p2p system (described in [3]) that the Erdős-Rényi random graph cannot capture:

- Nodes that exist in the network for a long time tend to increase their connectivity rapidly with the addition of new nodes (*incremental growth*)
- Nodes with a high connectivity tend to increase their connectivity as new nodes enter the system (*rich-gets-richer phenomenon*)
- A node that joins the network is most likely connected to a well-connected node that already exists in the network (*preferential attachment*).

A more realistic model for the overlay network of an unstructured p2p system is the *power-law random graph*. In power-law graphs, the degree of each node follows a power-law distribution with a few nodes having many connections while the majority of nodes have only a few connections. Figure 2.1 illustrates a possible structure of a network with power-law degree distribution.

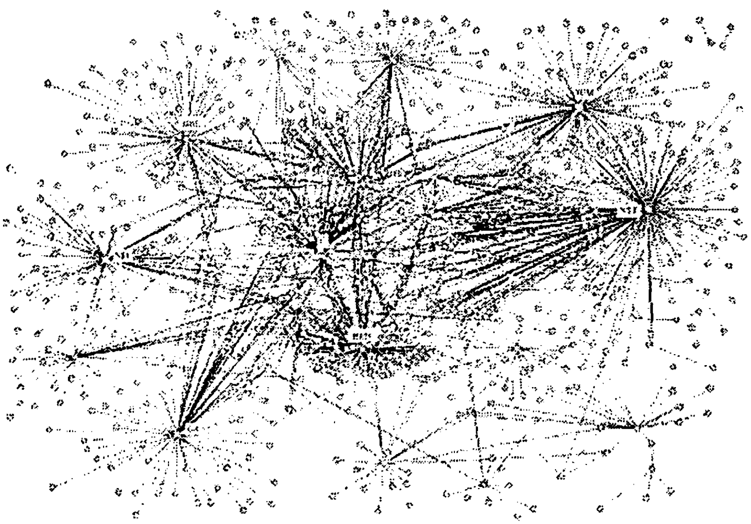

Figure 2.1: An example structure of a power-law network
(source: http://www.mathaware.org/mam/04/essays)

The existence of nodes with degree that greatly exceeds the average (usually referred to as *hubs*) makes all power-law networks share some properties. First of all, power-law networks exhibit a fault-tolerant behavior. This is due to the fact that failures occur in the network arbitrary with all nodes being equally likely to be affected by a failure. Thus, since there are only a few nodes that are considered as hubs and the vast majority of nodes are of law connectivity, the probability that a hub fails is not significant. Furthermore, even if a hub collapses, there are other hubs to guarantee that the network will remain connected. This means that power-law networks are tolerant of a small number of random failures. On the other hand, it is clear that in case of simultaneous failures of all the hubs of the network, the connectedness of the network would be lost and what would remain are numerous networks with no links between them. This weakness of power-law networks can be used by adversary users who can make a network fall apart by simultaneously attacking to all its hubs, causing them to collapse.

Another characteristic of power-law networks is that nodes tend to form small sub-networks (sub-graphs) where every node is connected to every other node in the sub-network and those sub-networks are connected to each other through hubs. For example, considering a social network, people tend to form small communities where everyone knows everyone (a friend of my friend is also my friend) and one community is connected to another only through some very popular people (such as politicians or famous artists). This phenomenon caused by nodes with high connectivity is known as *small-world phenomenon* and is responsible for the fact that power-law networks have very small average distance between two nodes and consequently a small diameter.

2.3.2. Power-law definitions

Faloutsos et al [11] have studied the topological metrics described in Sections 2.1 and 2.2 for random power-law networks and defined four power-laws regarding the degree of nodes, the degree frequency, the eigenvalues of the network graph and the number of shortest paths of certain length:

Power-law 1: rank exponent R: If all nodes are sorted in descending order of degree, the *rank* r_i of the node is the index of the node in this order. The degree, d_i, of a node i, is proportional to the rank of the node, r_i, to the power of a constant, R: $d_i \propto r_i^R$.

This power law indicates that the most central nodes (the hubs) have much more connections than the other nodes. The most connected the node is, the largest degree it has.

Power-law 2: degree exponent O: The frequency, f_d, of a degree d, defined as the number of nodes with degree d, is proportional to the degree to the power of a constant, O: $f_d \propto d^O$.

The intuition behind this power-law is that there exist only a few nodes with a high degree while there are many nodes with a small degree.

Power-law 3: hop-plot exponent H: The total number of pairs of nodes, $P(h)$, within distance of h hops, is proportional to the number of hops h to the power of a constant, H: $P(h) \propto h^H$ $with\ 0 < h < diam$, where $diam$ is the diameter of the network.

According to this power-law, many pairs of nodes are within a few hops (have small distance) and only a few nodes have a larger distance. This fact explains why power-law networks have small diameter (small-world phenomenon).

Power-law 4: eigenvalue exponent $\mathcal{E}$: If the eigenvalues of the adjacency matrix are ordered in descending order, the rank v_i of an eigenvalue λ_i is its index in the order. The eigenvalues, λ_i, of a graph are proportional to the order, v_i to the power of a constant , $\mathcal{E}$: $\lambda_i \propto {v_i}^{\mathcal{E}}$.

2.4. Generating network topologies that obey power-laws

2.4.1. Barabási model

Since undirected power-law graphs are used for modeling an unstructured p2p overlay network, an interesting issue is how can we generate a power-law network topology so as to be used in network simulation studies.

This problem has been addressed in [3] by Barabasi et al. The authors present a topology generator that obeys the four power-laws defined in Section 2.3.2. In particular, the model is based on the concepts of incremental growth and preferential attachment (Algorithm 2.2).

The model assumes an initial (typically small) N_0 number of nodes that form the network with no connections between them. At every timestep (periodically), until the desirable number of nodes n is attained, a new vertex (node) is added. The new node is connected to m different vertices that already exist in the network. The connections that are established are undirected. The node that joins the network chooses a node i as its neighbor with some probability which depends on the degree d_i of node i: the probability $\Pi(i)$ that the new coming vertex is connected to vertex i is defined as

$$\Pi(i) = \frac{d_i}{\sum_{j=1}^{net\_size} d_j} \tag{2.7}$$

where net_size is the number of nodes that participate in the network at each timestep.

Barabási model

```
Input:
        n   //the number of nodes of the final network
        N0  //the initial number of nodes
        m   //the number of edges with which each node is
            //connected when it joins the network
Output:
        A network topology that obeys power law
Variables:
        int A //the adjacency matrix representing the network
        int net_size = N0  //current network size
        int Num_connections = 0 //current number of connection of
                                //the newly inserted node
        int d // the matrix of degrees of nodes
       //Initially, the network contains N0 nodes and no edges
       while net_size < n
          //Add a new node u to the network
           Num_connections = 0
          //connect the new node to m other nodes of the network
          //with preferential attachment
```

```
        while Num_connections < m
            //choose a node i from the network at random
            //and add a connection between the new node u
            //and node i with probability Π(i)
```

$$\Pi(i) = \frac{d_i}{\sum_{j=1}^{net\_size} d_j}$$

```
                //add a connection between nodes u and i
                Num_connections = Num_connections+1
                d[i] = d[i] + 1
                d[u] = d[u] + 1
                A(i,u) = A(u,i) = 1 //add edge (i,u)
         end while
         net_size = net_size + 1
    end while
```

Algorithm 2.2: Barabási algorithm for generating a power law network

2.4.2. Power-Law Out-Degree Algorithm

C. Palmer and J. Steffan [21] have proposed an alternative method for generating network topologies that obey power laws: the Power-Law Out-Degree Algorithm.

According to *Power-Law Out-Degree Algorithm* (*PLOD Algorithm*) (Algorithm 2.3), each node i is initially assigned an number of degree credits, $credit_i$. The number of links (d_i) that a node is allowed to have in the final network is picked from an exponential distribution $\beta x^{-\alpha}$. Particularly, if the nodes of the graph are sorted in descending order of degree, the degree of the node whose rank in this ordering is x, will be $\beta x^{-\alpha}$ in the produced graph. After assigning a degree credit to each node, a pair of nodes ($n1$,$n2$) is randomly chosen and an undirected connection between them is established only if a connection between them does not already exist and neither of the nodes exceeds the degree credits that it was initially assigned. The above procedure is repeated until every node is left with zero credits. The α parameter is the exponent of the power-law distribution and determines the number of edges that should be added during the generation of the graph so that power-law distribution is achieved. It also represents the slope of the log-log plot.

PLOD algorithm

```
Input:
        n    //the number of nodes of the  network
        α,β //parameters of the exponential distribution
Output:
        A network topology that obeys power law
Variables:
        int A //the adjacency matrix representing the network
        int cur_edges = 0    //the number of edges  placed to
                             //the graph so far
        int d        //the matrix of degrees of nodes
        int credit   //the matrix of degree credits of nodes

//assign a number of credits to each peer picked from an
exponential distribution
       for i=1 to n
            x = random(1,n);
            credit[i] = βx^-α
       end for

       while there is a node with non-zero credits
           //choose a pair of nodes uniformly at random
           n1 = random(1,n);
           n2 = random(1,n);

           if n1 and n2 are not connected and credits[n1] > 0
              and credits[n2] >0
                    A(n1,n2) = A(n2,n1) = 1 //add edge (n1,n2)
                    d[n1] = d[n1] + 1
                    credit[n1] = credit[n1] - 1
                    d[n2] = d[n2] + 1
                    credit[n2] = credit[n2] - 1
                    cur_edges = cur_edges+1
           end if
     end while
```

Algorithm 2.3: Power- Law Out_Degree algorithm for generating a power law network

2.4.3. Recursive Algorithm

Palmer and Steffan in [21] also presented a method for generating topologies with power-law properties (Algorithm 2.4) that is based on the idea of generating an 80-20 distribution and can only be used for generating weighted undirected power law graphs of size which is a power of 2: an $2^n x 2^n$ adjacency matrix of a network is divided into 4 $2^{n-1} x 2^{n-1}$ sub-matrices. A distribution function can be defined so that a certain percentage of the final edges will be placed at each sub-matrix. Thus, we define a distribution of the following form: with probability *ul* an edge will be placed in the upper left sub-matrix A_{11}, with probability *ur* an edge will be placed in the upper right sub-matrix A_{12}, with probability *ll* an edge will be placed in the lower left sub-matrix A_{21} and with probability *lr* an edge will be placed in the lower right sub-matrix A_{22}. It should be clear that, as links are undirected, the adjacency matrix is symmetric with zeros on its diagonal. Therefore, the definition of the distribution depends on the number of non-zero elements that can potentially be placed in each sub-matrix.

By following such a distribution, the algorithm recursively picks one sub-matrix, splits that sub-matrix into 4 sub-matrices and picks one of these sub-matrices. This process goes on until an edge is finally returned. Each time an edge is returned, its weight in the adjacency matrix is incremented. This way, a symmetric weighted adjacency matrix is generated. The algorithm stops when *m* edges are added in the network.

Recursive algorithm

```
Input:
        n   //the number of nodes of the  network
        m  //the number of edges with which is node is connected
        ul,ur,ll,lr //the percentage of edges that will be placed
                    //in each submatrix
Output:
         A network topology that obeys power law
Variables:
         int A //the adjacency matrix representing the network
//Initially A contains zeros
```

```
while A has less than m non-zero elements
        //arbitrary choose two nodes to connect
        (i,j) = gensym(n)
        A(i,j) = A(i,j) + 1
end while

Function gensym(n)
   Variables:
    N1 = (n/2)^2 - (n/2) //the number of potentially non-zeros
                         //in the  upper- left lower-right sub-matrices
    N2 = (n/2)^2
    k = 1/((1/N1)ul + (1/N2)(ur+1) +(1/N1)lr)

   //We consider the 4 sub-matrices of A: A11 A12 A21 A22
      if n= 2 return (1,2)
      else
         with probability
           ul κ/N1          return  gensym(n/2)             //choose A11
           (ur + ll) κ/N2   return  gen(n/2) + (0,n/2) //choose A12 or A21
           lr κ/N1          return  gen(n/2) + (n/2, n/2) //choose A22
     end if

Function gen(n)
     if n = 1 return (1,1)
     else
        with probability
                   ul       return  gen(n/2)
                   ur       return  gen(n/2) + (0,n/2)
                   ll       return  gen(n/2) + (n/2,0)
                   lr       return  gen(n/2) + (n/2, n/2)
     end if
```

Algorithm 2.4: Recursive algorithm for generating a power law network

2.5. Generating connected graphs with power-law properties

For our simulation studies, we need to generate graphs that are connected. The graphs generated by Barabási (Section 2.4.1) and PLOD (Section 2.4.2) algorithms are not always connected. For this reason, we use a variation of the PLOD algorithm that is proposed in [17] in order to generate connected graphs with power-law properties.

In order to generate a connected graph, we first create a spanning tree. The algorithm used for generating a spanning tree is illustrated in Algorithm 2.5. We first generate a random permutation $a[1].....a[n]$ of all nodes. We then add edges to form a spanning tree. We begin with a tree consisting only of node $a[1]$ and no edges. At each step τ, we assume that nodes with indices $a[1],a[2]..a[t]$ in the random permutation are in the tree. We then add node with index $a[t+1]$ by randomly choosing one node $a[j]$ among $a[1],a[2]...a[t]$ $(0 \leq j \leq t)$ and adding an undirected edge between $a[t+1]$ and $a[j]$.

The PLOD algorithm (Algorithm 2.3) assigns a number of degree credits to each node picked from a power-law distribution and then a pair of nodes $(n1,n2)$ is selected and an edge is added between $n1$ and $n2$. This process is repeated until the desired number of edges is obtained. To ensure that eventually every node i will have a degree d_i equal to the number of credits $credit_i$ it was initially assigned and all nodes will have zero credits, a variation of the PLOD algorithm is proposed in [17]. The algorithm is presented in Algorithm 2.6: First a pair of vertices $n1$ and $n2$ of degree less than $credit_{n1}$ and $credit_{n2}$ respectively such that there is no edge between $n1$ and $n2$ is chosen. If such a pair of nodes exists, an edge between $n1$ and $n2$ is added. If no such pair of nodes exists, the fixup procedure is executed.

The fixup procedure works as follows: if all nodes except node u have non zero credits, an edge $(v1, v2)$ is selected such that $(u, v1)$ and $(u, v2)$ are not edges. Then we delete the edge $(v1, v2)$ and add edges $(u, v1)$ and $(u, v2)$. If more than one nodes have non zero credits, we assume that there are pairs of nodes each of non-zero credits, but each such pair is already connected by an edge. In that case, we find a pair of nodes $u1$ and $u2$ such that both nodes have non-zero credits and $(u1, u2)$ is an edge.

Then we choose an edge (*w1*, *w2*) such that (*u1*, *w1*) and (*u2*, *w2*) are not edges. Finally, we delete the edge (*w1*, *w2*) and add edges (*u1*, *w1*) and (*u2*, *w2*).

Creating a spanning tree

```
Function random_permutation(n)
Input:
        n   //the number of nodes of the  network
Output:
        a //a random permutation of peers from 1 to n

        for t: 1 to n
          a[t] := t
        end for

        for t: 1 to n
          j:= uniform_radom(1, n)
          swap(a[t], a[j])
        end for

Function create_spanning_tree(n)
Input:
        n   //the number of nodes of the  network
Output:
        int A  //a spanning tree of n nodes

    for τ: 2 to n
        r:= a[uniform_radom(1,τ-1)]
        //add an edge (r,τ)
        d[r] = d[r] + 1
        d[τ] = d[τ] + 1
        A(r,τ) = A(τ,r) = 1
    end for
```

Algorithm 2.5: Algorithm for creating a spanning tree

PLOD algorithm for connected graphs

```
Input:
     n    //the number of nodes of the  network
     α,β //parameters of the exponential distribution
Output:
     A network topology that obeys power law
Variables:
        int A    //the adjacency matrix representing the network
        int d        //the matrix of degrees of nodes
        int credit   //the matrix of degree credits of nodes

//assign a number of credits to each peer picked from an
//exponential distribution
       for i=1 to n
            x = random(1,n);
            credit[i] = βx^(-α)
       end for

       random_permutation(n);
       create_spanning_tree(n);

       while there is a node with non-zero credits
         if there is a pair of nodes with non-zero credits and
         no connection between them
            //choose a pair of nodes uniformly at random
            do
              n1 = random(1,n);
              n2 = random(1,n);
            while n1 and n2 are connected or credit[n1] ≤ 0
                  or credit[n2] ≤ 0

           A(n1,n2) = A(n2,n1) = 1; //And an edge between n1 and n2
           d[n1] = d[n1] + 1
           credit[n1] = credit[n1] - 1
           d[n2] = d[n2] + 1
           credit[n2] = credit[n2] - 1
         else call fix_up ()
         end if
     end while
```

```
Procedure fix_up()
    //check if there is only one node, u, with non-zero credits
    if all nodes except u have non-zero credits
         v1 = random(1,n);
         v2 = random(1,n);

        if v1,v2 are connected and (v1,u) and (u,v2) are
           not connected
           //remove edge (v1,v2) and add edges (v1,u) and (u,v2)
           d[u] = d[u] + 2
           credit[u] = credit[u] - 2
           A(v2,v1) = A(v1,v2) = 0 //remove edge (v1,v2)
           A(v1,u) = A(u,v1) = 1 //add edge (u,v1)
           A(v2,u) = A(u, v2) = 1 //add edge (u,v2)
        end if
    else
       u1 = random(1,n);
       u2 = random(1,n);

       if u1 and u2 are connected and credits[u1] > 0
          and credit[u2] >0

           w1 = random(1,n);
           w2 = random(1,n);

           if w1,w2 are connected and (u1,w1) and (u2,w2)
           are not connected
               A(w2,w1) = A(w1,w2) = 0 //remove edge (w1,w2)
               A(u1,w1) = A(w1,u1) = 1 //add edge (w1,u1)
               A(u2,w2) = A(w2,u2) = 1 //add edge (w2,u2)
               d[u1] = d[u1] + 1
               credit[u1] = credit[u1] - 1
               d[u2] = d[u2] + 1
               credit[u2] = credit[u2] - 1
           end if
     end if
   end if
```

Algorithm 2.6: PLOD algorithm for generating connected graphs

As far as the Barabási algorithm is concerned, each newly inserted node is connected to m of the nodes that already exist in the network. According to the model described by Barabási, we assume that initially there is a small number N_0 of nodes with no connections between them. Therefore, if each node selects less than N_0 nodes to connect with ($m<N_0$), there would be non-zero probability of never choosing some nodes and connect with another peer. Thus, to ensure that the generated graph is connected, we choose the values of parameters m and N_0 such that m is equal to N_0.

2.6. Estimating the quality of the synthetic graphs

Network topologies that obey power-laws should have the following four properties defined in Section 2.3.2: rank exponent, degree exponent, hop-plot exponent and eigenvalue exponent. In this section, we use the Barabási algorithm as well as the PLOD algorithm as described on Section 2.5 to generate connected network graphs with the above properties. We then perform experiments on the synthetic graphs to evaluate the extent to which the desired four power-law properties hold for the generated topologies.

The algorithms discussed in Section 2.5 for generating connected graphs with power-law properties were implemented in C++ using the OMNET++ simulation programming tool. OMNET++ (Objective Modular Network Testbed in C++) [20] is a modular, open-source discrete event network simulator. Each participating peer is simulated as an OMNET++ module and is implemented as a C++ object whose methods describe the expected behaviour of each peer. Modules can be connected through input and outputs gates and communicate by exchanging messages.

The topological metrics that need to be measured in order to estimate the quality of the synthetic undirected graphs are: the degree d_i of every node i in the network, the frequency f_d of an degree d that appears in the network, the distances $dist_{ijk}$ of any pair of nodes (i,j) within k hops, with k between 0 and the diameter of the network and the eigenvalues λ_i of the resulting graph.

First, we generate a network graph of size n = 3000 peers by using the model introduced by Barabási. We used various values for N_0 parameter. At each simulation the m parameter is chosen to be equal N_0. We also present experimental results for PLOD algorithm with various values for α parameter and $\beta = n^{0.66}$. Then we study the topological measures mentioned above. The algorithm parameters used for generating the synthetic graphs are summarized in Table 2.1.

Table 2.1: Simulation parameters for power-law graph generators

Parameter	Symbol	Value
Network size	n	3000
Initial number of peers for the Barabási model	N_0	[2-5]
Number of edges for each newly inserted peer in the Barabási model	m	[2-5]
α parameter of exponential distribution in PLOD algorithm	α	[0.4-1.5]
β parameter of exponential distribution in PLOD algorithm	β	$n^{0.66}$

First we study the degrees d_i of nodes in the generated network graph. A node's rank r_i is its index in the order of decreasing degree. In Figure 2.2 and Figure 2.3 we plot all (r_i, d_i) pairs in log-log scale. As it is shown on both figures, the (r_i, d_i) plots in log-log scale are approximated well by linear regression which indicates that the rank exponent property holds for the synthetic graphs generated either with Barabási's algorithm or PLOD. In the case of PLOD algorithm (Figure 2.3), we notice that in the resulting graph, the degree of the most connected node is determined by the value of β parameter ($n^{0.66} \approx 198$) while α parameter determines the total number of edges that are added. For example, for α = 1.5 there are much more nodes with few connections (equal or close to 1) than when α = 0.4.

In order to study whether the degree exponent holds, we study the distribution of degrees. Figure 2.4 and Figure 2.5 we plot the degrees d_i of the nodes in the undirected graph generated by Barabási's algorithm and PLOD respectively, with the frequency f_d of each degree d, measured as the number of nodes with degree d. Again, the plots are in log-log scale and observe a linear relationship, as degree exponents suggests. Consequently, we can claim that for both synthetic graphs that are generated, the property of degree exponent holds.

