J Supercomput (2009) 48: 15–42
DOI 10.1007/s11227-008-0203-3

Peer-exchange schemes to handle mismatch in peer-to-peer systems

Tongqing Qiu · Edward Chan · Mao Ye · Guihai Chen · Ben Y. Zhao

Published online: 30 April 2008

Abstract A self-organizing peer-to-peer system is built upon an application level overlay, whose topology is independent of an underlying physical network. A well-routed message path in such systems may result in a long delay and excessive traffic due to the mismatch between logical and physical networks. In order to solve this problem, we present a family of Peer-exchange Routing Optimization Protocols (PROP) to reconstruct the overlay. It includes two policies: PROP-G for generic condition and PROP-O for optimized one. Both theoretical analysis and simulation experiments show that these two protocols greatly reduce the average latency of the overlay and achieve a better logical topology with low overhead. Their overall performance can be further improved if combined with other recent approaches. Specifically, PROP-G can be easily applied to both structured and unstructured systems without the loss of their primary characteristics, such as efficient routing and anonymity. PROP-O, on the other hand, is more efficient, especially in a heterogenous environment where nodes have different processing capabilities.

Keywords Peer-to-peer · Distributed hash table · Mismatch · Topology-aware

1 Introduction

Peer-to-Peer (P2P) systems are massively distributed computing systems in which peers (nodes) communicate directly with one another to distribute tasks, exchange

T. Qiu · E. Chan (✉)
Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong
e-mail: csedchan@cityu.edu.hk

T. Qiu · M. Ye · G. Chen
State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, China

B.Y. Zhao
Department of Computer Science, University of California, Santa Barbara, CA, USA

information, or share resources. There are currently several P2P systems in operation and many more are under development. Gnutella [1] and Kazaa [2], which are often referred to as the first generation P2P file sharing systems, construct the unstructured overlay without rigid constraints for search and placement of files. They use a decentralized file lookup scheme. Requests for files are flooded with a certain scope. However, there is no guarantee of finding an existing file within a bounded number of hops. Chord [3], Pastry [4], and Tapestry [5] are examples of the second generation of peer-to-peer systems. These systems can be viewed as providing a scalable, fault-tolerant distributed hash table (DHT). Any data item based on a unique identification can be located within a bounded number of hops using a small per-node routing table. Unstructured P2P systems are widely used due to their simplicity; but structured systems can be more efficient. Consequently, these two models coexist and in some sense complement each other [6].

All P2P systems are built upon application-level overlays, the topology of which is independent of the underlying physical network. In unstructured systems, a new node randomly chooses some existing nodes of the systems as its logical neighbors; while in structured ones, a new node will get an identification by certain hash function and construct connections with other nodes based on specific rules of the DHT. As a result, the neighborhood of two nodes on the top of overlay does not inherently reflect proximity in the physical network, due to an arbitrary organization or the hash-based property. A well-routed message path in an overlay network with a small number of logical hops may lead to a long delay. The mismatch problem between the overlay and physical network is a major obstacle in building an effective large-scale overlay network.

Another important issue is the dynamic nature of peer-to-peer systems, in which peers can arrive or depart at any time. Without a timely reconfiguration mechanism, the logical overlay will stray from the optimal condition as inefficient routes gradually accumulate in the routing tables.

In this paper, we propose a family of Peer-exchange Routing Optimizing Protocols (PROP) to handle the mismatch in peer-to-peer systems. It includes two relevant policies: PROP-G (generic) and PROP-O (optimized). They both adaptively adjust the connections of the overlay, and efficiently reduce the average logical link latency of the whole system. Combining them with other recent mechanisms will further improve their performance. Moreover, they are adaptive to dynamic changes in the system. PROP-G, to the best of our knowledge, is the first scheme that can be deployed effortlessly on both unstructured and structured P2P systems, while preserving the logical topology of overlay at the same time. PROP-O, on the other hand, is more efficient, especially in a heterogenous environment where nodes have different processing capabilities.