We then study the size of the neighborhood within some distance. The neighborhood $p(h)$ within some distance is defined as the number of pairs of peers with distance at most h. Figures 2.6 and 2.7 show the number of pairs $p(h)$ within distance h towards the distance h when Barabási's algorithm and PLOD respectively are used. The plots are in log-log scale. The plots show a linear relationship between $\log(p(h))$ and $\log(h)$ which implies that the hop-count exponent also holds for the generated graph topologies.

Finally, we study the eigenvalue of the synthetic graphs which are defined as the eigenvalues of the adjacency matrices A. For the calculation of eigenvalues the *eig* function of MATLAB programming tool was used. Among all eigenvalues, only the greatest have a physical meaning when it comes to adjacency matrix. For this reason, we plot only the 50 greatest eigenvalues λ_i in descending order in comparison to the index v_i of of the eigenvalue in the descending order. The plots are in log-log scale and as shown in Figure 2.8 and Figure 2.9, $\log(\lambda_i)$ and $\log(v_i)$ are proportional, so the claim that the synthetic graphs have the eigenvalue exponent property also holds.

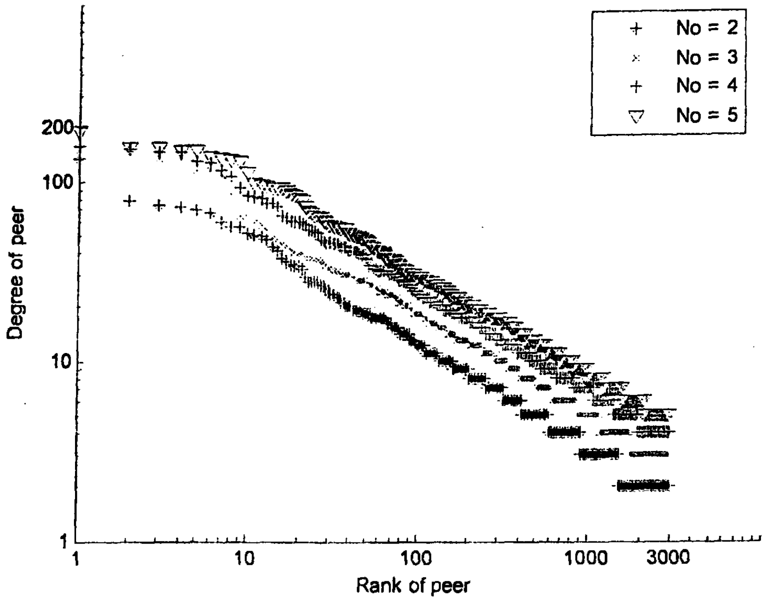

Figure 2.2: Rank exponent for the network topologies generated by Barabási model under various values for No parameter

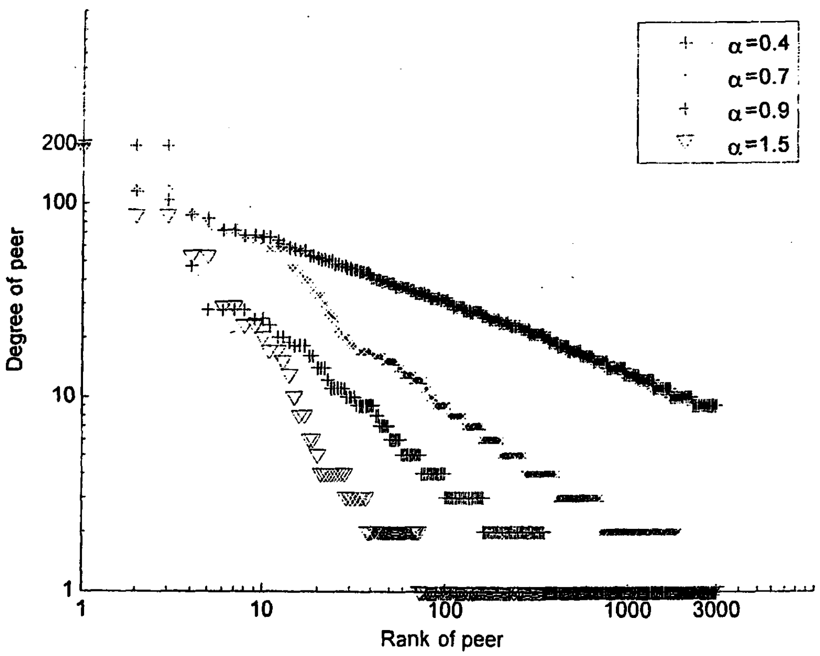

Figure 2.3: Rank exponent for the network topologies generated by PLOD algorithm under various values for α parameter

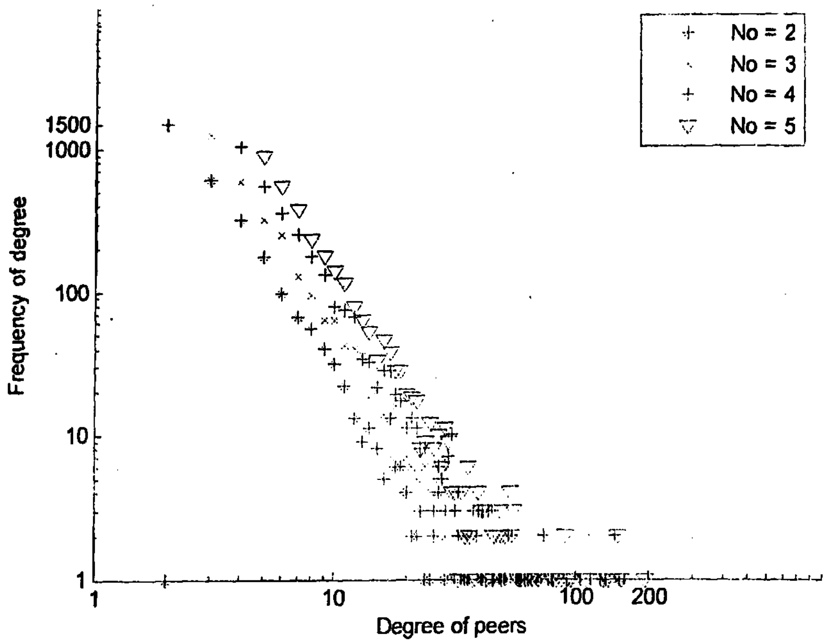

Figure 2.4: Degree exponent for the network topologies generated by Barabási model under various values for No parameter

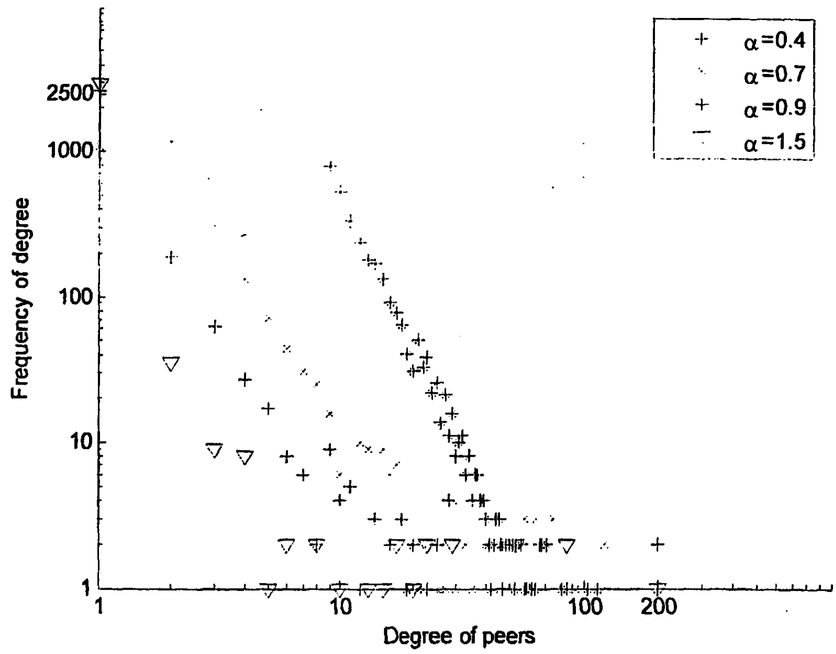

Figure 2.5: Degree exponent for the network topologies generated by PLOD algorithm under various values for α parameter

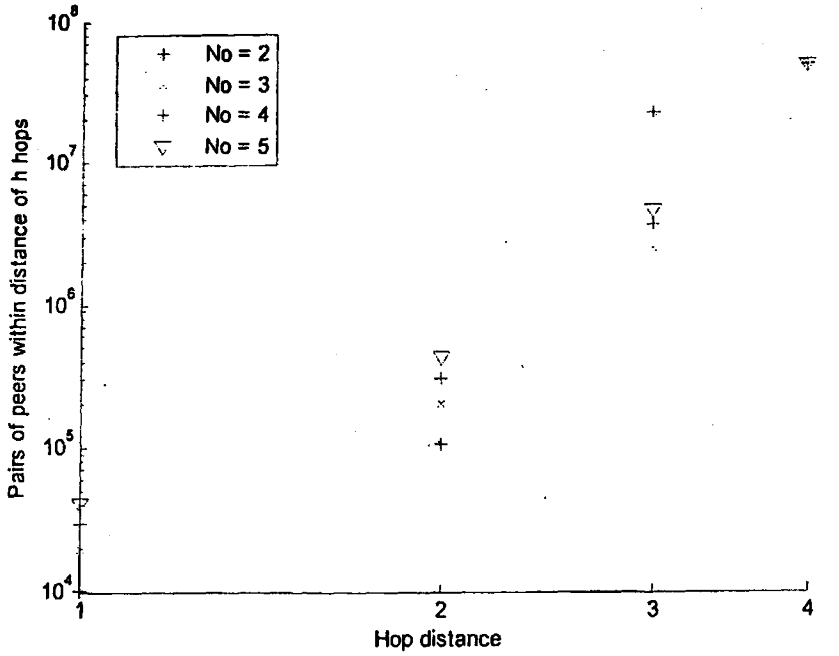

Figure 2.6: Hop-plot exponent for the network topologies generated by Barabási model under various values for No parameter

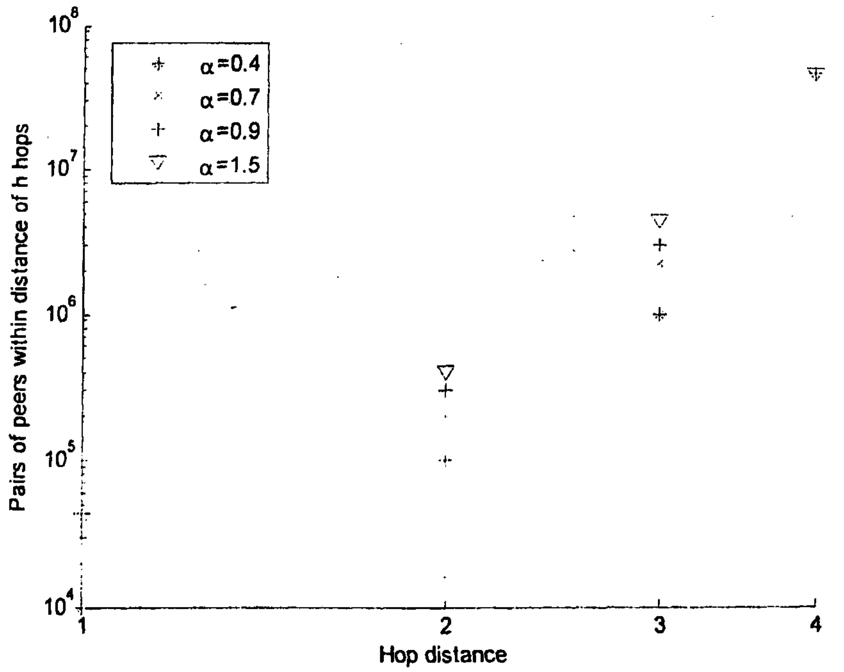

Figure 2.7: Hop-plot exponent for the network topologies generated by PLOD algorithm under various values for α parameter

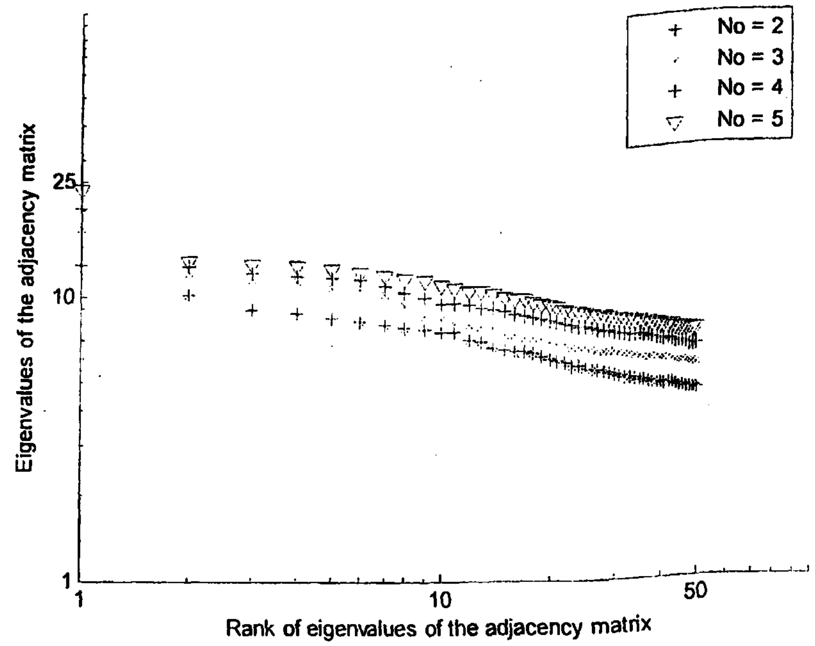

Figure 2.8: Eigenvalue exponent for the network topologies generated by Barabási model under various values for No parameter

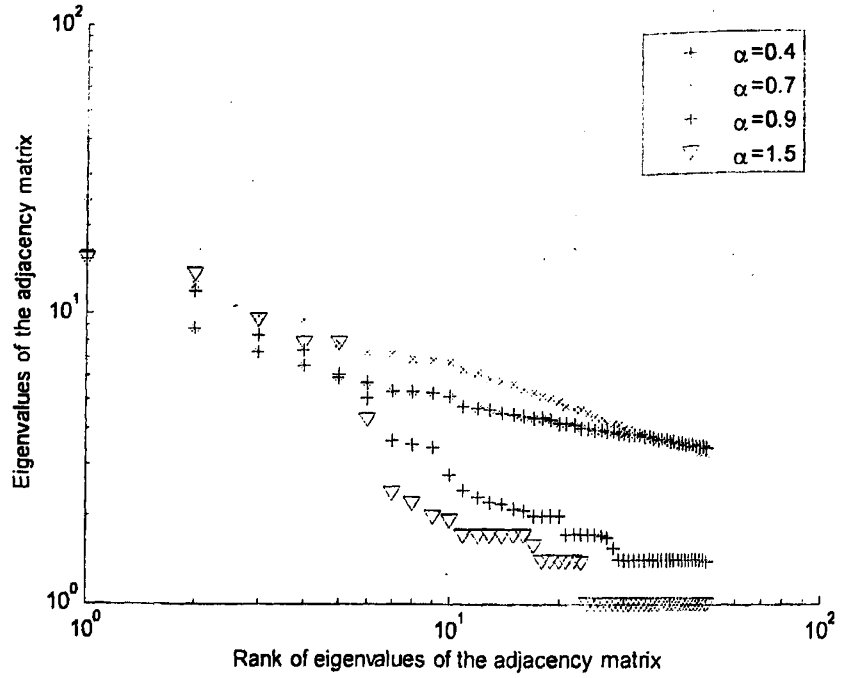

Figure 2.9: Eigenvalue exponent for the network topologies generated by PLOD algorithm under various values for α parameter

CHAPTER 3. SEARCH AND REPLICATION METHODS FOR UNSTRUCTURED P2P SYSTEMS

3.1. Locating data items of interest in unstructured P2P systems

Distributed p2p systems have attracted a lot of attention as a means of data sharing among a large and dynamic population of peers. Peers join and leave the system dynamically, thus forming self-organizing overlay networks. A basic functionality of p2p systems is discovering data items of interest. Any peer may ask to retrieve one of the shared data items. When a peer poses a query for a data item, it uses the overlay network to communicate with its neighbors and a look-up process is initiated for locating the peers that hold the requested data item. Locating a data item must be achieved by contacting as "small" a number of nodes in the overlay as possible and by maintaining as "little" state information at each node as possible.

To assist lookup, *structured overlays* map (keys of) data items to nodes. In structured overlays, the mapping is usually done by hashing the key space of the data items to the id space of nodes. Thus, each node in the overlay maintains a partition of the data space. In structured overlays, lookup reduces to locating the node in the overlay that is responsible for the corresponding data partition. In *unstructured overlays* on the other hand, there is no correlation between nodes and data items.

Therefore, "*blind*" search procedures are used and look-up queries get propagated through the network so as to locate peers offering the requested data item.

The most commonly used blind search strategy for unstructured p2p overlay networks is flooding. In *flooding*, a peer that wants to retrieve a specific data item initiates a look-up process by communicating with all of its neighbors in the overlay network. A peer that receives a look-up (query) message propagates it to all its neighbors, unless it knows about the data item in question. The look-up messages are allowed to be propagated until the data item is located or for a limited number of steps (hops), which is the so-called *time-to-live* (TTL) parameter. The TTL value is defined by the peer that initiated the look-up process and is included in the look-up message. Each intermediate peer that receives a look-up message decrements the TTL value by 1 and peers that observe a zero TTL value stop propagating the message any further. When the propagation of a message is terminated, a reply is forwarded back following the same path until it reaches the peer that initiated the look-up process.

Such a TTL-limited flooding has several shortcomings. First of all, defining an appropriate TTL value is not an easy task. The choice of a large TTL value may overload the network with look-up messages, while if a small TTL value is used, many look-up queries may be *unsuccessful*, as their propagation may be terminated before peers that hold the requested data item are located. Additionally, in flooding a peer may receive the same message more than once due to cycles in the path through which the messages are forwarded or because a peer may receive the same look-up message more that once from multiple neighbors (different paths). For example, Figure 3.1 illustrates a possible structure of an overlay p2p network, consisting of 7 peers. In this example overlay network, we assume that peer *A* wants to retrieve a data item that is held by peer *C* and uses flooding that is restricted to TTL = 3 hops for propagating its look-up query. The scenario for the propagation of the look-up message could be the following: First peer *A* forwards the message to its neighboring peers *D*, *B* and *F*. Then, peer *D* will continue forwarding to peers *E*, *B*, *A*, peer *B* will forward the message to peers *A*, *D*, *F*, *C*, *E*, *G* and peer *F* will forward to peers *A*, *B*, *G*. Finally, peer *C* that offers the desired data item would send a reply back to peer *B* and *B* would forward the reply back to peer *A*. All other peers, apart from *C* would continue forwarding the message to all of their neighbors as they are not aware that the data item has already been locating through another path. During this look-up process many duplicate messages overloaded the network with each peer receiving the look-up query from all its neighbors.

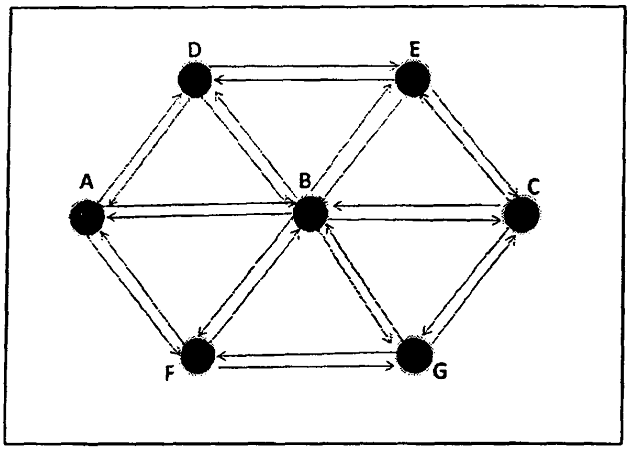

Figure 3.1: Flooding propagation in an unstructured overlay network

To reduce the number of messages produced during flooding, an alternative search technique is used: the *random walks method.* With random walks, each peer that receives a look-up message, if it does not know of the data item in question, it selects only one peer from its neighborhood and forwards the message to it instead of forwarding the message to all its neighbors. The selection of the neighbor to which the message will be forwarded can be uniformly random or biased according to some criteria. To further improve the performance of this method, multiple walkers can be deployed simultaneously. In the *k random walks method,* the requesting peer instead of selecting only one neighbor, it selects *k* neighbors and its query request is propagated through *k* different random walks. Returning to the example of the p2p network illustrated in Figure 3.1, if 3 random walkers with TTL = 3 were used instead of flooding the following scenario could improve the communication cost in comparison to flooding: Peer *A* forwards the message to peers *D*, *B* and *F* and peers *D, B, F* forward the message only to peers *E, C* and *G* respectively (not all of their neighbors). Peer *C* then sends a reply back to peer *F* while peers *E* and *G* randomly select one of their neighbors, say D and B respectively, and forward the message to them. It is obvious that this search strategy reduces the number of duplicate messages.

Another variation of *flooding* is the *random BFS* or *teeming* where each peer propagates the look-up message to each of its neighbors with some fixed probability φ. A decay parameter can be used so that φ decreases with the distance. This way, the probability that a look-up message is forwarded is very low if the message has already been forwarded for a few steps.

An improvement of k random walks that achieves termination of search when a data item is located at one of the different walkers is *random walks with checking* according to which every walker asks the peer that initiated the search whether the search was successful through some other path before propagating the look-up message.

Another search method that has been proposed is *expanding ring*. In expanding ring, a peer starts with a small TTL and floods the look-up message. If the search is not successful, the TTL value is increased and the flooding process is repeated until the data item is located or until a maximum TTL value is reached.