The rest of paper is organized as follows. In Sect. 2, we review related work. Section 3 describes the design of PROP. In Sect. 4, we evaluate the effectiveness of our design analytically. The methodology and results of simulation experiments are presented in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Related work

The issue of mismatch between physical and logical networks in P2P systems has been the focus of intensive research in recent years. A location-aware topology matching (LTM) technique [7] is proposed for unstructured P2P systems. In LTM, each peer issues a detector in a small region so that the peers receiving the detector can record the relevant delay information. Based on the information, a receiver can detect and cut most of the inefficient and redundant logical links and add closer nodes as its direct neighbors. LTM is a typical method which is only applicable for Gnutella-like overlay networks where each peer can freely cut and add connections. Moreover, free modification of connections, to some extent, impairs the natural feature of self-organizing overlay where powerful, reliable nodes always provide more services and inherently have more connections [8]. Other methods for unstructured systems like [9] and [10] share similar features with LTM and will not be discussed in detail due to space limitation.

Regarding structured P2P systems, most solutions fall into three broad categories [11, 12].

- *Proximity Neighbor Selection (PNS)*: The neighbors in the routing table are chosen based on their proximity. Pastry and Tapestry are examples.
- *Proximity Route Selection (PRS)*: Once the routing table is chosen, the choice of the next-hop when routing to a particular destination depends on the proximity of the neighbors. CAN is an instance.
- *Proximity Identifier Selection (PIS)*: The node identifiers are selected based on their geographic location. Topologically-aware CAN [13] is an example in this category.

However, all of these approaches have a common limitation: *protocol-dependence*. For example, the entries in routing table are deterministic in systems like Chord or CAN, where the PNS scheme cannot be applied directly.[1] Similarly, PRS also has the requirement that there must be more than one choice for the next hop. Topologically-aware CAN, which ensures that nodes which are close in the network topology are close in the node ID space, is only suitable for systems like CAN [14], where the similarity of node IDs means less hops in routing. In short, recent methods based on DHT cannot be applied to other variants of the DHT protocols, not to mention other unstructured P2P systems.

Recently, some researchers focus on the configuration of AS or ISP level [15, 16]. Although this kind of central or cluster-like management can improve the efficiency of the system, it is more related to the deployment of different nodes instead of the deployment of end systems. Moreover, such control is impractical in loosely organized peer-to-peer systems.

There is some overlap between mismatch and another issue: heterogeneity in P2P systems. We classify heterogeneous factors into two categories: *network heterogeneity* and *node heterogeneity*. In a large-scale area like the Internet, the connections

[1] The P2P systems mentioned here are the original ones. Our goal is to find an auxiliary way to make recent P2P systems more efficient. It is quite different from the idea of Gummadi et al. [11] whose aim is to determine if the routing geometry precludes choosing neighbors based on proximity. That is why they argue that ring geometry allows the greatest flexibility while we argue that the real system of original Chord cannot use PNS.

between any two nodes may vary in transmission delay and bandwidth. Things get more complicated in an overlay network as a logical connection of an overlay is composed of several physical connections. Handling the mismatch problem in some sense amounts to exploiting the variations in network connections. Brocade [17] is an early attempt to exploit network heterogeneity. It constructs a secondary overlay of super-nodes to improve routing efficiency. Similarly, Xu et al. construct an auxiliary expressway network to take advantage of heterogeneity [18]. Accordion [19] maintains variable size routing tables to handle the efficiency versus bandwidth tradeoff over a wide range of operating conditions.

Heterogeneity is not limited to network connections. Different nodes have various capabilities, including processing and storage size. SmartBoa [20] categorizes the nodes into different levels based on the capability of the nodes. Heterogeneous information can be broadcasted in this tree-like structure. Gia [21] modifies the Gnutella protocol to ensure that high capacity nodes are indeed the ones with high degree and that low capacity nodes are within short reach of higher capacity ones.

In this paper, we concentrate on the efficiency of PROP in a network-heterogeneous condition and introduce node heterogeneity when comparing PROP-G with PROP-O. We have previously explored using peer exchange schemes in overlay networks [30, 31] and this paper builds on these works and enhances both the theoretical analysis as well as experimental results of these prior works.

3 Design description

3.1 Motivation

Before presenting our proposed algorithms, we describe the overlay more formally. The overlay can be modeled by a directed graph $G = (V, E)$, where V is the set of nodes in the network, and E is the set of links between nodes. An edge xy in E means that x knows a direct way to send a message to y. For simplicity, we will not explicitly point out the direction of connections in this paper, and we will explain the reason to support this kind of simplicity later.

Figure 1 shows an example of the mismatch problem. Figure 1(a) represents a physical network with three nodes, where each number represents one unit of routing delay between two nodes in the physical environment. If every node is connected with each other, there will be no mismatch problem. Unfortunately, the overhead cost will be too high to manage such a fully-connected overlay, so that only a limited number of logical connections will be possible in practice. Here, we assume that at most two connections can be preserved when it is mapped to a logical overlay. There exists three different topologies (b), (c), and (d), where each dashed line stands for an indirect connection.

It is obvious that overlay (d) is optimal because it has the smallest accumulated delay ($AB + BC + AC$). In order to adjust the topology from (b) or (c) to (d), the intuitive operation is to cut a longer connection and add a shorter one. However, this *cut-add* operation cannot be performed freely without any constraints in many cases. First of all, the reconfiguration of topology should never introduce overlay

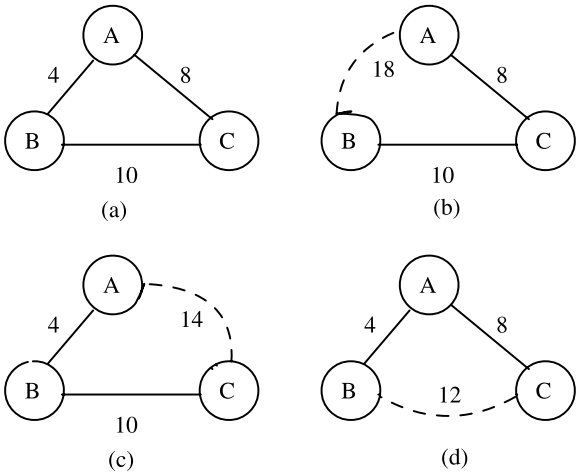


Fig. 1 An example of topology choices for overlay. (**a**) Physical network. (**b**)–(**d**) Overlay network

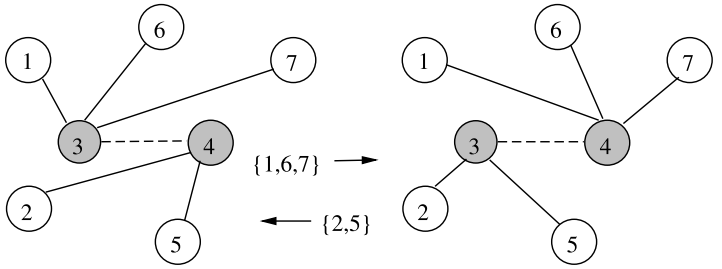
partitioning, which may significantly reduce the success rate of lookup or even lead to a collapse of the system. Furthermore, as an auxiliary method, this operation needs to be independent of the existing P2P protocols. In other words, adjustment of the logical overlay must not affect the existing original routing and searching algorithms, so that it can be plugged-in directly on to the underlying protocols. Finally, the traffic overhead of the reconfiguration algorithms should be small when compared with the traffic savings.

3.2 PROP-G and PROP-O

In order to satisfy the above requirements, we use "peer-exchange" as the basic operation of our scheme. Generally speaking, peer-exchange means a series of exchanges of some neighbors between two peers, and one exchange can be viewed as a pair of cut-add operations.