3.2. Creating replicas of data items in unstructured P2P systems

A commonly used approach in distributed P2P systems for improving the performance of a look-up process is caching or replication of either data items or search paths (or both).

Replication increases the number of copies for each shared piece of data in the system. By doing so, the probability that some or all the data is temporarily or permanently lost (because of a node departure or a link failure) significantly decreases, thus the *dependability* of the system in terms of *reliability* and *availability* is increased. Additionally, by having more copies for popular data items, the load for routing and answering queries can be evenly distributed among the servers that hold the copies. This way, the *performance* of a search method is improved in terms of *throughput* and *response time*, since congestions in "hot" servers may be avoided. While unstructured overlays which adopt flooding-based techniques are effective for locating popular data, they are poorly suited for locating rare data. Thus, by replicating the rare data, the probability of locating it during a search process increases, consequently increasing *data recall.*

Several issues are associated with replication. For example, a replication strategy should determine *what* should be replicated. There are two choices of what to replicate: actual data items or index entries (pointers) of the real data. If actual data items are replicated, the storage space required for holding the replicas is increased compared to replicating indices. Also, replicating indices does not improve reliability or availability since it does not lead to more physical copies. In addition, a replication strategy should define *how many* replicas of each data item should be stored in the network. Another issue that should be taken into consideration is *where* the replicas of data items should be placed. The number of replicas for each data item and where these replicas are placed can significantly affect the performance of the search method that is used for locating peers with specified data items. Moreover, an extra overhead is imposed by replication not only for storing the multiple copies of each data item, but also for *maintaining the consistency* of the different replicas that are kept and propagating the updates that may occur in one data item.

As far as the question of how many replicas of each data items should be created is concerned, there are two natural ways of replicating data items, namely uniform and proportional replication. In *uniform replication* (UR), the same number of replicas is created for each data item, regardless of how popular it is considered. The popularity of a data item x, also referred to as its *query rate* q_x, is defined as the probability that a peer poses a query for that data item. In *proportional replication* (PR) the number p_x of replicas for each data item is proportional to the popularity of the item. Although it seems natural to create more replicas for more popular data items so as to favor most common queries, this is done at the expense of rare ones.

For the random probes strategy, the search process proceeds until the data item is found or until a TTL value is exceeded. We consider a network consisting of n peers which share s different data items. For each data item x there are p_x copies in the network. Each peer has a storage space for holding up to c replicas of data items. Given that, for random graphs, the probability $Pr(r)$ of locating a data item after r probes is equal to the probability of not locating it in the previous r - 1 probes and locating it in the r-th probe and is given as

$$Pr(r) = \left(1 - \frac{p_x}{n}\right)^{r-1} * \frac{p_x}{n} \tag{3.1}$$

The *expected search size* (average number of probed peers) for locating a particular data item x is $ess_x = \frac{n}{p_x}$. Thus, the expected search size (ESS) for all data items (the average number of probed peers per query) is

$$ESS = \sum_{x=1}^{s} q_x * ess_x = n * \sum_{x=1}^{s} \frac{q_x}{p_x} \tag{3.2}$$

The replication schema that minimizes equation 3.2 is the *square root replication*, where the number of replicas for each data item is proportional to the square root of its query rate ($p_x \propto \sqrt{q_x}$).

However, things are different if the topology of the network is not random, but the degrees of the peers follow a non-uniform (power-law) distribution. For the random probes strategy it holds that

- the choice of the next peer does not depend on the previous peer (*memoryless random walks*)
- the probability of visiting a particular peer is proportional to its degree

Under these assumptions, for networks with power-law topology, the probability $Pr(r)$ of locating a data item after r probes is

$$Pr(r) = \left(1 - \frac{D_{S_x}}{D}\right)^{r-1} * \frac{D_{S_x}}{D} \tag{3.3}$$

where D_{Sx} is the sum of the degrees of peers that belong to the set S_x of peers that hold a replica of data item x and D is the total sum of degrees of all peers. Thus, the expected search size for locating data item x is $ess_x = \frac{D}{D_{S_x}}$, and the overall expected search size is

$$ESS = \sum_{x=1}^{s} q_x * ess_x = D * \sum_{x=1}^{s} \frac{q_x}{D_{s_x}} \quad (3.4)$$

If all peers of the network have the same capacity c, then the total number of replicas R is equal to $c * n$ ($R = c * n$) and the total sum of degrees for all data items $\sum_{x=1}^{s} D_{s_x}$ is equal to $c * D$ ($\sum_{x=1}^{s} D_{s_x} = c * D$) because the degree of each node is counted once for each replica it holds. Thus, the quantity $\sum_{x=1}^{s} D_{s_x}$ is limited to a constant: $D * \frac{R}{n}$ and the equation (3.4) is minimized when $p_x \propto \sqrt{D_{s_x}}$.

Various replication strategies have been proposed for achieving SR replication. In *owner replication*, which is used in Gnutella, when a search for a data item is successful the peer that initiated the search process (the requester peer) stores a replica of the data item. In *path replication*, which is used in Freenet, each query keeps track of the path it follows starting from the requester peer. When a search succeeds, all peers that exist in the path from the requester peer to the peer that provides the data item are forced to keep a replica of the data item. When k random walk strategy is used, the number of peers that are in the path from the requester peer to the provider peer is expected to be *1/k* of the total peers that where probed during the search. Since path replication creates for each item a number of replicas that is proportional to the search size for locating it, it should result in square-root replication. Path replication has the drawback that it tends to create replicas of data items to peers that are topologically along the same path, which is not very effective. To overcome this problem, a third replication strategy, the random replication has been proposed. The *random replication* strategy counts the number of peers on the path between the requester and the provider peer, say p. Then p of the nodes that the k walkers visited are randomly selected to replicate the data item.

A replication strategy that is based on random probes is presented in [16], the *Pull-then-Push (PtP)* replication. When a peer issues a request for a data item, first it checks if it possesses the data item. If the peer does not possess the data item, the request is propagated through the network following the k-random walk strategy (*pull-phase*): the requesting peer forwards the request to k of its neighbors and each other peer that receives the requests randomly picks one of its neighbors and forwards the request. The propagation goes on either until the data item is found or until it has been propagated for more than TTL hops. As peers are probed, along with the request, information about the path that is followed from the requesting peer is also propagated. When a data item is found, a reply is sent back to the requesting peer by reversing the path from the requesting to the provider peer. After a successful search, the requestor enters a replication (*push-phase*) where peers are randomly probed and forced to hold a replica of the data item. During the push-phase the same strategy as in the pull phase is used and the TTL value is set to the number of hops for locating the data item minus one hop. This way, the number T_x of replicas that is created after a successful search for x is approximately the same as the number of probed peers during the search process (#copies = # probed peers), which leads to a square root replication.

CHAPTER 4. REPLICATION ON UNSTRUCTURED P2P SYSTEMS WITH A POWER-LAW OVERLAY NETWORK TOPOLOGY

The overall effectiveness of a replication strategy that is used in a p2p system is heavily dependent on the topology of the overlay network. In order to investigate the effect of the overlay network topology (especially the topology of power-law characteristics) on the replication methods presented in Chapter 3, we have performed a series of experiments under a simulation environment. In Section 4.1, we describe the simulation environment and present our experimental results. In Section 4.2, based on some observations that derive from the simulation results, we present a replication strategy that considers the characteristics of power-law and reduces the total communication cost without increasing the average number of hops (*average depth_found*) for locating a replica.

4.1 Influence of Power-law overlay topology to replication strategies

4.1.1. Simulation Environment

The replication methods discussed in Section 3.2 were implemented in C++ using the OMNET++ simulation programming tool.

Network model: For our simulation study, we consider networks that have random and power-law topologies. For generating networks with power-law properties, we have used the Barabási model as described in Section 2.4.1 (Algorithm 2.2). The PLOD algorithm for connected undirected graphs (Algorithm 2.6) could have as well been used for generating a network graph with power-law topology. Since the generated graphs for both algorithms have very similar (practically the same) topological properties (as shown in Section 2.6), it makes no difference which of the two implemented methods will be used.

For the random (Erdős-Rényi) graph we assume that there are n peers in the final network and the average degree of each peer is *davg*. The network is generated according to the following process: first a spanning tree consisting of n peers and n-1 edges is created as described in Section 2.5 (Algorithm 2.5). Then, the rest of the edges are placed by randomly selecting a pair of peers $n1$ and $n2$ and adding a bidirectional edge $(n1,n2)$ if there is not already one. This process is repeated until $\frac{davg*N}{2}$ edges are finally added.

Query model: A simulation starts by distributing the s distinct data items (simply indicated by an integer from 1 to s) randomly to the network. Then, each peer is periodically triggered through a self message and performs the *generate_query* process: The peer randomly chooses one of the s distinct data items and issues a query for that item. Each simulation is executed for 360 seconds (real simulation time). The frequency f with which each peer is triggered to generate a query (*generate query rate*) is also measured in seconds and is given as a simulation parameter. The probability with which each data item is chosen (*query distribution or query rate*) follows a zipf distribution with a given *theta* value. This means that not all items are chosen with the same probability but there are some items that are more popular than others.

The search method that is used in the simulation is either flooding or the k random walkers where k is given as an input in the simulation. When a peer issues a request for a data item, first it checks its local cache. If the requested item is not offered locally, the requester peer chooses at random (uniformly) k different peers from its 1-hop neighborhood (or all of them, if flooding is used instead) and sends a *request* or *pull message* to them. This request message contains:

- A *sequence number* that is unique for each new query that is generated
- The *initiator*, which contains the id of the peer that issued the query
- The requested *data item*
- The whole *path* that the message traverses starting from the initiator
- The *hop counter* that counts the length of the path from the initiator to the previous hop.
- The *depth_found* that simply indicated the number of hops between the peer that offers the requested data item (provider) and the initiator. Until the desired data is located, the depth_found is set to -1.

When a peer receives a request message it first checks if it offers the data item in question. If the peer locates the requested data item in its cache, it sets the *depth_found* field equal to *hop counter* and sends a *reply message* to the peer from whom it received the request message (taken from the *path* field). The reply message has the same form as the request message. If the requested item is not in peer's cache, the peer checks if the *hop counter* does not exceed a certain value, defined by TTL parameter. Since a message is not forwarded more than TTL times, if the hop counter is greater than TTL, the message is not forwarded anymore. Otherwise, if k random walkers strategy is used, the peer chooses a neighbor at random and forwards the message to it and increments the *hop counter* by one, adding its own id to the path. In the case of flood, the peer forwards the message to all of each neighbor, not only one.

When a peer receives a *reply message*, if it is not the initiator of the query, it deletes itself from the path list and forwards the message to the peer in the path that comes next.

Replication model: Every peer has a local cache for storing a limited number of known data items. The capacity of a cache c is a user-defined parameter. When the cache of a peer is full and a new data item needs to be stored, a data item is randomly chosen from the cache and is deleted to make room for the new one. This cache replacement policy is generally known as *random deletion policy*.

In our simulations, after a successful search we create and distribute replicas of data items according to one of the following replication mechanisms: owner, path and pull-then-push replication (Section 3.2). In owner replication, when a peer receives a reply message for a query it has posed, it stores a replica of the data item in its cache, while in path replication a replica of the data item is stored in the cache of every peer that receives a reply message. In push-then-pull (PtP) replication strategy if the peer that initiated the search (*initiator*) receives a reply message for its query, it begins a push phase by sending *push messages* to k (or all) of its neighbors. The pull messages are of the same form as pull and reply messages. For the propagation of push messages we use the same strategy as for locating the data items, with the TTL value set equal to the value of the *depth_found* field minus one hops.

Summary: Depending on the needs of the simulation, at each experiment we need to define the topology of the network (random or power-law), the replication method used (owner, path or pull-then-push) and some parameters concerning the number of peers in the network, the number of shared data items or the search method that is used. Table 4.1 summarizes the parameters that are used in our simulations and their default values.

Table 4.1: Simulation parameters for replication strategies

Parameter	Symbol	Default value
Network size	n	5000
Number of data items	s	100
Random walkers	k	[5-8]
TTL	t	10
Generate query rate	f	0.005 (seconds)
Simulation time	T	360 (seconds)
Query Distribution (zipf's parameter)	*theta*	1.2
Capacity of each peer's cache	c	10
Average degree for Random topology	*davg*	4
Initial number of peers for the Barabási model	N_0	3
Number of edges for each newly inserted peer in the Barabási model	m	3

Output parameters: The metrics that are measured at each simulation are the following:

a. The *normalized replication ratio* for each data item x, which is computed as $\frac{r_x}{\sum_{y=1}^{s} r_y}$, where r_x is the total number of replicas that exist in the network for the particular data item x

b. For each data item x, *the sum of degrees* D_{Sx} of all peers that hold a replica of data item x

c. The *average depth_found* for each data item which is the average number of hops at which the item was located among all the successful queries for that data item.

d. The total *communication cost* which is the total number of messages that are exchanged during the simulation. The total number of messages is measured as the sum of messages that are forwarded for replica location (pull messages), for replying to a query when the data item is located (reply messages) or for the creation of new replicas (push messages)

Each experiment is performed 10 times and the results presented for each measure are the average of the 10 executions.

4.1.2. Simulation Results

In the first set of experiments, the owner, path and pull-then-push replication strategies were applied to networks with random and power-law topologies. The parameters that were used in the simulations are those defined on Table 4.1. The scope of the experiments is:

- to investigate whether the optimal replication, as described in Section 3.2, is attained by the replication strategies (considering that for random networks optimal replication is considered to be the square-root replication while for power-law networks optimal replication is achieved when the sum of degrees of all peers that hold a replica of a particular data item is proportional to the square root of its query rate)
- to investigate the effect that the topology of the overlay network has on the performance of the replication strategies

For PtP we have used various *k* random walkers and present the results for two of them: a 5 random walker with TTL = 10 and an 8 random walker with TTL = 10. We have also experimented with flooding.

In Figure 4.1, we present results for a network with random topology where the owner, path and PtP replication strategies were employed. Particularly, Figure 4.1 shows the normalized replication ratio of each data item in comparison to its query rate. The plot is in log-log scale and it includes the optimal square-root (SR) distribution, drawn with a thick line.

It is clear from Figure 4.1 that path replication does not achieve SR replication. Path replication is closer to SR distribution than owner replication, but still not close enough. The replication method that seems to approximate SR replication better is PtP.

In Figure 4.2, we plot (again in log-log scale) the normalized replication ratio of each data item compared to its query rate for networks with power-law overlay topology. We notice again that PtP approximates SR quite accurately and obviously better than owner and path replication. Intuitively, PtP replication strategy results in SR replication because it creates for each data item a number of replicas that is approximately the same as the number of visited peers during the search process, regardless of the number of visited peers or the topology of the overlay network.

In Figure 4.3, we plot (in log-log scale) for each data item the *sum of degrees* of all peers that hold a replica of the data item in comparison to the query rate of each data item for the networks with random and power-law, when PtP replication strategy is adapted (with flooding, 5 random walkers and 8 random walkers). As we observe, for random networks, with PtP replication strategy, the sum of degrees of peers that holds a replica of the data item is also proportional to the square root of the query rate. However, it is clear from Figure 4.4 that with networks with power-law topology, the sum of degrees of peers that hold a data item is not proportional to the square root of the query rate. This is more clearly shown in Figure 4.5 and Figure 4.6 where for each data item the actual number of replicas that exist in the network along with the sum of degrees of peers that hold a replica of it is presented for networks with random and power-law topology. Since the results obtained using 5 walkers, 8 walkers and flooding are very similar, we present the results only for the case of 8 random walkers. As we can notice, for random networks the number of replicas for each data item is proportional to the sum of degrees of peers with a replica of the data item while for power-law networks this claim is not true.

Figure 4.7 presents the *average depth_found* for each data item with its query rate in log-log scale, for both random and power-law networks when PtP replication strategy is used. Since all algorithms that are used with PtP result in the same replication ratio, we present the results for 8 random walkers only. Flooding and 5 random-walkers exhibit the same behavior so they are omitted from the plot for clarity. As expected, we observe that the *average depth_found* of each item has a linear relationship to its query rate. We also observe that for the network with random topology, the *average depth_found* for each data item is greater than for networks with power-law topology.

This comes both from the fact that in power-law networks the average distance between two peers is smaller in comparison to random networks. As a consequence, the location of data items is achieved within fewer hops (on average) on networks with power-law topology.

Figures 4.8 and 4.9 show the total number of messages that are exchanged (communication cost) both on random and power-law networks when PtP replication strategy is used. It should be clear from the figures that the communication cost is larger on the network with power-law topology in any case. This is due to the fact that in a power-law topology, random walks result in many cycles with peers receiving and forwarding *pull messages* from many of their neighbors more than once. Apart from this, since the average distance between any pair of peers is smaller, there are more successful queries and thus more *reply* and *push* messages.

In conclusion, PtP replication strategy results in SR replication when applied both on random and power-law networks. However, we can claim that this replication is optimal only for random networks.

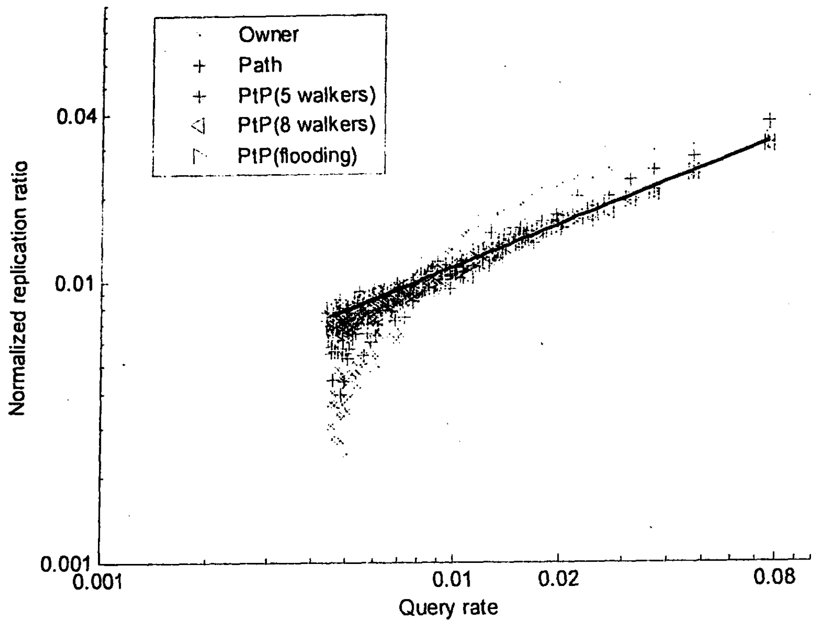

Figure 4.1: Distribution of replication ratios under various replication strategies on networks with random topology

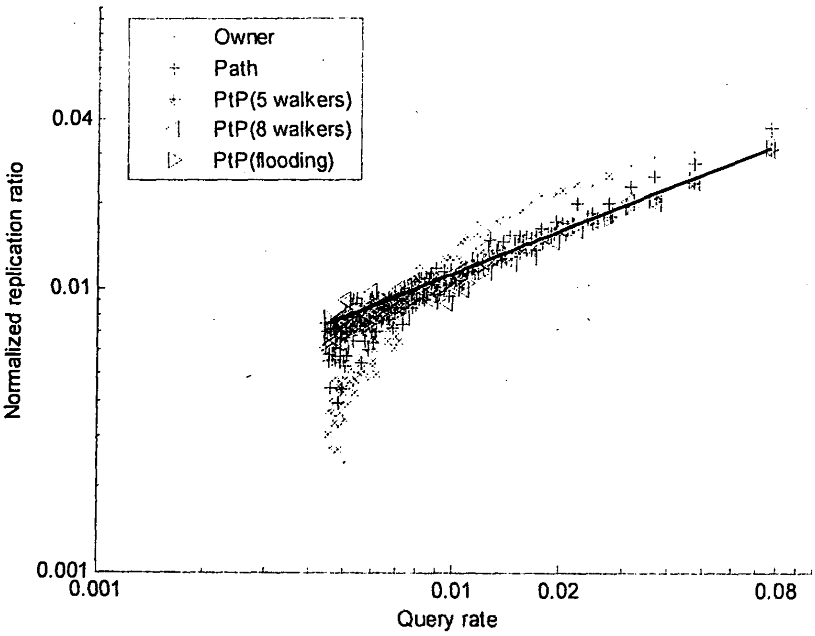

Figure 4.2: Distribution of replication ratios under various replication strategies on networks with power-law topology

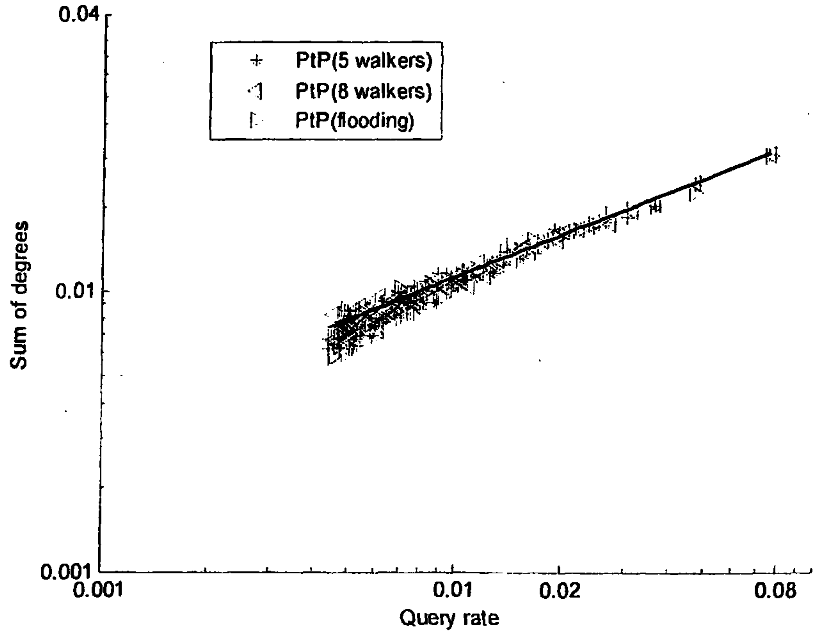

Figure 4.3: Sum of degrees of peer with a replica of the data item for each data item vs its query rate for networks with random topology