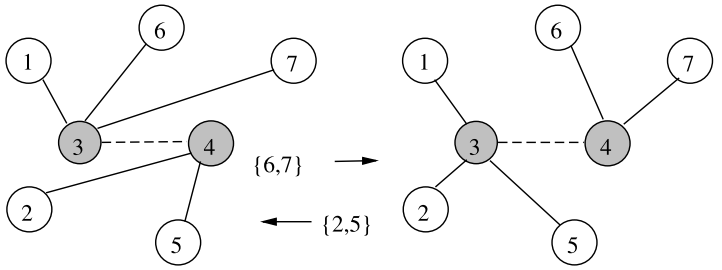
A simple and direct way of peer-exchange, called PROP-G, is to exchange *all* neighbors of the two nodes. Figure 2 shows an example of PROP-G, where nodes 3 and 4 exchange their neighbor sets ($\{1, 6, 7\}$ and $\{2, 5\}$). This can be viewed as exchanging their "position" in the overlay network. Intuitively, the topology of overlay is not affected by the PROP-G operation. That is why we call it a *generic* method, which will be proved in Sect. 4.

Another way for peer-exchange, called PROP-O, is to *selectively* choose neighbors for exchange. Figure 3 illustrates the process: nodes 3 and 4 exchange equal number ($m = 2$) of neighbors. Note that exchanged neighbors should never lie on the path of nodes 3 and 4, which ensures that nodes 3 and 4 will still be connected after the exchange. The primary reason that we exchange equal number of connections instead of an arbitrary number is to ensure the degree of each node remains the same after the exchange, so that the topology can maintain its essential features. The effectiveness and characteristics of both PROP-G and PROP-O will be illustrated by theoretical analysis in Sect. 4 and validated by simulations in Sect. 5.



(a) Before the exchange (b) After the exchange

Fig. 2 PROP-G, exchange all neighbors



(a) Before the exchange (b) After the exchange

Fig. 3 PROP-O, exchange m neighbors, where $m = 2$

A traditional way to accomplish topology optimization is to let each source node select one nearest node in the candidate list and establish the connection with it. This "selfish" method, in our opinion, is beneficial to the source node itself but is not always beneficial to (or in some case may actually detracts from) system-wide optimization. Our approach is to utilize the collaboration of two peers, say u and v, to discover potential opportunities to optimize their neighborhood environments, and then perform the exchange operation. In this way, reconfiguration of the overlay will improve overall system performance and avoid many, if not all, of the potential conflicts and pitfalls in "peer competition."

3.3 Description of basic method

Assuming that there is a potential exchange between nodes u and v, node u is the *counterpart* of v, and vice versa. $d(u, v)$ means the delay (latency) between nodes u and v. t_0 represents the time before an exchange, while t_1 represents the hypothetical time when the potential exchange really occurs. The neighbor set of node u is formally defined as follows:

$$N(u) = \{i \mid i \in V \wedge (ui \in E \vee iu \in E)\}. \quad (1)$$

```
time = INIT_TIMER;
add all neighbors into neighborQ;
while ntrial < MAX_INIT_TRIAL do
    s = neighborQ.pop;
    neighobrQ.addTail(s);
    make node s as destination of the first hop;
    find node v which is nhops away;
    exchange neighbor information with node v;
    measure Var when a potential exchange occurs;
    if Var > MIN_VAR then
        do exchange operation
    end
    ntrial = ntrial + 1;
    wait timer before next trial;
end
```

Algorithm 1: Initialization procedure

There are two separate procedures here: the warm-up and the maintenance phases. Having joined the system based on a random or DHT based assignment, a new node u will start the warm-up procedure, like in Algorithm 1. It begins probing its neighbors and collecting the initial latency information $\sum_{i \in N_{t_0}(u)} d(u,i)$. Then it will periodically contact a random node v which is *nhops* hops away at each time interval of *timer*. A priority queue *neighborQ* is used to choose nodes s for the first hop of random walk. The use of priority is to ensure that "active" nodes will be probed first, which is useful in the maintenance procedure. In the beginning, it is initialized with a random sequence of node u's neighbors, so each neighbor has an equal probability to be probed. A message containing the source IP address, the source timestamp and a small TTL value *nhops* is used to realize the random contact. Any node that receives this message will add an identifier like the IP address into the message,[2] decrement the TTL field by 1, and forward it. The target node v is located when TTL value becomes zero. Then nodes u and v selectively exchange their address lists and initial latency information with arbitrary m neighbors for PROP-O and all neighbors for PROP-G. So, the value of m is no more than the minimum degree of overlay $\delta(G)$. We choose $m = \delta(G)$ by default. After collecting the new latency information $\sum_{i \in N_{t_1}(u)} d(u,i)$ and $\sum_{i \in N_{t_1}(v)} d(v,i)$ by probing new neighbors (hypothetical neighbors when the potential exchange occurs), they exchange information and calculate the variable Var independently, as in the following equation:

$$\begin{aligned} \mathrm{Var} = & \sum_{i \in N_{t_0}(u)} d(u,i) + \sum_{i \in N_{t_0}(v)} d(v,i) \\ & - \sum_{i \in N_{t_1}(u)} d(u,i) - \sum_{i \in N_{t_1}(v)} d(v,i). \end{aligned} \tag{2}$$

[2] To avoid repetitive forwarding and exchange neighbors which stand on the random walk path.

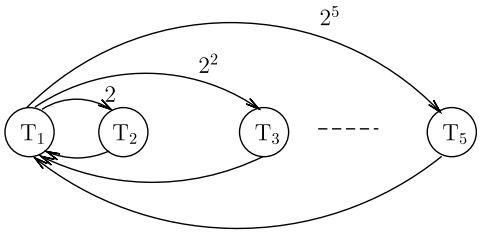


Fig. 4 A Markov chain of *Timer*

If Var $\leqslant$ MIN_VAR, it means that the exchange cannot gain any benefit, and, therefore, no subsequent operation will be performed. Otherwise, nodes u and v will do the peer-exchange operation as follows: they rewrite corresponding routing entries and even exchange node identifiers (for PROP-G in DHT systems), respectively. Both of them cache the address of their counterparts so that the lookups in progress during peer-exchange can be forwarded correctly. Moreover, both of them will notify their neighbors to change the routing tables and recalculate the initialized sums.

If the routing tables are extended to record both successor nodes and predecessor ones (bidirectional connections in other words), the notification can be realized directly. We have at least two reasons for this simplification. First, most structured systems selectively record several predecessor nodes in order to improve fault resilience. The size of the extended routing table is at most twice as large as the size of the original one. There is even no increase in some symmetrical systems like Gnutella or CAN. More importantly, even if there is no such extension, notifications can still be implemented by using the underlying mechanisms just as what happens when peers arrive or depart, although it leads to more complicated reconstruction operations. The warm up procedure will last for MAX_INIT_TRIAL times; simulations in a later section show this number to be less than ten.