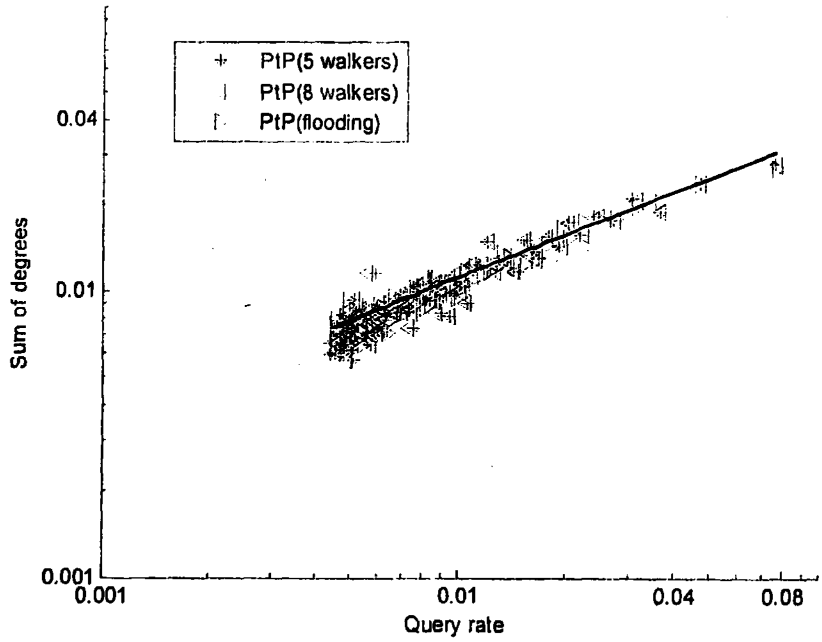

Figure 4.4: Sum of degrees of peer with a replica of the data item for each data item vs its query rate for networks with power-law topology

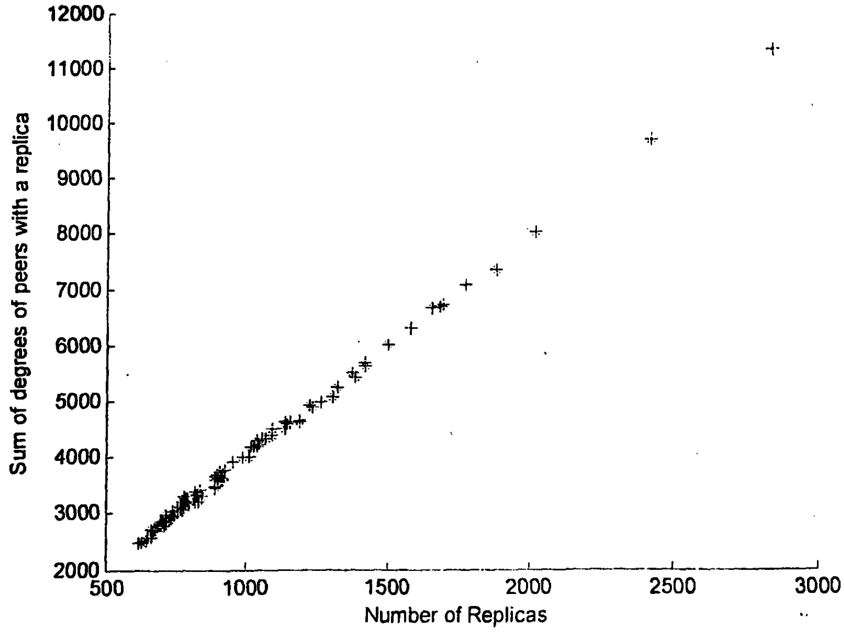

Figure 4.5: Number of replicas of each data item vs sum of degrees of peers with a replica of the data item for networks with random topology

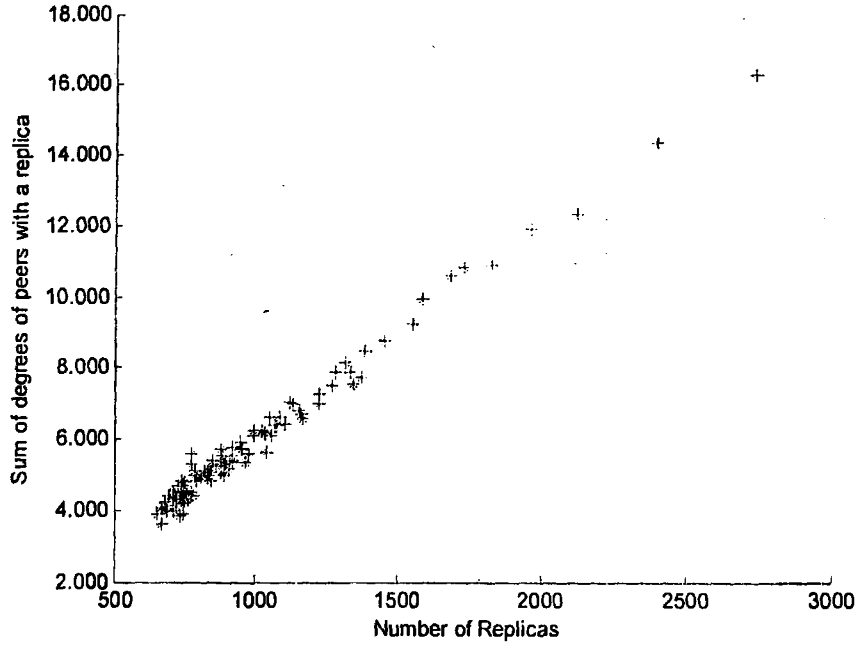

Figure 4.6: Number of replicas of each data item vs sum of degrees of peers with a replica of the data item for networks with power-law topology

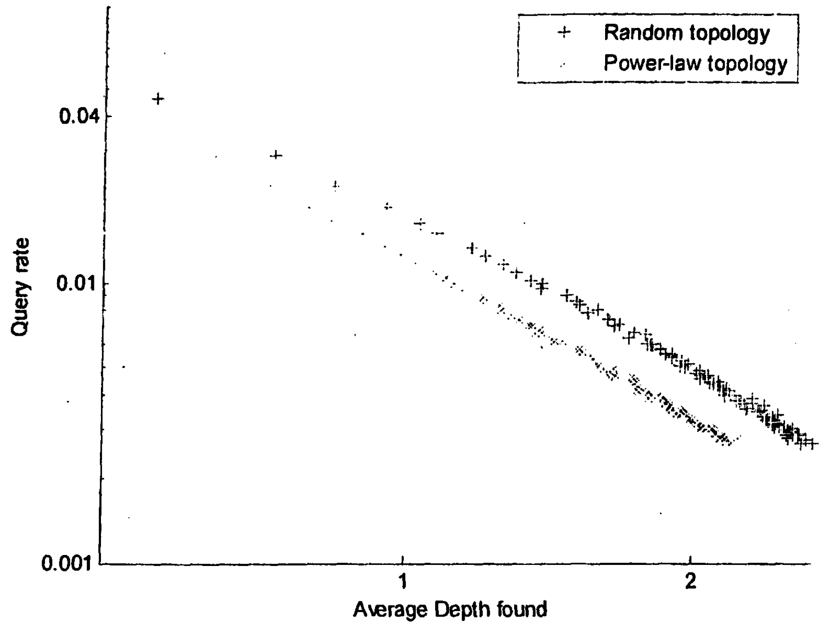

Figure 4.7: Average depth_found for each data item vs its query rate for networks with power-law and random topology

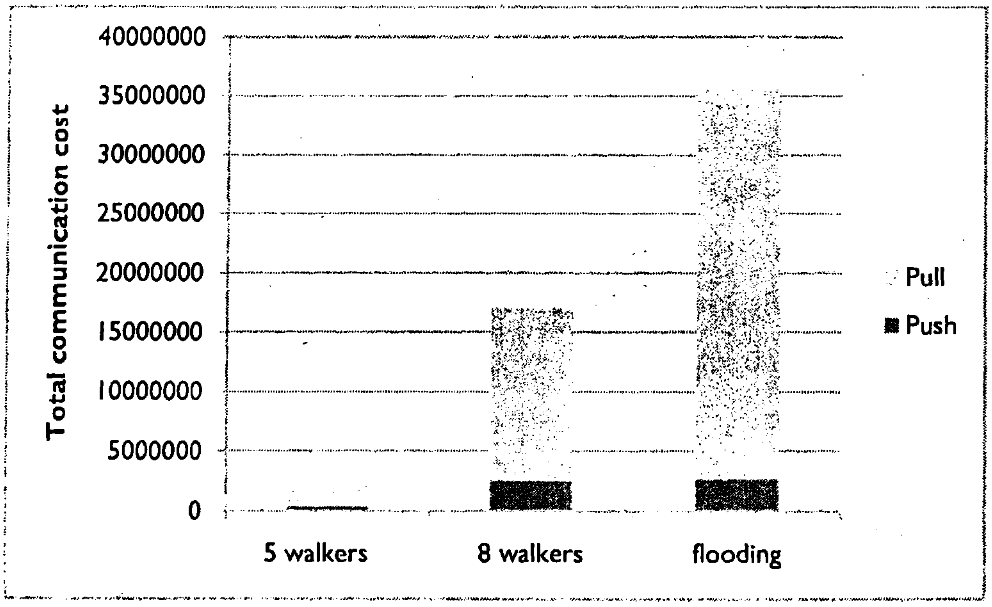

Figure 4.8: Total communication cost under various replication strategies on networks with random topology

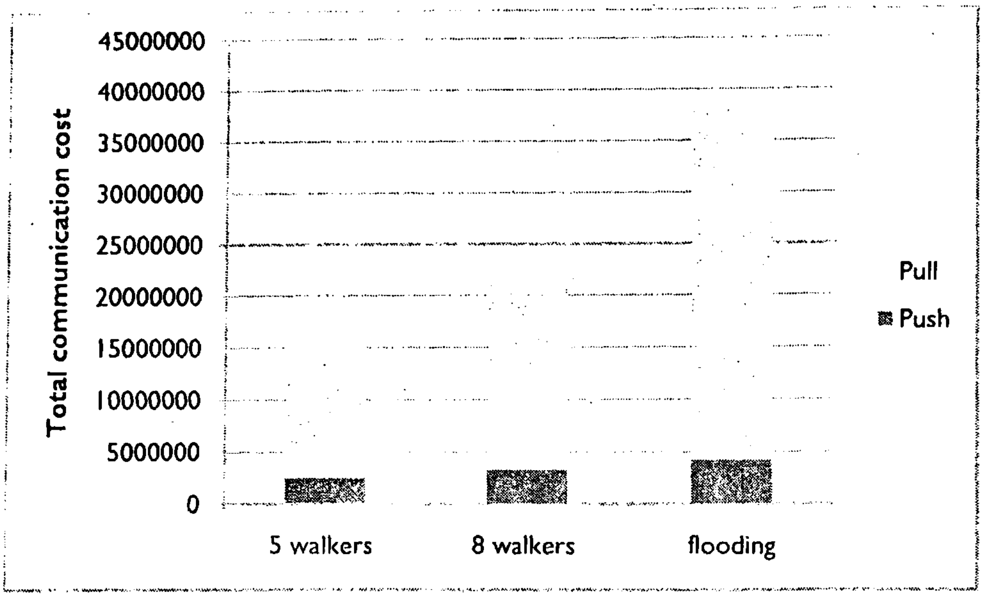

Figure 4.9: Total communication cost under various replication strategies on networks with power-law topology

4.2. A degree-based replication strategy for unstructured p2p systems with a power-law overlay network topology

As it is obvious from our simulation study, PtP replication strategy achieves optimal replication ratio when used on a network with power-law topology. However, the communication cost increases due to the cycles in random walkers. Moreover, a great proportion of the messages are duplicates. When a peer of low degree forwards a push message, there is a great probability that it will choose a peer with high degree (one of its few neighbors) to forward the message to. This means that if all peers of low degree forward the push messages they receive, peers with high degree will receive the same push message from many neighbors more than once and cycles in the random walkers occur more frequently. Thus, a way to reduce the communication cost is preventing peers with low degree from forwarding push messages and let only peers with high degree propagate the push message to other peers. Since a peer has no knowledge of the degree distribution or the average degree of all peers, the average degree of the peers in the path from the initiator to the provider peer is used as an indication of whether the peer is well-connected and has to forward the message or not.

Based on this observation, we present a variation of the PtP replication ratio which considers the properties of power-law topology for the creation of new replicas. With the *Degree-based Push-then-Pull* (DPtP) replication strategy, an extra field is added to the *pull* (and *reply, push*) *messages*: the *sum_degree* which counts the sum of degrees of all peers that are probed by a random walker. This field is initially set equal to the degree of the *initiator*. Each peer that receives a *pull message* apart from adding itself to the *path*, it also adds its degree to the value of degree-sum. When a peer receives a reply for a query it has posed, it enters a push phase and forwards push messages to *k* of its neighbors by including the *sum_degree* field. A peer that receives a push message stores the data item in its cache. Then it computes the average degree *avg_degree* of all peers in the path through which the data item was located as: $avg\_degree = \frac{sum\_degrees}{depth\_found}$. If the degree of the peer is greater than the *avg_degree*, the peer keeps forwarding the push message provided that the *hop count* does not exceed the TTL value which is set equal to *depth_found* minus one, just like in PtP replication strategy. If the degree of the peer is less than the *avg_degree,* the push message is not forwarded.

In order to evaluate the performance of the DPtP replication strategy, we have performed a series of experiments. We have applied PtP and DPtP algorithms on the same power-law network and experimented again with flooding with TTL = 10, 5 random walkers and 8 random walkers with the same TTL value. All the other simulation parameters are as described in Section 4.1 (Table 4.1). Figure 4.10, Figure 4.11 and Figure 4.12 present the normalized replication ratio and the sum of degrees of peers that hold a replica of the data item compared to the query rate and the query rate compared to the average *depth_found* for each data item respectively, in log-log scale. The figures show that PtP and DPtP result in very similar replication ratios and approximately the same average *depth_found* for all data items. The sum of degrees of peers that hold a replica for each data item is also remains practically the same. This is expected, since DPtP replication strategy does not practically reduce the number of replicas that are created after a successful search for a data item. What is reduced is the number of times that a peer may receive the same push message.

In Figure 4.13 we plot the total communication cost induced on power-law network by PtP and DPtP replication strategies. As it was expected, in any case, the total number of messages is less when DPtP replication strategy in comparison to the PtP (Figure 4.9). This is due to the fact that with DPtP not all peers that receive a push message forward the message.

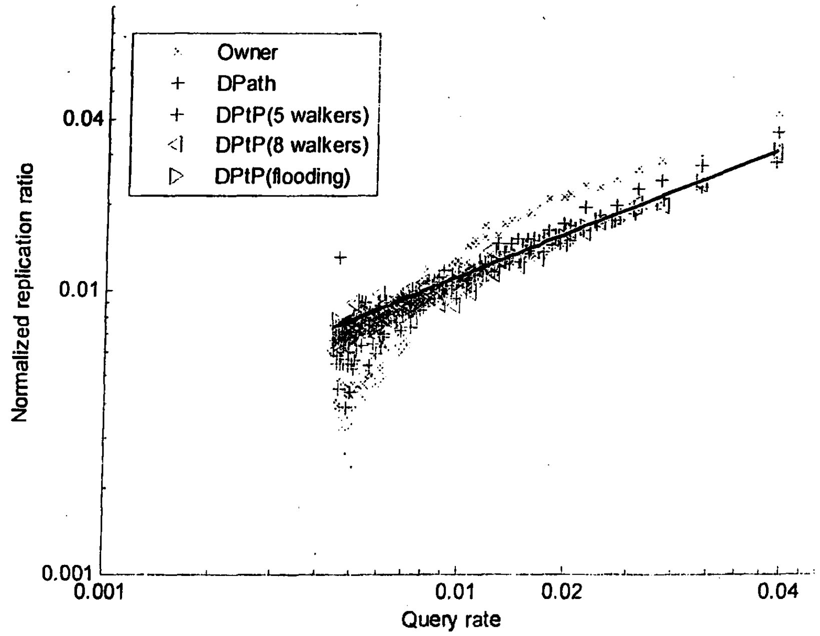

Figure 4.10: Distribution of replication ratios under various replication strategies using DPtP on networks with power-law topology

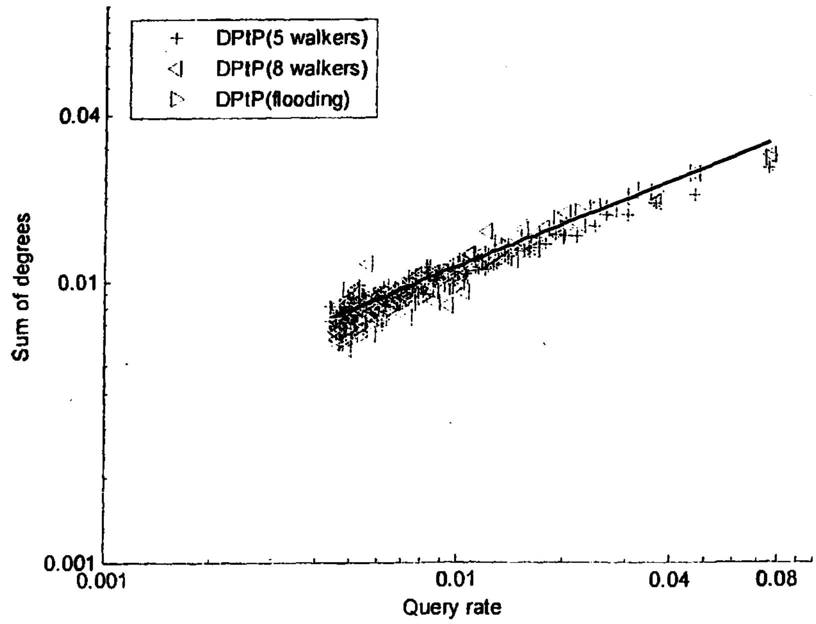

Figure 4.11: Sum of degrees of peer with a replica of the data item for each data item vs its query rate for networks with power-law topology

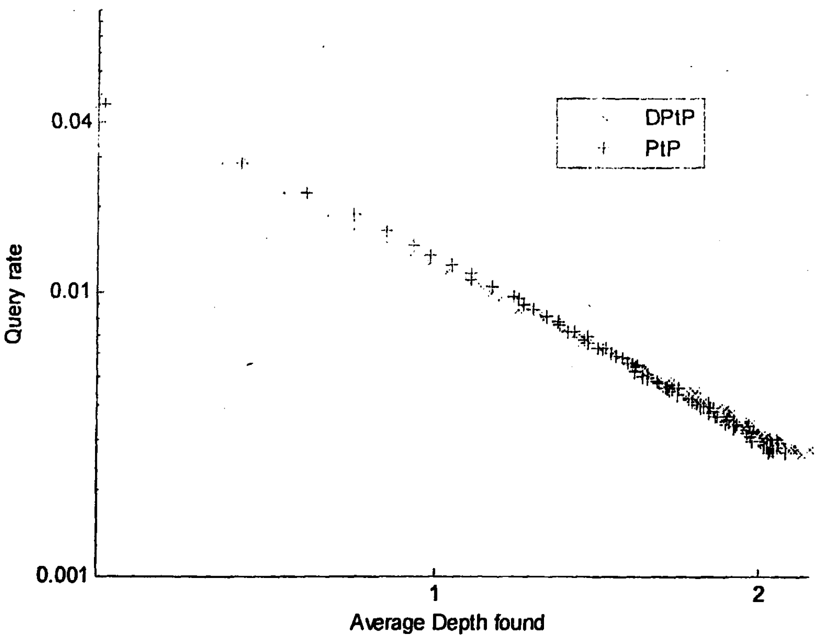

Figure 4.12: Average depth_found for each data item vs its query rate for networks with power-law topology under PtP and DPtP replication strategies

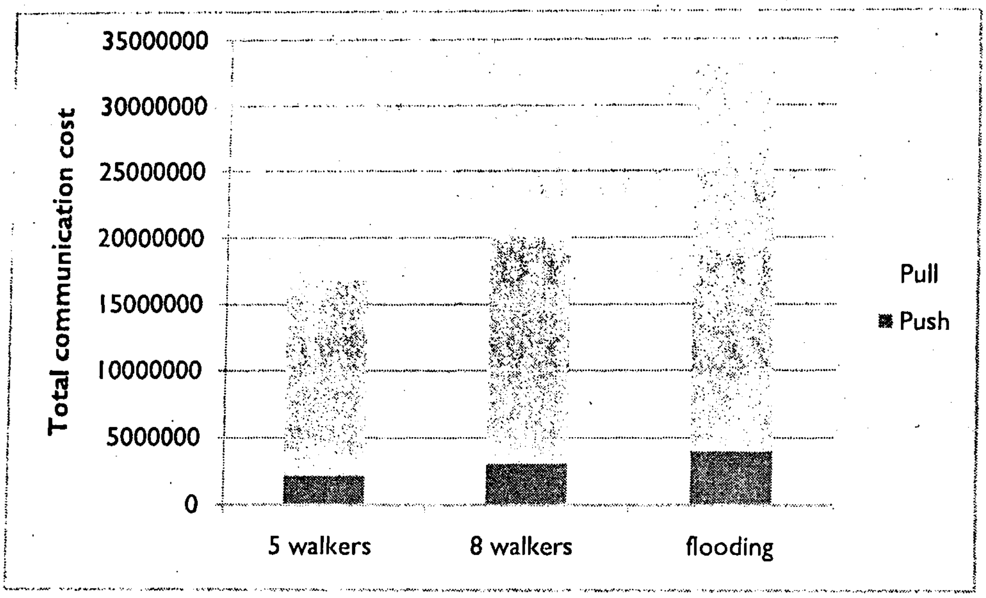

Figure 4.13: Total communication cost under various replication strategies on networks with power-law topology using DPtP replication strategy

CHAPTER 5. UPDATES ON UNSTRUCTURED P2P SYSTEMS WITH A POWER-LAW OVERLAY NETWORK TOPOLOGY

5.1. Maintaining the consistency of replicas on unstructured P2P systems

Most of the traditional p2p systems that have been widely used over the past years, such as Gnutella and Kazaa, enable users to retrieve and share data. The main characteristic of the data items that are shared through those systems is that they are considered to be static (users only wish to read shared data but do not modify them). However, in future p2p applications, a new challenge will emerge: the need for sharing data that may be frequently modified by the users. In p2p networks that will support such dynamic content, data items could both be read and written. Maintaining multiple replicas of each data item in the network introduces the additional overhead of keeping all replicas up-to-dated. If not every peer that holds a replica of a data item sees the same updates applied to the data item in equivalent orders, then not every peer could respond to a query request (at least not with an up to date and not stale content). Maintaining the consistency of all data items that are stored in the network requires that every update that is made in a data item by some peer is propagated to all other peers that hold a copy (replica) of the same data item. Therefore, consistency maintenance polices should be used along with replication strategies.

Each consistency maintenance policy deals with three crucial issues: *where* (i.e at which peers) updates take place, *when* updates are propagated to other replicas and *how* the propagation is achieved.

In terms of the *where* aspect, update policies can be classified as single or master copy and multi-master or group. In policies that are based on the *single master* or *primary copy* approach, it is assumed that each data item is owned by a single peer, known as the data item's *owner*. The replica that is held by the owner of the data item is called the *primary copy*. Every replica is allowed to be read but when an update takes place, the update must be first applied to the primary copy of the data item and then propagated. All the other replicas of the data item must be made consistent with the master copy. In the *multi-master* or *group* approach, multiple peers can hold primary copies of the same data item. Such an approach is more demanding in terms of communication cost and system complexity since concurrent updates on different replicas need to be coordinated and any replica divergences that occur should be reconciled.

As far as the question of *when* the update process takes place is concerned, there are several approaches. One approach requires that the update propagation process is initiated *periodically*. Alternatively, each peer that modifies a data item propagates the update to other peers right after the update is completed (*eager* or *synchronous replication*) or when an inconsistency is detected (*lazy or asynchronous replication*).

According to the *how* aspect, there exist two different types of consistency maintenance policies: policies that use push methods, policies that use pull methods and policies that use a combination of push and pull methods. In *push* policies, the peer that changes the content of a data item is responsible for propagating (pushing) the update to other peers. On the other hand, in *pull* policies, the peer that holds a replica of a data item takes the initiative of contacting other peers so as to keep its replica up to date. Push mechanisms are easily implemented but induce greater message overhead. Furthermore, there is a possibility that an update could not reach some peer, considering that peers join and leave the network and are not constantly connected. On the other hand, with pull methods an appropriate pull period should be determined. If a peer pulls too often the communication overhead is increased but pulling too rare decreases the consistency levels. Usually, a combination of *push/pull* techniques is used: a peer performs a limited push of the updates but the peers do not rely only on the other peer's push to get the update. Instead, they occasionally contact other peers (pull) to make sure they have not missed an update of the data item. It is proved in [16] that when a push/pull mechanism is used, better consistency levels and less message overhead is achieve than plain push or pull.

An adaptive pulling policy is presented in [15] to determine how frequently the peer should pull. It makes sense that for data items that are more frequently updated the pulling frequency is greater than for less frequently updated data items. To adjust the pulling frequency for each data item to the update rate, a time-to-refresh (*TTR*) value is associated with each data item in cache. This value indicates when the next pull for this item should happen. The *TTR* value varies according to previous pull results. Specifically, the algorithm starts with a minimum *TTR* value and adapts it to the update rates: if the peer finds out that a data item has not been modified between two successive pulls, the estimate TTR_e for the next pull is increased by an additive amount C according to the equation:

$$TTR_e = TTR + C \qquad (5.1)$$

If a new version of the data item is pulled, the estimate TTR_e of the next TTR is reduced by a multiplicative factor b, in portion of the difference D between the version that the peer had before pulling and the pulled version of the data item, according to the equation:

$$TTR_e = \frac{TTR_e}{D+b} \qquad (5.2)$$

The next TTR is a weighted average of the TTR_e estimate and the current TTR:

$$TTR_e = w * TTR_e + (1-w) * TTR \qquad (5.3)$$

where w determines the *rate of changes*: smaller values of w make TTR change very slowly, while larger *w* make *TTR* change quickly.

As mentioned before, a peer can either use a push method to communicate an update of a replica to other replicas of the same data item or pull other peers in order to retrieve any updates that have been made by other peers. Alternatively, a peer could combine push and pull methods in order to achieve better consistency levels. In particular, in a *push/pull* update propagation policy, a peer pushes any updates it makes on a data item by applying a mechanism similar to the search mechanism used for locating data items (for example k-random walks with a fixed TTL value). Apart from pushing, a peer periodically performs a pull for every replica it holds. During the pull process, a peer contacts a set of other peers and checks whether they hold a more up-to-date version of the same data item.

An alternative push/pull hybrid update propagation policy is discussed in [16]. It is assumed that the creation of replicas in the P2P network was determined by the *Pull-then-Push* algorithm described in Section 3.2 where a peer that requests a data item, after a successful search (*pull phase*) enters a *push phase* where it visits peers using the same algorithm as in the pull phase and forces them to hold a replica of the data item. Given this replica creation approach, each peer that holds a data item is characterized as *owner* if it holds a primary copy of the data item, *responsible* if it has requested the data item before and has forced the creation of replicas or *indifferent* if it has been forced to hold replica without requesting the data item. According to the *PtPU* policy, the owner broadcasts the new versions of a data item as soon as an update occurs. If a peer that is characterized as responsible for a data item receives a push message with a new version of the data item, it undertakes the task of informing its neighbors of the new version of the item. This is done by propagating the broadcast message exactly as in the push phase (*U-push phase*) when it has created the replicas (using the same algorithm as in PtP with the same parameters). By using the same algorithm in both the push phase and the U-push phase, it is guaranteed that a responsible peer will visit the same or approximately the same number of peers when pushing updates it becomes aware of as when creating replicas. Apart from pushing the updates they receive from the owner, peers that are considered responsible for a data item, pull periodically for that data item in order to obtain more updates.

5.2. Influence of Power-law overlay topology to update policies

5.2.1. Simulation Environment

We have evaluated the performance of both the Push-Pull and PtPU update policies described on Section 5.1 through simulations. We also investigated whether and how the performance of Push/Pull and PtPU is affected if the topology of the overlay network is not a uniform random graph. For this reason we have evaluated the performance of both plain Push/Pull and PtPU policies on networks with random and power-law topologies.

For our simulation study, we have used the same environment settings as described in Section 4.1.1. The networks with random (Erdős-Rényi) and power-law topology were generated exactly as described in Section 4.1.1.

Query model: The simulation time at each experiment is divided into two parts: the *replicas creation* part and the *replicas update part.* During the first part peers randomly issue requests for data items and after a successful query the peer that received a reply to its request enters a push phase where new replicas of the data item are created. The propagation of *queries*, *replies* and *push* messages is done according to the mechanism described in Section 4.1.1. The search method that is used for locating data items is either k random walks or flooding and the replication strategy that is employed is PtP.

During the second part, peers start updating data items. At the beginning of each simulation each of the s data items is randomly assigned to one of the n peers, which is considered as its *owner*. For simplicity we assume that each peer updates a data item only if it is considered its owner. The simulation time is divided into *time rounds*. At the beginning of each *simulation time round*, one of the peers is triggered through a self message and performs the *generate_update* process: if the peer is regarded as *owner* for at least one of the s data items, it randomly chooses one of those and updates it by incrementing the *version number* associated with that data item and assigning a new random value. After updating a data item, the peer enters a

U-push phase where it starts spreading the update by sending a *U-push message* to k of its neighbors with the new value and the new *version number* of the data item. For propagating the *u-push messages* each peer uses the strategy as when creating the replicas. When a peer receives a *U-push message* it checks if it holds a replica of the data item. If it hold a replica of the data item and has an older version number for that data item, it updates the data item in its cache and then forwards the *U-push message*. Otherwise, it just forwards the *U-push message*, as long as the TTL value is not exceeded. In case the *PtPU* update policy is used, if a peer is characterized as *responsible* for a data item and receives a *U-push message* with a new version for that data item, it enters a *U-push phase* itself and propagates *U-push messages* using the same strategy and the same TTL value as with *push messages*.

Apart from *U-push messages*, peers also exchange *U-pull messages* in order to get informed of new versions of data items they hold. For each peer we follow the adaptive pulling policy discussed in Section 5.1 in order to determine the moment of the next pull for a particular data item. With *Push/Pull* update policy, each peer pulls for every replica they hold using the same strategy as update push while with *PtPU* each peer initiates a pull only for data items that is held *responsible* for and if a peer that is responsible for a data item receives a reply with a new version of the data item, it will begin a *U-push phase* using the same strategy as in the replica creation (PtP) part.

Summary: At each experiment we need to define the topology of the network (random or power-law), the update policy that is used (*Push/Pull* or *PtPU*). The parameters concerning the number of peers in the network, the number of shared data items or the search and replication methods that are used are those defined in Table 4.1. Table 5.1 summarizes the parameters that are used for the update policies.

Table 5.1: Summary of simulation parameters for update policies

Parameter	**Symbol**	**Default value**
Generate update rate	u	0.2 (seconds)
Simulation time for replica creation part	T	360(seconds)
Simulation time for replica update part	t	50 (seconds)
Initial TTR for each peer	TTR	0.5 (seconds)
C parameter for adaptive pulling policy	C	0.4
b parameter for adaptive pulling policy	b	0.2
Rate of changes for adaptive pulling policy	w	0.8

Output parameters: The metrics that are measured at each simulation are the following:

a. The *consistency percentage* at the beginning of each simulation time round, which is computed as the percentage of replicas in the whole network that are consistent (have the same version number as the primary copy's version number)

b. The number of *u-pull* and *u-push* messages that are exchanged during the *replica update simulation part*. The sum of those push and pull messages constitutes the *message overhead* of each method.

5.2.2. Simulation Results

In this set of experiments, the Push/Pull and PtPU update policies were applied to networks with random and power-law topologies. The scope of the experiments is:

- to investigate whether PtPU achieves better consistency levels in comparison to plain push/pull and with lower message overhead.
- to investigate the effect that the topology of the overlay network has on the update policies

For this series of experiments, we have used a 5-random walks strategy, an 8-random walks strategy and flooding with TTL = 10 for the PtPU algorithm (both creating replicas of a data item (push phase) and for the propagating updates that a responsible peer becomes aware of (u-push phase)). Each simulation was executed 10 times and the results that are presented are the averages among the 10 executions.

Figure 5.1 illustrates the percentage of consistent replicas at the beginning of each simulation time round on networks with random topology when Push/Pull and PtPU update policies are used with responsible peers following a 5-random walks strategy with TTL = 10 for propagating any updates they receive, while in Figure 5.2 the consistency percentage under Push/Pull and PtPU on networks with power-law topology is plotted. As expected, PtPU keeps better consistency levels than push/pull both in the case of networks with random topology and networks with power-law topology because each peer is held responsible for updating the replicas it has created.

It is also clearly shown in Figure 5.1 and Figure 5.2 that both Push/Pull and PtPU update policies result in better consistency levels when applied to power-law networks than when applied to uniform random networks. This can be explained by the fact that in a network with power-law topology, a peer can easily communicate an update or become aware of an update of a replica it holds through the well-connected peers. When a peer pushes an update, there is a great probability that the push will reach one of the few well-connected peers which in turn will push the update to other peers. As a result more peers will become aware of an update so the consistency will increase.

Figure 5.3 and Figure 5.4 show the massage overhead associated with Push/Pull and PtPU update policies for random and power-law topologies respectively. As it can been observed from the figures, in both cases, Push/Pull results in greater message overhead in comparison to PtPU. With PtPU each peer does not pull for every data item that is stored in its cache, but only for data items it is held responsible for, thus requiring less pull messages. However, with PtPU more push messages are exchanged because if a peer is responsible for a data item, when it receives a new version for a data item it pushes the new version to other peers. The total message overhead is decreased though when PtPU update strategy is adapted. In power-law networks, the diameter of the network is smaller and consequently the average search size of a data item is decreased. As a result, the number of peers that manages to get a reply for their request is greater. This means that more peers become responsible for a data item. A peer pushes an update it receives only if it is held responsible for the data item so the more the responsible peers, the more the push messages. When 8 random walkers are used instead of 5, as one would expect, the consistency levels are higher both for random (Figure 5.5) and power-law (Figure 5.6) network topologies as more peers are informed of an update during a U-push. However, contacting more peers means more pull messages and consequently extra message overhead (Figure 5.7 and Figure 5.8). The same holds when we use flooding instead of 8 random walkers (Figures 5.9, 5.10, 5.11).

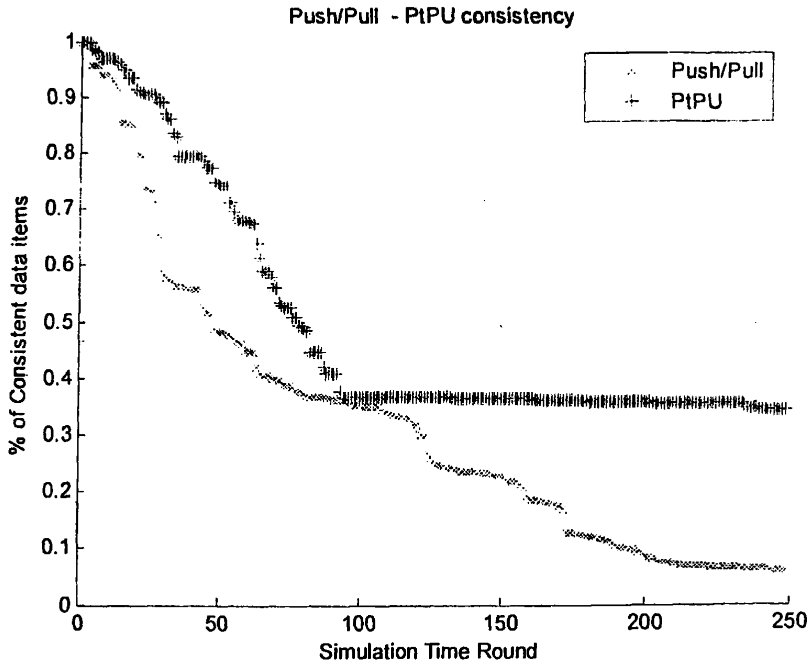

Figure 5.1: Consistency percentage at each time round under Push/Pull and PtPU strategies on networks with random topology when for the U-push phase a 5 random walker with TTL = 10 was used

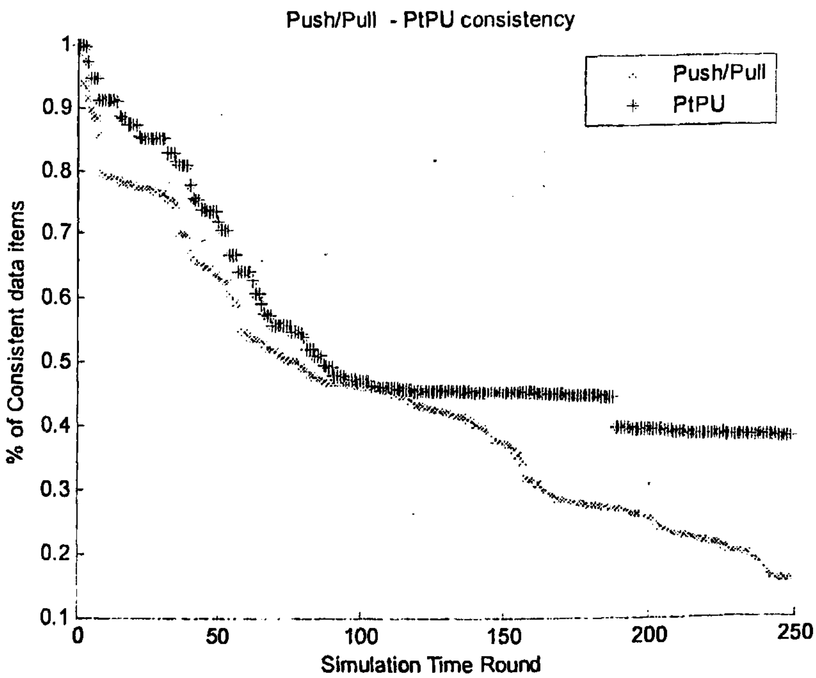

Figure 5.2: Consistency percentage at each time round under Push/Pull and PtPU strategies on networks with power-law topology when for the U-push phase a 5 random walker with TTL = 10 was used

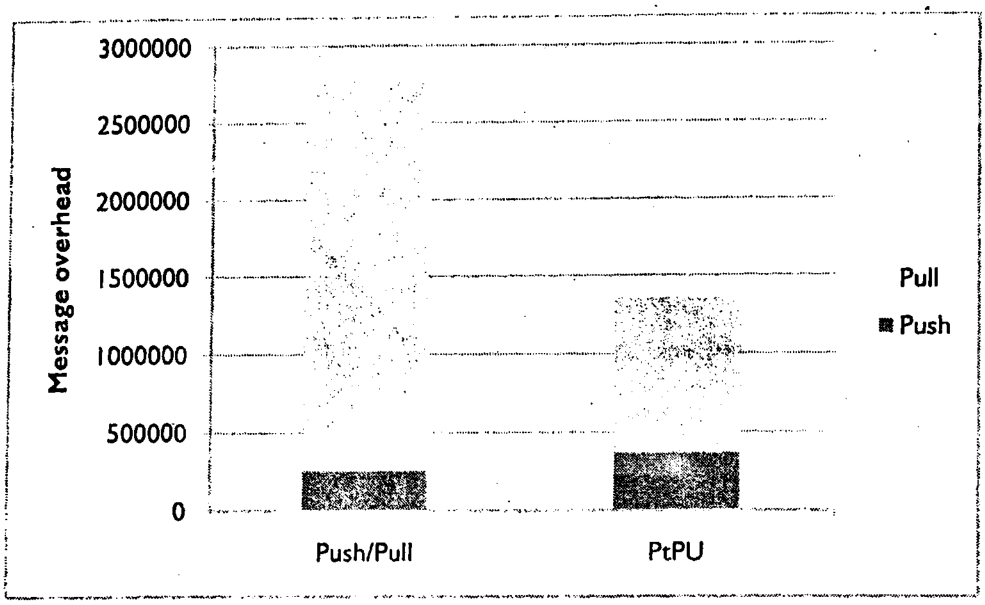

Figure 5.3: Total message overhead under Push/Pull and PtPU strategies on networks with random topology when for the U-push phase a 5 random walker with TTL = 10 was used

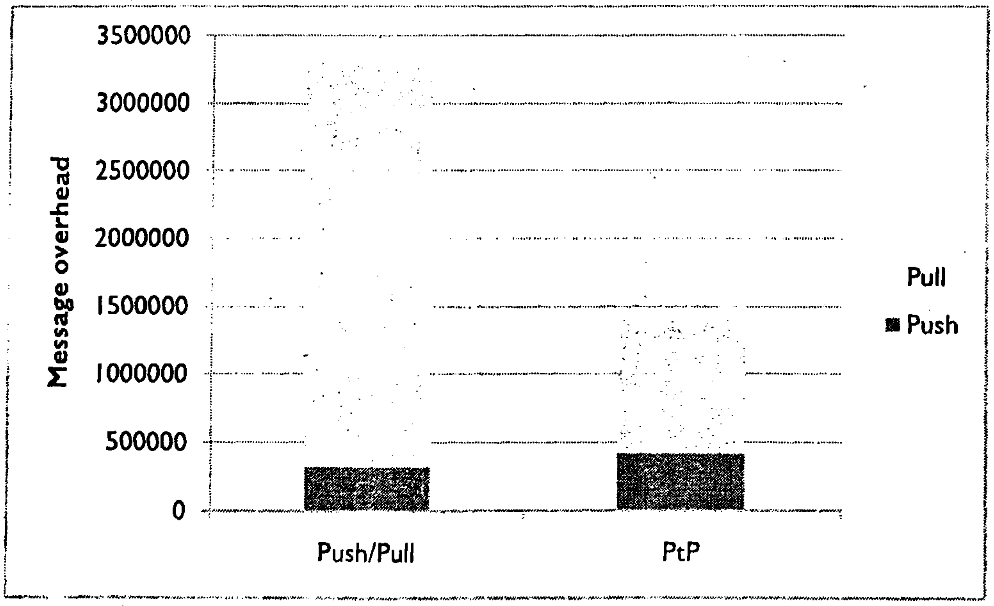

Figure 5.4: Total message overhead under Push/Pull and PtPU strategies on networks with power-law topology when for the U-push phase a 5 random walker with TTL = 10 was used

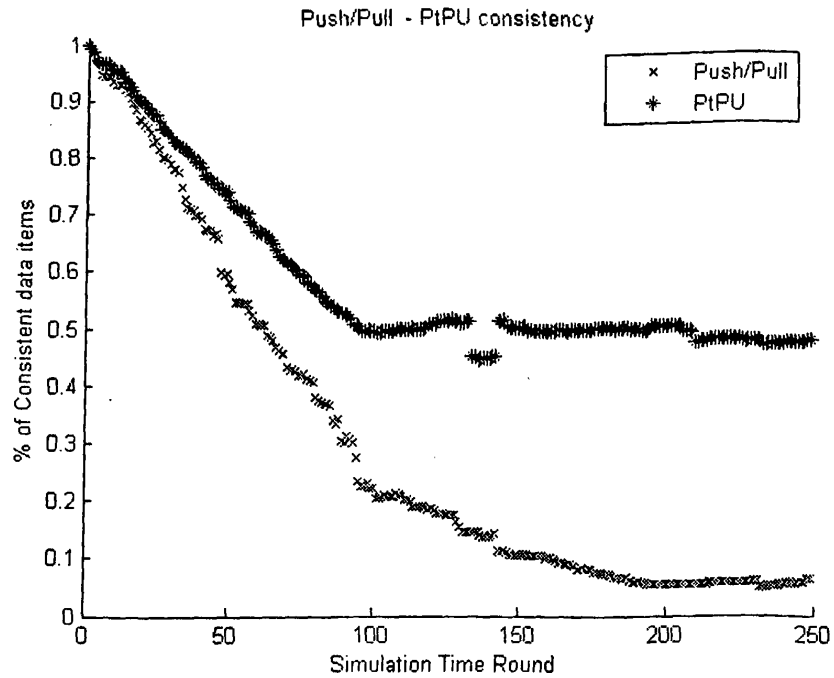

Figure 5.5: Consistency percentage at each time round under Push/Pull and PtPU strategies on networks with random topology when for the U-push phase a 8 random walker with TTL = 10 was used

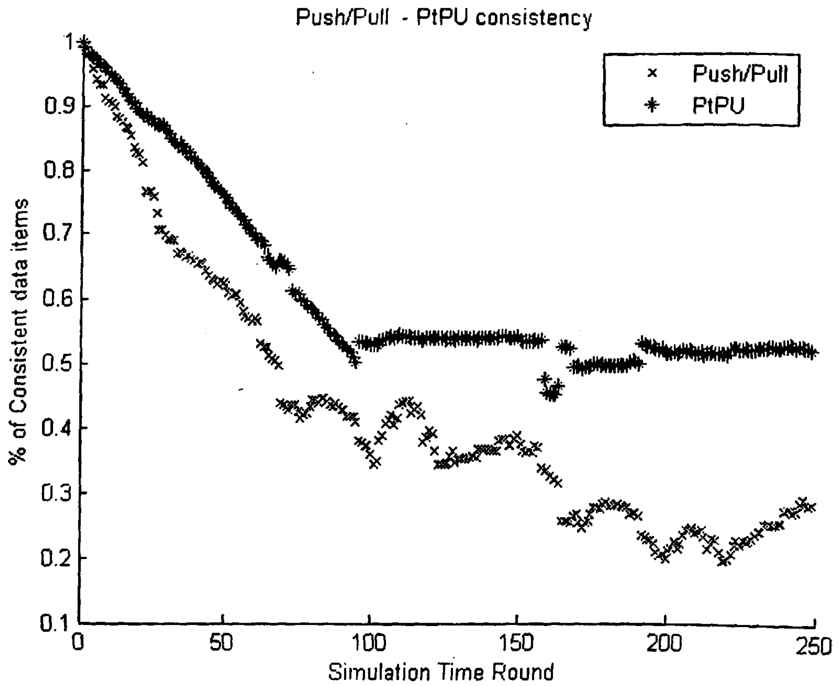

Figure 5.6: Consistency percentage at each time round under Push/Pull and PtPU strategies on networks with power-law topology when for the U-push phase a 8 random walker with TTL = 10 was used

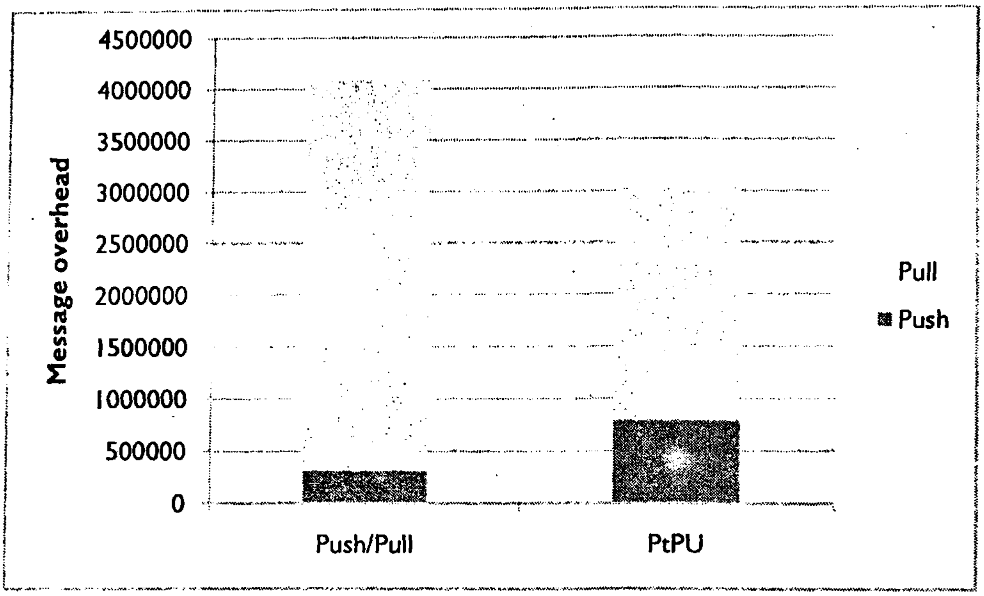

Figure 5.7: Total message overhead under Push/Pull and PtPU strategies on networks with random topology when for the U-push phase a 8 random walker with TTL = 10 was used

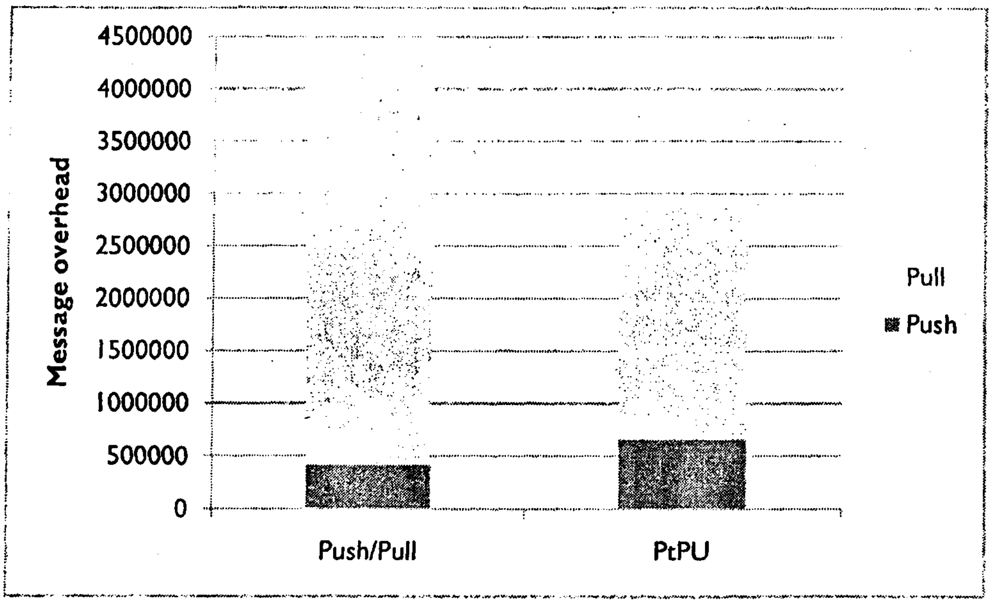

Figure 5.8: Total message overhead under Push/Pull and PtPU strategies on networks with power-law topology when for the U-push phase a 8 random walker was used

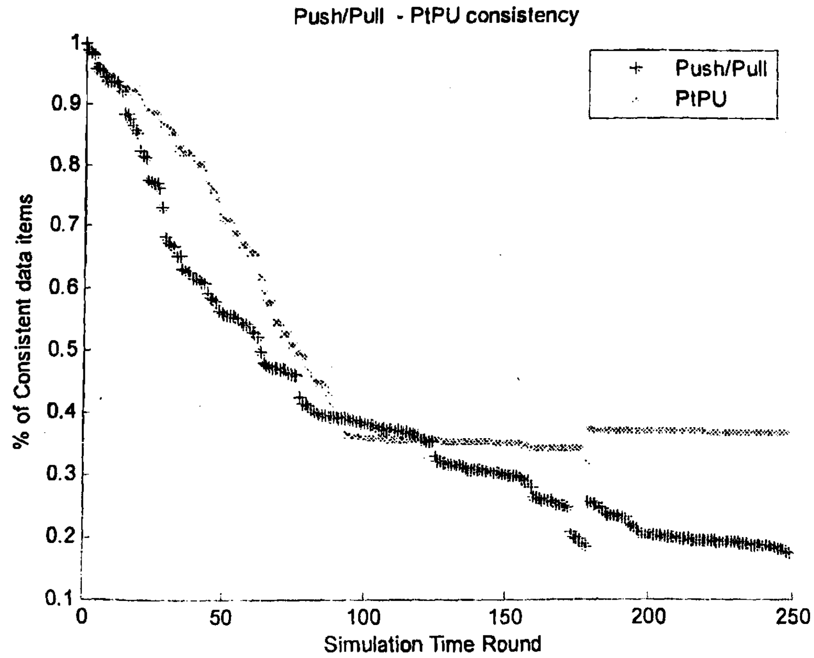

Figure 5.9: Consistency percentage at each time round under Push/Pull and PtPU strategies on networks with random topology when for the U-push phase flooding with TTL = 10 was used

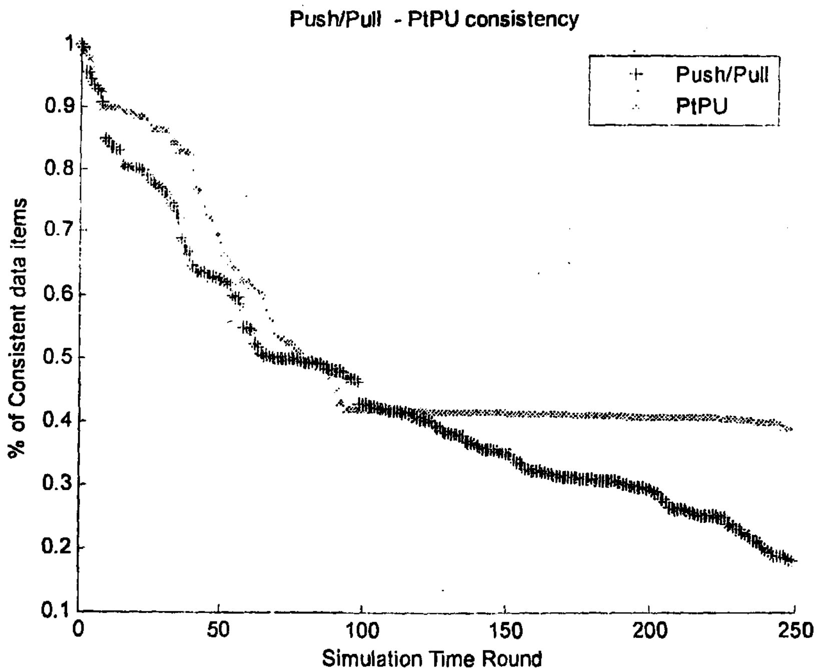

Figure 5.10: Consistency percentage at each time round under Push/Pull and PtPU strategies on networks with power-law topology when for the U-push phase flooding with TTL = 10 was used

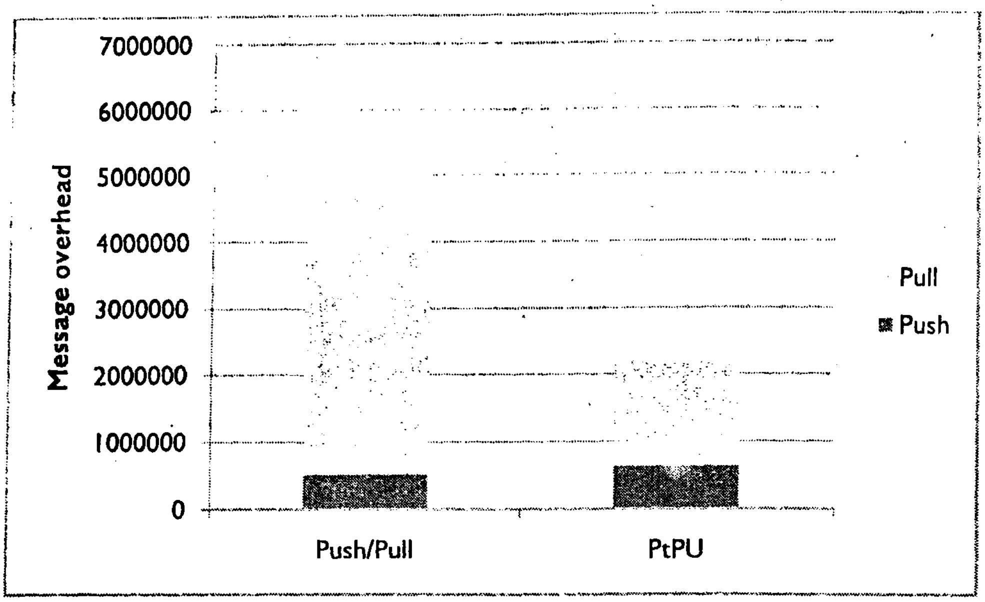

Figure 5.11: Total message overhead under Push/Pull and PtPU strategies on networks with random topology when for the U-push phase flooding was used

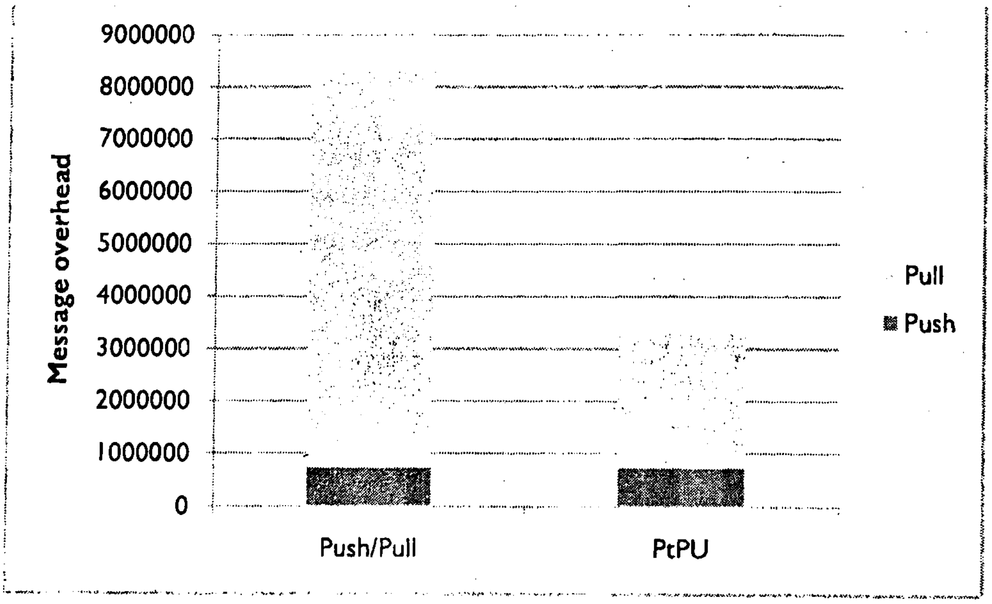

Figure 5.12: Total message overhead under Push/Pull and PtPU strategies on networks with power-law topology when for the U-push phase flooding was used

5.3. A degree-based update policy for P2P systems with a power-law network topology

As indicated by the experimental results presented on Section 5.2, power-law networks cause a larger message overhead but achieve better consistency levels than uniform random networks when PtPU is applied. Due to the cycles in random walks, the well-connected peers may be informed of an update made by some other peer more than once through multiple paths. The reduction of such duplicate messages would decrease the total message overhead. Furthermore, the message overhead produced by the well-connected peers would be decreased if those well-connected peers did not pull as often as other peers and rely more on other peer's pushes in order to obtain a new version. Since a push from any peer quickly reaches one well-connected peer (according to random probes strategy), a well-connected peer will be informed of any updates without having to pull frequently. Motivated by this observation, we present a variation of the PtPU update policy with an adaptive pulling policy, where the *TTR* value that indicates the time interval between two successive pulls of a peer for the same data item, depends on the degree of the peer.

According the *Degree-based Pull-then-Push Update* (*DPtPU*) strategy, replication is done using the DPtP instead of PtP strategy. During the push phase, as replies are propagated back to the requestor, the sum of degrees of all peers in the path from the provider to the requestor peer is computed as described on Section 4.2. When a peer receives a reply for a request it has posed, it stores the data item in its cache and considers itself as responsible for the received data item. Apart from storing the value and the version number of the received data item, the responsible peer also stores the sum of degrees (*sum_of_degrees*) of all peers in the path through which the reply came.

A peer that is considered as responsible for a data item that is stored in its cache, periodically pulls for that data item. For determining the time interval *TTR* between two successive pulls of a peer for a specific data item, the adaptive pulling policy discussed in Section 5.1 is used. The adaptive pulling policy is adjusted so that peers with high degree pull less frequently than peers with low degree. An estimation of weather a peer is well-connected or not (since it cannot have knowledge of the degree of all other peers) is computed as the portion of its degree to the sum of degrees of peers in the path through which the peer received a reply:

$$TR_e = \frac{out\_degree_i}{sum\_of\_degrees} \tag{5.4}$$

so the TTR estimation from equation 5.3 becames:

$$TTR_e = w * TTR_e + (1 - w) * TTR + h * TR_e \tag{5.5}$$

where h is given as an input parameter with a positive value.

To further reduce the number of unnecessary push massages, after a pull that returned a newer version of the data item, a responsible peer initiates a *U-push* phase only if the degree of the peer is larger than the average degree of peers in the path through which the peer received the new version. Otherwise, it is considered a low-degreed peer, therefore there is no meaning in starting pushing the new version, as most of its neighbors most likely will have already have received the new version.

In order to investigate the effect that DPtPU policy has on the performance of PtPU when applied to networks with power-law topology, we have performed a series of experiments. As in Section 5.2 we consider three cases for the PtPU strategy: 5 random walks, 8 random walks and flooding with TTL = 10. In Figures 5.13 and 5.14 we plot the consistency level and message overhead in case of 5 random walks while Figures 5.15 and 5.16 show the same plots in the case of 8 random walkers and finally Figures 5.17 and 5.18 show the results when PtPU uses flooding during the U-push phase of a responsible peer. As one can notice, in all three cases, the number of both push and pull messages are decreased causing the total message overhead to decrease compared to the PtPU. However, the consistency percentage seems to be influenced by the number of duplicate messages. For example, when 4 random walks is used, PtPU has slightly better consistency levels that DPtPU due to the fact that with DPtPU peers tend to pull less frequently than in PtPU. However, as the probability of having duplicate messages increases (8 walkers or flooding), DPtP tends to result in higher consistency levels. This happens because with PtPU, the TTR value of peers (especially well-connected peers) is decremented more frequently because more pull messages are received without locating a new version of the data item.

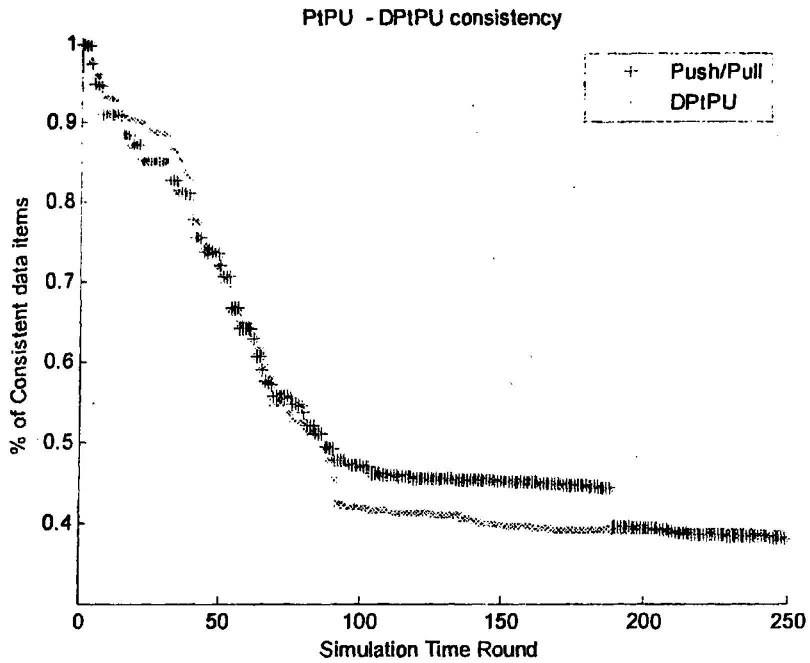

Figure 5.13: Consistency percentage at each time round under PtPU and DPtPU strategies on networks with power-law topology when for the U-push phase a 5 random walker with TTL = 10 was used

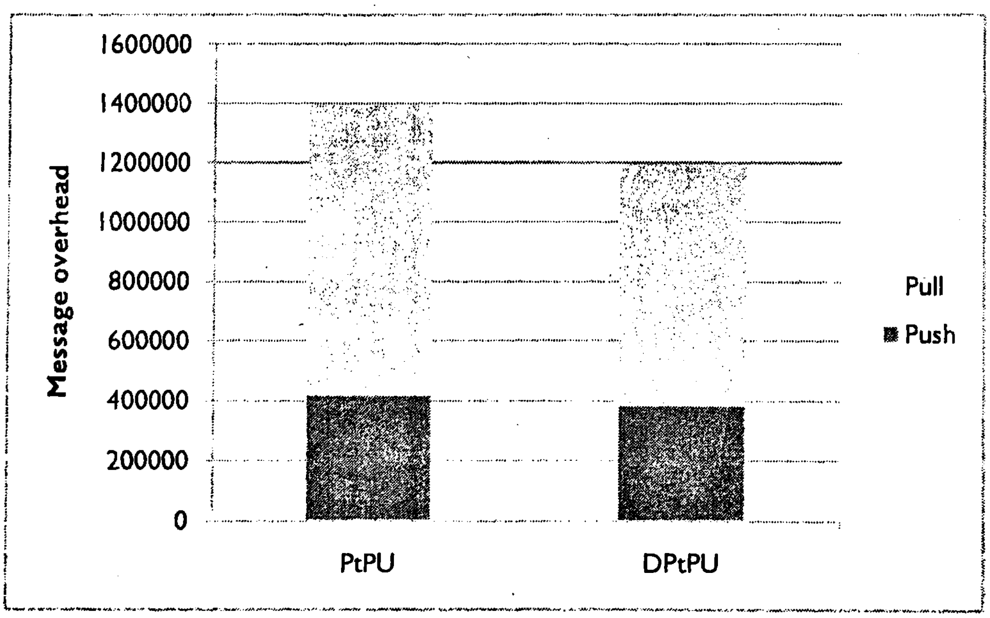

Figure 5.14: Total message overhead under PtPU and DPtPU strategies on networks with power-law topology when for the U-push phase a 5 random walker with TTL = 10 was used

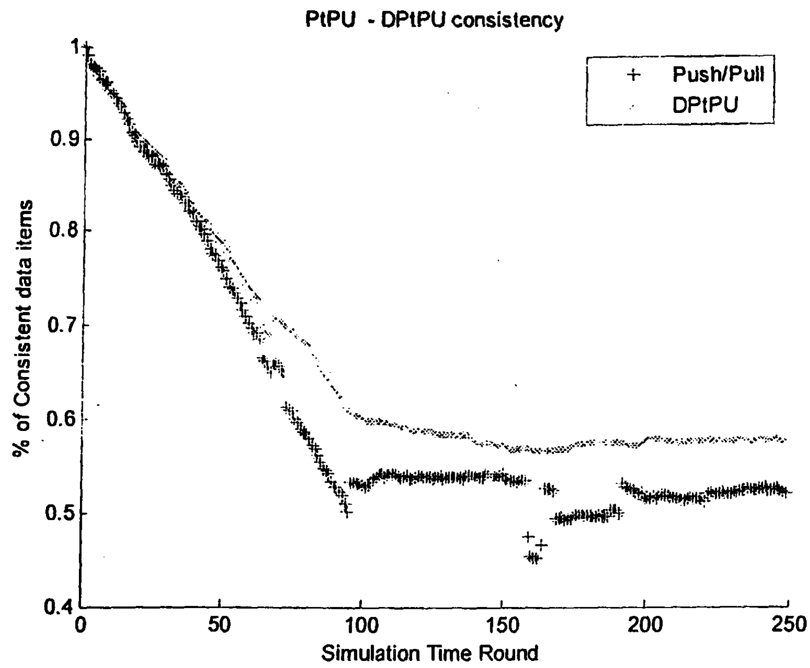

Figure 5.15: Consistency percentage at each time round under PtPU and DPtPU strategies on networks with power-law topology when for the U-push phase a 8 random strategy with TTL = 10 was used

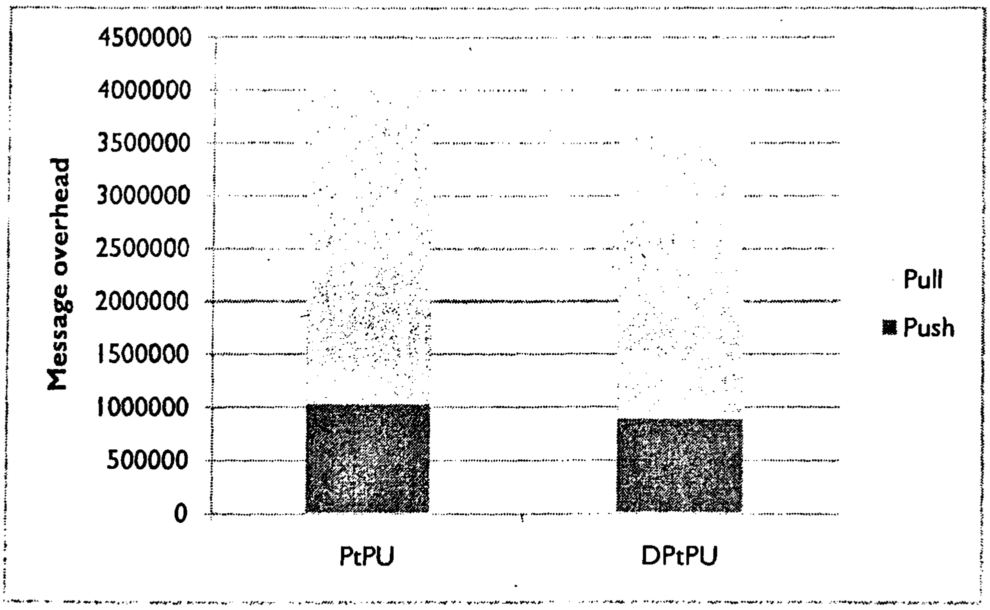

Figure 5.16: Total message overhead under PtPU and DPtPU strategies on networks with power-law topology when for the U-push phase a 8 random walker with TTL = 10 was used

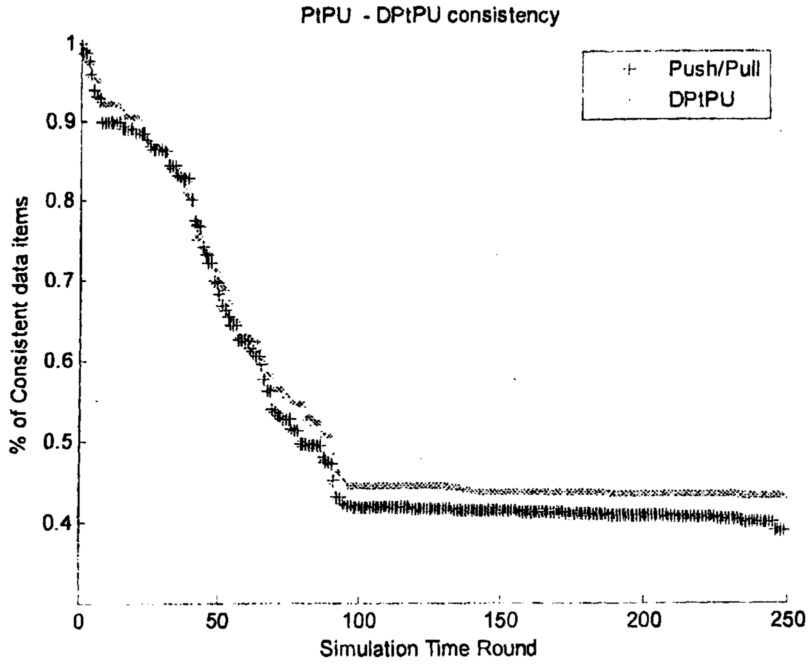

Figure 5.17: Consistency percentage at each time round under PtPU and DPtPU strategies on networks with power-law topology when for the U-push phase flooding with TTL = 10 was used

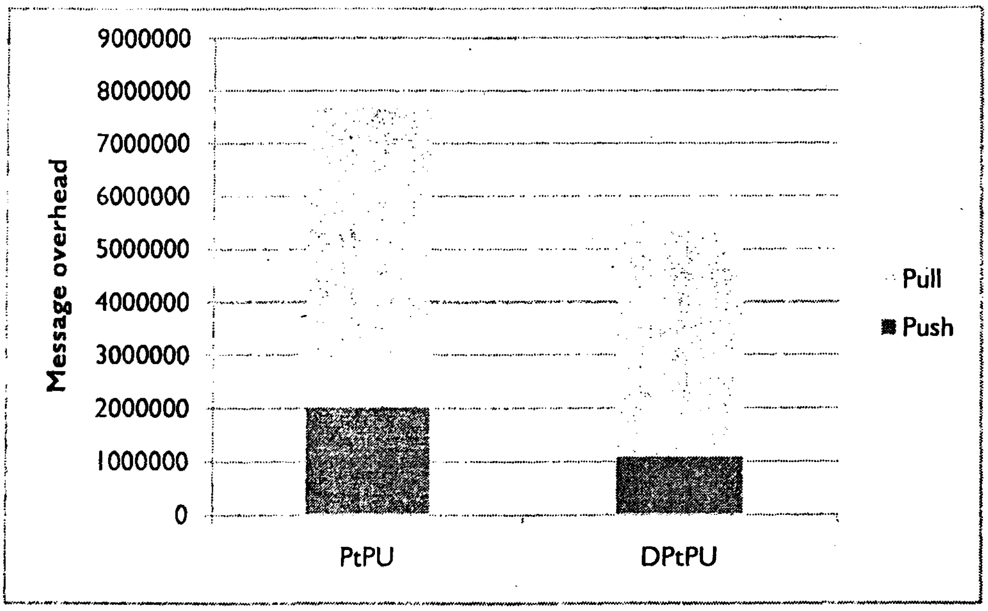

Figure 5.18: Total message overhead under PtPU and DPtPU strategies on networks with power-law topology when for the U-push phase flooding walker with TTL = 10 was used

5.4. An adaptive quorum-based update policy

Traditional quorum consensus [1, 4, 27] has been widely used as a technique for maintaining consistency among multiple replicas of shared data items. The basic idea behind quorum-based techniques for consistency maintenance is that for a particular data item we consider apart from a version number, a number of *votes*: when a peer wants to read or write a data item, a minimum number of votes (*quorum*) must first be obtained. When the peer has collected a specified minimum number of votes, known as *vote threshold* or *quorum level*, it can read the data item by choosing the most up-to-dated version number among the received votes or write a data item by pushing the new version number to all the received votes. Usually, two different quorum levels are defined: one for reading (*read quorum level*) and one for writing (*write quorum level*). These thresholds indicate the minimum number of votes that need to be collected for reading and writing.

In order to ensure consistency maintenance, there are two restrictions regarding the values of read and write quorum levels. The first restriction is that the sum of read quorum level and write quorum level must be greater than the total number of votes that are available in the system for the particular data item. The second restriction is that the write quorum level must be more than half of the total number of votes assigned for the data item. Those two restrictions ensure that the set of votes that will be collected for a reading and the set of votes that will be written after an update will overlap. This way, all reads in the system will be consistent as in the set of votes that will receive, there will certainly be one vote that has the most up-to-dated version of the data item.

The choice of appropriate *read* and *write quorum levels* may affect the total consistency of the system and depends on the needs of the particular system. For example, in a system with frequent reads and infrequent writes it makes sense to set a small read quorum level and larger write quorum level but in a system with frequent updates and rare writes a small write and a larger read quorum levels are preferred.

In a completely distributed and dynamic environment, where replication is used, assigning each data item a number of votes is not easy as there is no global information concerning the total number of replicas for the data item. Practically, the PtPU method can be considered as a *read-one-write-all* quorum based policy where at every read only one data item is accessed (*read quorum level = 1*) and any updates are propagated using a TTL value equal to the TTL value used for locating the data item. We present an adaptation of the PtPU update policy that is based on the idea of quorum consensus. In the adaptive quorum based technique, when a peer needs to read a data item, it is forced to read 2 replicas of the data item (*read quorum level = 2*) and when an update occurs in the owner of a data item, the update is forwarded (*u-push phase*) with TTL value set to half as much as the TTL value used for locating the data item. This way, we make the update process less expensive while making reads more expensive. In a system where the updates are more frequent than reads, the total communication cost would be decreased if the PtPU with read quorum level set to 2 is used.

We have evaluated the performance of the adaptive quorum-based PtPU policy through simulation results on a network with power-law topology. Particularly, we have altered the PtPU policy so that apart from creation of replicas and update of data items, a peer may also occasionally perform a read where it issues a request for a data item but if it receives a reply it does not force the creation of new replicas for the data item. In the case of PtPU with *read quorum level = 1,* at each read only one replica needs to be located and the read is considered to be *consistent* if the version number that the peer received as a reply is equal to the version number of the primary copy of the data item. When a peer creates a new replica of a data item it pushes the update for TTL hops, equal to the TTL used for locating the data item. In the case of PtPU with *read quorum level =2,* at each read two replicas need to be located and the read is considered to be *consistent* if the greatest of the version numbers that the peer received as a reply is equal to the version number of the primary copy of the data item. When a peer creates a new replica of a data item it pushes the update for half as much hops in PtPU with *read quorum level = 1* (TTL/2). During the simulation we count the percentage of reads that are consistent and the total messages exchanged for pushing updates (*push messages*) and reading data items (*read messages*).

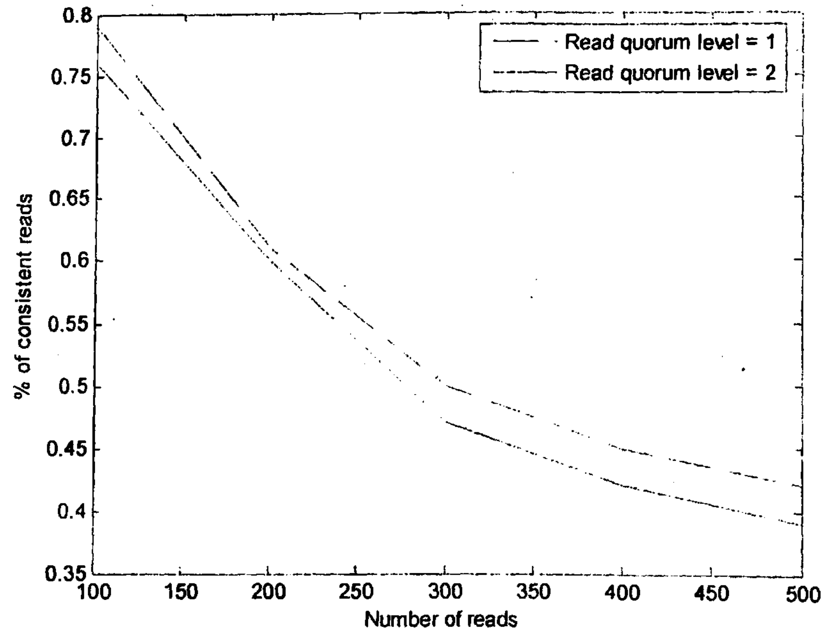

Figure 5.19: Percentage of consistent reads for PtPU with read quorum level 1 and 2

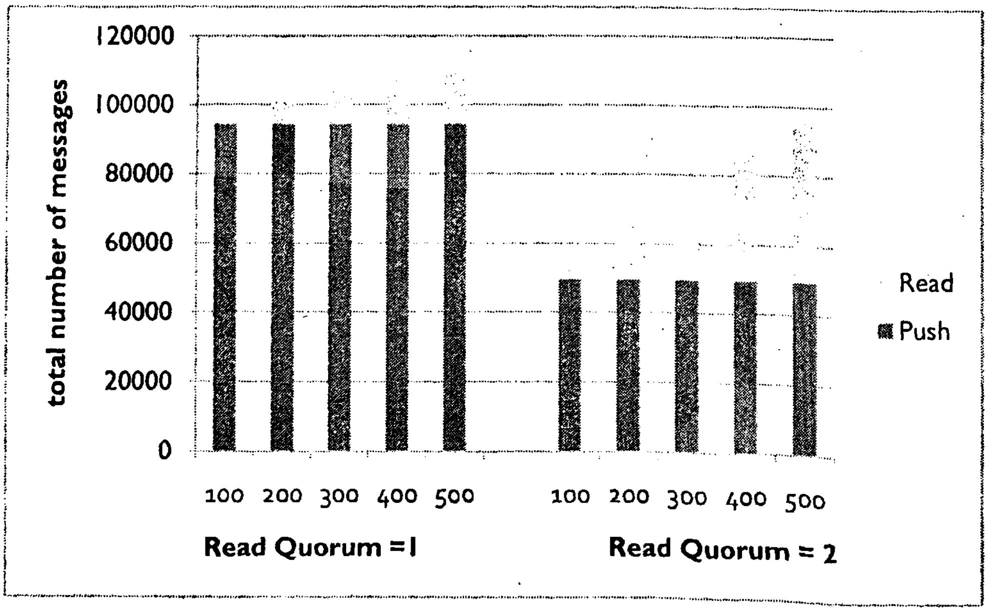

Figure 5.20: Total number of push and read messages for PtPU with read quorum level 1 and 2

Figures 5.19 and 5.20 present our experimental results for PtPU with read quorum level 1 and 2 for a system where the updates are more frequent than reads. Particularly, the number of updates that are made is 1000 for both policies and the number of reads varies from 100 (10% of updates) to 500 (50% of updates). All the other parameters are as summarized in Table 4.1 and Table 5.1. As it is shown in both figures, as the number of reads increases, the percentage of consistent reads is decreased while the total nuber of messages is increased for both policies. This happens because the more often the reads are, the less pulls other peers have performed. We also notice that when PtPU with read quorum level 2 is used, the percentage of consistent reads is smaller than PtPU with read quorum level 2 but the message overhead is smaller. There is a tradeoff between the incurred message overhead and the achieved consistency levels; depending on the system, we can choose to sacrifice the consistency levels of the data item read in order to achieve less communication cost.

CHAPTER 6. RELATED WORK

In Chapter 6 we survey replication methods applicable to unstructured p2p systems. Although there exist some general techniques, methodologies are distinguished according to the issue they are targeted for (how many replicas should be created or where should they be placed at). After replicas are created and distributed, a major issue is their maintenance. We present strategies that have been proposed for keeping replicas up to date so as to achieve a desired level of consistency.

6.1. Replication in unstructured p2p networks

Number of replicas: Assume that there are n peers participating in the network and s different data items to be shared among peers. Each peer on average has a storage capacity for storing c replicas of data items and the network has a total budget of R copies overall ($R = nc$). The *query rate* or *popularity* of item x, q_x, is the probability that any arbitrary peer issues a request for item x.

The problem of determining what is the optimal replica configuration is discussed by Cohen & Shenker in [7], for overlays that are modeled as Erdos-Renyi random graphs. Specifically, the authors deal with the problem of how many replicas of each data item should exist in the network so that the search overhead for locating the item is minimized, with the constraint of fixed storage capacity in the network. Given the query rates for each data item, the objective is determining which fraction p_x of R

should be allotted to each data item x, so that the expected search size (ESS), i.e. the number of peers probed during the search process is minimized.

As mentioned in Chapter 3, two natural ways of replicating data items, namely uniform and proportional replication, are shown to be suboptimal under the above assumptions. In *uniform replication* (UR) the same number of replicas is created for each data item, regardless of its query rate. In *proportional replication* (PR) the number of replicas for each data item is proportional to the popularity of the item. Although it seems natural to create more replicas for more popular data items so as to favor most common queries, this is done at the expense of rare ones. In fact, it can be shown that the ESS for a successful query is the same for both uniform and proportional replication strategies. The optimal configuration, proved to minimize the expected search size, is *square-root replication* (SR), where the number of replicas of each data item is proportional to the square root of its query rate.

Since global knowledge is unavailable at each peer, the authors also consider ways of realizing square-root replication using simple distributed protocols. In one of the simplest, the number of copies created after a successful search is equal to the size of the search, i.e. the number of peers probed during search. At steady state, and under reasonable assumptions, this simple strategy can be shown to converge to SR. The only critical assumption is that the fixed storage capacity of each node is managed through replacement policies that do not depend on the identity and the query rate of the stored items. As such, at a full node, the item that must be deleted so as to make room for another replica cannot be given by usage-based policies such as LRU or LFU but rather by policies like FIFO or random deletions.

Notice that although the idea is quite simple, the size of the search is normally not known. Lv, Cao, Cohen, Li & Shenker [18] discuss two practical strategies that try to approximate the search size, namely owner and path replication. In *owner replication*, which is used in Gnutella, when a search for a data item is successful (only) the peer that initiated the search process stores a replica of the data item. In *path replication*, each query keeps track of the path it follows from the peer that issues the request to the peer that offers the data item. When the search succeeds, all peers in this path are forced to keep a replica of the data item. Clearly, path replication comes quite closer to approximating the search size and experimental results show that it comes close to achieving SR. Path replication is used on Freenet [6] where all nodes along the search path are forced to create a replica using an *insert* message. Each node keeps both the item and a pointer to the original data holder of the file. The replacement policy used to manage the finite storage space at each node is LRU. Subsequent incoming requests of evicted files, however, can still be served for much longer since the node also holds a pointer to the original holder.

Path replication works only for search strategies based on random walks. Even in such cases however, it may fail to discover the search size. If multiple walkers are used [7], only the successful ones will be used to create replicas while the others will be ignored, creating a number of replicas smaller than the total number of visited nodes. To closely approximate the number of probes, the *Pull-then-Push* (PtP) strategy is proposed [16] (already discussed in Chapter 3), where replica creation becomes a responsibility of the inquiring peers. PtP replication consists of two phases: the *pull phase* during which the requesting peer is trying to locate the desired data item and the *push phase* which begins after a successful search whereby the requesting peer transmits the data item and causes other peers to hold replicas of it. In order to achieve SR, the number of peers that are probed during the push phase should be equal to the number of peers that where probed during the pull phase. Therefore, it is essential that the same search strategy is used both for searching for the data item (pull) and the data item transmission (push) and with the same hop limit (TTL). Finally, every peer that is probed during the push phase is forced to hold a replica of the data item. PtP works for both flooding and random-walker based strategies and leads easily to SR.

For Erdos-Renyi random graphs, if flooding-based search is used and if the objective is to minimize the search *time* (as opposed to search size) then proportional replication is the optimal configuration as shown by Tewari & Kleinrock in [25]. Search time is the distance from the inquiring node where a replica of the queried item is found. Optimality is achieved under the assumption of an ideal "controlled" flooding strategy where search stops immediately when the data item is located. A practical but slightly suboptimal search mechanism that approximates controlled flooding is the expanding rings method described in [7]. PR has additional benefits as well, e.g. the minimization of used network bandwidth (estimated as the average number of links traversed per download). Tewari & Kleinrock in [25] additionally consider practical ways of achieving PR. They basically follow owner replication (an inquiring node keeps a copy for itself), which should naturally lead to a number of replicas proportional to the request rates of data items. Again, a crucial factor is the replacement strategy used in managing each node's fixed storage space. Experimentally, all known strategies have good but not optimal performance, with LRU and LFU the better ones. Almost perfect PR can be achieved with a replacement strategy based on random evictions combined with additional replica creations even if the item is found in the inquiring node's storage space.

Placement of replicas: The works presented so far deal mostly with determining the optimum number of replicas and with ways to achieve this number, under certain assumptions and constraints. Another approach is to determine *where/how* to place the replicas (without striving for a particular number of them) so as to optimize some objective. For example, the objective may be the minimization of search size or the maximization of the percentage of successful searches.

Gia [5] has been proposed as an improvement of Gnutella to exploit peer heterogeneity and includes mechanisms that dynamically adapt the overlay topology and the search algorithms. The topology adaptation mechanism ensures that high-capacity nodes are the ones that have high degree. Gia follows *one-hop replication*: an index of the content of every peer is replicated to its immediate neighbors. The rationale behind this is that since high-degree nodes are visited more frequently and high-degree nodes are the ones with high capacity, having them know the content of their neighbors will make them capable of providing answers to a greater number of queries.

Jia, Pei, Li & You [14] compare various mechanisms for the problem of replica placement in power-law networks. They consider replication of location information (i.e. not the actual data) so as to maximize the overall performance of search queries.

The spread mechanisms considered are flooding, percolation-based (randomized) flooding, random walks and high-degree random walks (HDRW). The later is a variation of random walks where a visited peer selects the next peer randomly among its highest-degree neighbors. By spreading location information along an HDRW, more information reaches high-degree nodes more quickly. As a result, because it is well known that search queries gravitate towards the high-degree nodes in the network, potentially more searches will be resolved successfully and quickly. This was confirmed through simulations which showed that for the same message overhead, spreading replicas by HDRW results in better search performance than the other mechanisms, under both flooding-based and random walk-based search.

Morselli, Bhattacharjee, Marsh & Srinivasan [19] propose *LMS* (Local Minima Search), a search method and replication protocol. Assuming that both peers and data items obtain ids uniformly at random from a given large set (so as to guarantee uniqueness with high probability), the replication mechanism tries to replicate an item with id x to peers with id 'close' to x. Such a node is called a local minimum for item x in that its id is closest to x among the ids of all peers in the node's h-hop neighborhood, where h is a given parameter. A random walk is used first, followed by a deterministic walk that progresses towards the closest local minimum node by selecting at each step the neighbor with the smallest distance from x. When this random local minimum is reached, a replica is created if there is not one there already; otherwise, the process is repeated with a random walker of double length. For locating the item, the same procedure is used. A local minimum that receives the query replies with the replica or with a failure message depending on whether it stores the item or not. To improve success rate and response time, multiple such walkers can be utilized. The protocol can achieve quite high query success probabilities but at the expense of a possibly large number of replicas ($O\left(\sqrt{n/d_h}\right)$, where d_h is the minimum size of an h-hop neighborhood), which can be a problem if the storage space in each peer is limited.

Maximization of the probability of success is also the subject of the work by Sozio, Neumann & Weikum [23]. They consider the problem of replica placement in arbitrary networks that are searched by random walks. Given the peer capacities and the query rates q_{xj}, i.e. the fraction of all queries (issued in the whole network) for data item x by peer j, the problem of finding an assignment of replicas to peers so as to maximize the probability of a successful query is shown to be related to the multi-knapsack problem, where there is a set of bins with given capacities and a set of elements each with size and profit and the aim is to find a feasible packing that maximizes the profit. The problem can be tackled by good approximation algorithms, which however are centralized. The authors present *P2R2*, a distributed algorithm to solve the problem, which is based on each peer j keeping a special counter for each data item x, r_{xj}. The counter r_{xj} is incremented for each query about x that passes through node j and is unsuccessful or is satisfied by a peer with larger id. This requires that certain information is piggybacked on the query messages and that random walks are always unfolded to their maximum length even if the item is located at some step earlier than the expiration of TTL. P2R2 leads to a probability of query success which is within a factor of 2 from the optimal.

Summary: Replication methods that are applicable to unstructured p2p systems provide answers to the questions of how many replicas are created for each data item, according to which optimization criteria, and where those replicas are placed. Table 6.1 summarizes how replication methods described above deal with each of these issues.

Table 6.1: Summary of replication methods for unstructured p2p systems

	How many	Where/How	What	Goal
Sqaure-root Replication [18]	Proportional to the square root of the query rate of each data item	-	Data items	Minimum expected search size
Owner Replication [18]	Proportional to the query rate of each data item	Only to the requesting peer	Data items	Minimum expected search size

Path Replication [18]	Proportional to the number of probes for locating the item	Along the path from the requesting peer to the provider peer	Data items	Minimum expected search size
Pull-then-Push Replication [16]	Proportional to the number of probes for locating the item	-	Data items	Minimum expected search size
Proportional Replication [24]	Proportional to the query rate of each data item	-	Data items	Minimum expected search time
Gia [5]	Equal to the degree of each node	1-hop neighborhood	Location information	Maximum success rate
HDRW [14]	Proportional to the number of probes for locating the item	Along a degree-biased search path	Location information	Good success rate and search size
LMS [19]	-	At peers considered as local minima for a data item	Data items	Good success rate and search size
P2R2 [23]	-	At peers resulting in greatest success rate for a data item	Data items	Maximum success rate

6.2. Updates in unstructured p2p networks

As mentioned above, the consistency mechanisms that have been proposed use a push-based and/or a pull-based propagation algorithm. One more possibility can be found in the work of Demers et al [9], who have applied the theory of epidemics to the problem of update propagation in a distributed environment, proposing a number of generic methods. The first method the authors examine is *direct mail*, where the owner of a data item contacts ('mails') all the other peers at every update. This approach, although simple, can be overwhelming in a p2p network with a large number of nodes. In the *anti-entropy* method each peer regularly chooses a neighbor and by exchanging their content resolves any differences between them (if a newer version of an item is found, it updates its own replica). A peer can either push its content to the other peer letting it check for inconsistencies, or pull content, or even push and pull content at the same time. Another update spreading algorithm considered is *rumor mongering*: at first all peers are considered 'ignorant' when an update is out and the update becomes a 'hot rumor'. If a peer knows of such a rumor, it periodically chooses another peer and tries to communicate the rumor. If the peer sees that the rumor is no longer hot (i.e. most of the peers it contacts already know it), it stops propagating it any further.

If the direct mail method is to be used, a natural plan would be to know (most of) the peers that hold a replica of the particular data item (statefull replication) so as to only contact those upon an update. A mechanism like this is assumed by Datta, Hauswirth & Aberer in [8]. The authors study the performance of a generic hybrid push-pull consistency maintenance protocol for p2p environments where peers join and leave the network at a very high rate. At the push phase, the owner sends the updated item, along with its version number, to the peers that hold replicas. This requires knowledge of who holds replicas of what, but the update is not communicated through direct mail; it is rather propagated with a randomized flooding among the affected peers. The owner performs a selective push of its updates to a subset of the peers that will be affected by it (because they have a replica of the updated data item); each peer that receives the update also propagates it to a subset of affected peers it knows, and so on. To reduce the overhead, each message contains a partial list of the peers that have already been contacted. The method is accompanied by a pull phase that takes place whenever a peer is reconnected to the network after a disconnection or has not received updates for a long time (in the spirit of the anti-entropy method); during this pull phase, it contacts online peers with replicas of the items it stores, for their latest versions.

UPTReC – update propagation thought replica chain [28] – exploits similar pull and push mechanisms to scatter updates in decentralized and unstructured p2p systems. The peers that hold the replicas of an item x form a logical bi-directional chain, where each peer maintains information about the k closest peers in the chain in each direction. Peers may join (when a new replica is created) or leave (when removing a replica) by pushing messages at appropriate directions in the chain. Updates are similarly propagated by pushing messages at both directions, informing up to $2k$ nodes; at each direction the furthest known peer undertakes the responsibility of reaching the next bunch of k nodes in the chain and so on. Nodes that reconnect after a disconnection pull in order to synchronize. Maintaining such a chain for every item reduces the message overhead on updates while also providing better consistency levels than Datta, Hauswirth & Aberer [8] as shown experimentally.

Update propagation in the last two methods occurs strictly among the interested peers; although this seems efficient in terms of overheads and consistency levels, it nevertheless incurs the extra state overhead of keeping track of all peers holding a replica of the data item, which could be prohibitive in an unstructured and dynamic p2p network. Three update propagation policies (two based on push and pull techniques and a hybrid one that combines the push and pull policies) are proposed in [15] for practical networks. The authors assume a master-copy schema where the owner of the data item always has the most up to date version and all peers that hold a replica need to be kept consistent; the overlay network is unstructured and the owners do not know who/where replica holders are. To achieve consistency, each data item is associated with a version number which is incremented by the owner every time an update occurs. In the push-based policy, the owner of a data item broadcasts an invalidation message when a data item is modified. The invalidation message is propagated through the network using a flooding algorithm, limited to a predefined number of hops (TTL). When a peer receives an invalidation message, it checks its cache. If it holds a replica of the data item and the stored version is smaller than the received version number, it invalidates the replica in its cache. In the proposed pull-based policy, a peer polls the owner of an item it holds in its cache to determine if the replica is stale or not. An *adaptive polling policy* is used to determine how frequently the peer should poll. It is based on a time-to-refresh (TTR) value associated with each item in the cache, which indicates when the next pull for the item should occur. The TTR is increased by an additive amount C (TTR = TTR+C) if the peer finds out that a data item has not been modified between two successive polls, otherwise TTR is reduced by a multiplicative factor D (TTR = TTR/D). A hybrid push and pull approach can also be used to combine both techniques. In this hybrid scheme, the owner propagates invalidation messages using a limited push. In addition, a peer that holds a replica may pull adaptively to make sure that the replica is valid. TTR can be further tuned by a factor that depends on the degree of a peer; the intuition behind this is that highly connected nodes should poll less frequently since they are potentially easier to reach by the owner push.

An alternative hybrid push/pull update propagation policy, *PtPU* (already discussed in Chapter 5), is in [16]. It is assumed that for the creation of replicas in the p2p network the Pull-then-Push algorithm was used where a peer that requests an item, after a successful search (pull phase) enters a push phase where it transmits replicas of the item using the same algorithm as in the pull phase. Given this replica creation approach, each peer that holds a data item is characterized as *owner* if it is allowed to apply updates, *responsible* if it has requested the data item and has forced the creation of replicas or *indifferent* if it has been forced to hold a replica without requesting the data item. In the PtPU policy, the owner performs a limited broadcast of the new version of a data item when an update occurs. If a peer that is characterized as responsible for an item receives the broadcast message with a new version of the data item, it undertakes the task of informing the indifferent peers. This is done by propagating the update message (*U-push phase*) exactly as in the push phase when the replicas were created. Apart from pushing the updates they receive from the owner, responsible peers also pull periodically in order to become aware of more updates. To determine the frequency of the pull, the adaptive polling policy is used, where a TTR value is increased or decreased depending on weather the data item has been changed or not between two successive poll periods.

Wang, Kumar, Das & Shen in [29] consider multi-master replication where all replica holders (termed "replica peers" – RPs) are allowed to update the item. In particular, a subset of RPs become "virtual servers" (VRPs) for the data item. The set of VRPs changes dynamically over time, based on node availability. Any replica peer updating the item contacts a VRP to undertake the update coordination. This "master" VRP first enters an agreement phase with the other VRPs in order to commit the update. When agreement is achieved, the master VRP obtains the updated item from the replica peer and pushes it to the rest VRPs and to a partial list of the other RPs. Among the other RPs the update propagation is implemented using a combination of push and pull. The protocol achieves one-copy serializability.

CHAPTER 7. CONCLUSIONS

7.1. Summary

The peer-to-peer communication model has attracted considerable attention over the past few years as a new network model, which is widely used for data sharing. Unlike the client-server model, p2p model does not require a central node (a server) to provide access to shared resources. The peers participating in a p2p system construct a logical (overlay) network that is built on top of the physical one (typically the Internet). Any peer in the network can pose a query for retrieving a particular data item. Such queries are forwarded through the overlay network until peers that hold the data item in request are located. Replication has been proposed for improving the delay of a search process, robustness against frequent peer failures or departures and system availability. However, the effectiveness of a replication strategy seems to be influenced by the topology of the overlay network. The topology of a p2p overlay network shows evidence of power-law behavior [11]. This property means that peers in the overlay network are not connected randomly, but most peers are connected to a limited number of peers. Peers with many connections have a big influence on the stability of the overlay network. For this reason, we have defined and analyzed some basic properties from graph theory that are reflected to real networks, including unstructured p2p networks, and described a model for constructing network graphs that obey power-laws.

We have performed a series of experiments in order to investigate the effect of the overlay network topology on some replication strategies that exist in the bibliography. Our results show that in power-law networks, the well-connected peers cause large communication cost part of which can be avoided. The problem that we deal with is how can we reduce the communication cost imposed on networks with power-law topology. We have also studied the problem of maintaining consistency of replicas when updates in data items occur. Particularly, we have investigated the influence of the power-law overlay network topology on some known consistency maintenance protocols through simulation experiments. Our result revealed that part of the message overhead induced by those methods could be avoided if the power-law properties are taken into consideration. Based on this observation, we have proposed a new update policy that is intended for p2p systems with power-law overlay network topology. Finally, we have proposed an alternative approach for maintaining consistency in distributed, dynamic p2p systems that is based on the traditional quorum consensus.

7.2. Future Work

Throughout our simulation studies, we have assumed that the network does not change during the execution of a replica creation or an update phase. The proposed strategies were applied to practically static networks. However, applying those strategies on a more dynamic environment where peers join and leave the network at will, would be an interesting topic for further research. When it comes to the DPtPU update policy that is used, in highly dynamic networks, we encounter the problem that if a "responsible" peer leaves the system, a number of peers are left with replicas that will remain stale as no one will communicate any updates to them.

It would also be useful to study the behavior of the proposed strategies on a real p2p application so that we could obtain more realistic estimation of the real search delay or the communication cost, taking into account the shortcomings that are encountered in real-life (network bandwidth, traffic). Finally, adopting a quorum-based technique for unstructured p2p networks, especially in networks with power-law topology, remains an open research area.

REFERENCES

[1] Abraham, I., Malkhi, D. (2005) Probabilistic quorums for dynamic systems., *Distributed Computing*, vol. 18, no. 2, (pp. 113-124).

[2] Alqaralleh, B.A., Wang, C., Zhou, B. B., Zomaya, A.Y. (2007). Effects of Replica Placement Algorithms on Performance of structured Overlay Networks, In *Proc. IPDPS 2007, IEEE Int'l Parallel and Distributed Processing Symposium* (pp. 1-8). Long Beach, CA, USA.

[3] Barabási, A.L, Albert, R., (1999). Emergence of Scaling in Random Networks. *Science*, vol. 286, pp. 509-512, 1999.

[4] Chakinala, R.C, Kumar asubramanian A., Laing K.A, Manokaran,R., Pandu Rangan R., Rajaraman, R.(2006) Playing push vs pull. models and algorithms for disseminating dynamic data in networks.In Procs of *SPAA 2006*: (p.p) 244-253

[5] Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., and Shenker, S. (2003). Making gnutella-like P2P systems scalable. In *Proc. of SIGCOMM'03* (pp. 407-418). Karlsruhe, Germany.

[6] Clark, I., Miller, S.G., Hong, T.W., Sandberg, O., & Wiley, B. (2002). Protecting free expression online with Freenet. *IEEE Internet Computing*, 6 (1), 40-49.

[7] Cohen, E., and Shenker, S. (2002). Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proc. of SIGCOMM'02* (pp. 177-190). Pittsburgh, Pennsylvania, USA.

[8] Datta, A., Heuswirth, H.,and Aberer, K. (2003). Updates in highly unreliable, replicated peer-to-peer systems. In *Proc. of ICDCS 2003, 23rd Int'l Conference on Distributed Computing Systems* (pp. 76-85). Providence, Rhode Island, May 2003.

[9] Demers, A., Green, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. (1987). Epidemic algorithms for replicated database maintenance. In *Proc. PODC 1987, 6th Annual ACM Symposium on Principles of Distributed Computing* (pp. 1-12). Vancouver, Canada.

[10] Erdős, P., Rényi., A, (1960), On evolution of random graphs., *Publications of the mathematical Institute of the Hungarian Academy of Sciences*, (p.p 17-61).

[11] Faloutsos M., Faloutsos P., Faloutsos.,(1999). On power-law Relationships of Internet Topology, In Proc. of *ACM SIGCOMM'99* (pp 251-262), Boston, USA.

[12] Gopalakrishnan, V., Silaghi, B., Bhattacharjee, B., and Keleher, P. (2004). Adaptive replication in peer-to-peer systems. In *Proc. ICDCS 2004, 24th Int'l Conference on Distributed Computing Systems* (pp. 360-369). Tokyo, Japan.

[13] http://en.wikipedia.org/wiki/Perron-Frobenius_theorem

[14] Jia, Z., Pei, B., Li, M. and You, J. (2005). A Comparison of Spread Methods in Unstructured p2p Networks. In *Proc. of ICCSA 2005, Int'l Conference on Computational Science and its Applications* (pp. 10-18). Singapore.

[15] Lan, J., Liu, X., Shenoy, P., Ramamritham, K. (2003). Consistency Maintenance in Peer-to-peer File Sharing Networks. In *Proc. of WIAPP'03, 3rd IEEE Workshop On Internet Applications* (pp. 76-85). San Jose, CA, USA.

[16] Leontiadis, E., Dimakopoulos, V. V., and Pitoura, E. (2006). Creating and Maintaining Replicas in Unstructured Peer-to-Peer Systems, In *Proc. of EURO-PAR 2006, 12th Int'l Euro-Par Conference on Parallel Processing* (pp. 1015-102). Dresden, Germany. Springer: LNCS 4128.

[17] Johnsonbaugh, R., Kalin, M., (1991). A Graph Generation Software Package, In Proc of 22nd SIGCSE Technical Symposium, (pp 151-154), San Antonio, Texas, USA, March 7-8, 1991.

[18] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker S. (2002). Search and replication in unstructured peer-to-peer networks. In *Proc. of ICS 2002, 16th ACM Int'l Conference on Supercomputing* (pp. 84-95). New York, New York, USA.

[19] Morselli, R., Bhattacharjee, B., Marsh, M.A., and Srinivasan, A. (2005). Efficient Lookup on Unstructured Topologies. In *Proc. PODC 2005, 24th Symposium on Principles of Distributed Computing*. Las Vegas, NV, USA.

[20] OMNET++ - A tool for simulation programming available at http://www.omnetpp.org

[21] Palmer, C.R., Steffan, J.G., (2000) Generating network topologies that obey power-laws, In Proc. *Global Internet Symposium (GLOBECOM)*, pp. 434-432, San Francisco, CA, USA.

[22] Roussopoulos, M., and Baker, M. (2003). CUP: Controlled Update Propagation in peer-to-peer networks, In *Proc. of the Annual USENIX Technical Conference* (pp. 167-180). San Antonio, Texas, USA.

[23] Sozio, M., Neumann, T. and, Weikum, G. (2008). Near-Optimal Dynamic Replication in Unstructured Peer-to-Peer Networks. In *Proc. PODS'08, 27^{th} ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (pp. 281-290). Vancouver, BC, Canada.

[24] Susarla, S., and Carter, J. (2005). Flexible Consistency for Wide Area Peer Replication. In *Proc. ICDCS 2005, 25th IEEE Int'l Conference on Distributed Computing Systems* (pp. 199-208). Ohio, USA.

[25] Tewari, S.,and Kleinrock, L. (2006). Proportional Replication in Peer-to-Peer Networks. In *Proc. of INFOCOM 2006* (pp 1-12), Barcelona, Spain.

[26] Tewari, S.,and Kleinrock, L. (2005). Analysis of search and Replication in Unstructured Peer-to-Peer Networks, In *Proc. of SIGMETRICS 2005* (pp 404-405). Banff, Canada.

[27] Vecchio D. and Son, S. H. (2005). Flexible Update Management in Peer-to-Peer Database Systems. In *Proc. IDEAS 2005, Int'l Database Engineering and Applications Symposium*. Montreal, Canada.

[28] Wang, Kumar, M., Das,S.K and Shen, H. (2006) File Consistency Maintenance Through Virtual Servers in p2p Systems, In *Proc. 11^{th} IEEE Symposium on Computers and Communications (ISCC'06)* (pp. 435-441). Sardinia, Italy.

[29] Wang, Z., Das, S. K., Kumar, M., and Shen, H. (2007). An efficient update propagation algorithm for p2p systems. *Computer Communications* 30, 1106-1115.

SHORT CV

Mirto Ntetsika was born in Ioannina, Greece in 1984. She was admitted at the Computer Science Department of the University of Ioannina in 2002 and obtained the BSc degree in 2006. Since the same year she is an MSc student at the same department. She is a member of the Distributed Management of Data (DMOD) Laboratory since 2007. So far, her research was mainly focused on Databases and Peer-to-Peer systems, with particular interest on issues concerning replication and consistency maintenance on Peer-to-Peer systems with non-uniform overlay topologies, such as power-law networks.