Next node u will enter the maintenance phase, which differs from the initialization procedure in two ways. First, the selection of node s will depend on the result of peer-exchange trials. If an exchange occurs, which implies that selection of main "direction" is successful, node s will merely decrease the priority number by a small number like 1 so that it could be chosen in near future. Otherwise, it will be replaced at the tail of *neighborQ*, waiting for the next probing cycle. Another difference lies in the modification of *timer* based on a Markov chain model [22]: *Timer* will be doubled after a failed peer-exchange attempt, and reset to INIT_TIMER after a successful one; if *Timer* $\geq$ MAX_TIMER, it will also be set as INIT_TIMER. Here MAX_TIMER $= 2^5 \times$ INIT_TIMER, so there are at most five times of suspending (half of MAX_INIT_TRIAL). Figure 4 shows the process. Similarly, in order to handle departures and sudden failures gracefully, the value of *timer* will be reset to INIT_TIMER and the new neighbors will be added into the front of *neighborQ* with a maximum priority value, so that these peers can be probed earlier during the maintenance procedure. The details of the maintenance process is presented in Algorithm 2.

```
while timer expires do
    s = neighborQ.pop;
    make node s as the destination of the first hop;
    find node v which is nhops away;
    exchange neighbor information with node v;
    measure Var when a potential peer-exchange occurs;
    if Var > MIN_VAR then
        do peer-exchange operation;
        s.priority = s.priority − 1;
        timer = INIT_TIMER
    else
        s.priority = neighborQ.minPriority − 1;
        timer = min(timer ∗ 2, MAX_TIMER)
    end
    refresh neighborQ;
    wait timer before next trial;
end
if receive join/leave messages or detect failure entries then
    timer = INIT_TIMER;
    add new entries to the front of neighborQ;
end
```

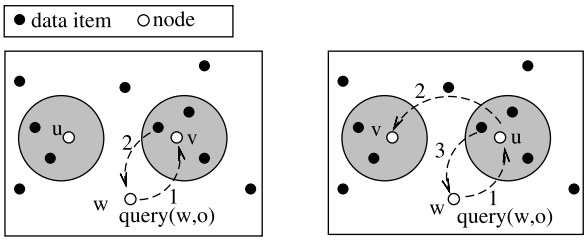
Algorithm 2: The procedure of maintaining connections

3.4 Technical details

3.4.1 Synchronization

The optimization of overlay is conducted simultaneously from multiple peer-exchange operations. When one node u probes a random node v, all neighbors of these two nodes should not change their states until the probing and peer-exchange operations complete. This ensures the correctness and accuracy of the exchange. We provide a simple mechanism based on a query-response model to synchronize operations between related nodes. Both u and v that decide to perform an exchange operation after the calculation of Var will broadcast an identical synchronization message to their neighbors. If those nodes do not receive other synchronization messages except this one, they send a corresponding acknowledgment message to node u or v. Once both u and v have received all the acknowledgments, the exchange operation is initiated.[3] If either u or v did not receive all the responses after a period of time, TIME_LIMIT, the exchange attempt is aborted. Recent studies show that flooding with a small number of TTL hops is highly effective and the synchronization based on this kind of flooding is acceptable [22]. Without flooding, the synchronization zone of our method is limited to the two exchanging nodes and their neighbors, resulting in lower synchronization overhead.

[3] It also requires a similar negotiation between nodes u and v.



(a) Before exchange (b) After exchange

Fig. 5 A solution of data movement. There is a query (w, o) from node w to query item o which resides on node v. After the peer-exchange, the query will be redirected by node u to the node v without data movement

3.4.2 Data movement

In a real peer-to-peer application based on DHT, data items reside on different nodes. After exchanging the routing tables and identifiers, the items may also need to be exchanged accordingly. The movement of data items will consume significant bandwidth, since each node may store many gigabytes of data.

However, a new node has already performed peer-exchange operations when it joins the systems, and during the warm-up procedure, it can start from a clean slate, i.e., without a large amount of data such as shared files from other nodes. The initial optimization without data movement can achieve performance improvement in a short time—less than 10 cycles of probing as shown in the simulation experiments.

In fact, even when nodes which share a lot of data are involved in the exchange operation, the overhead of data movement does not really have to be high if pointers of objects are kept, as is the case for many peer-to-peer data sharing systems [23, 24]. A typical solution is illustrated in Fig. 5. Both nodes u and v own a partition of data items. We assume that data item o resides on node v at the beginning. There is a query (w, o) from node w to query item o. The query process is shown in Fig. 5(a). After the nodes u and v have performed exchange operation, they will not exchange the items immediately. Instead, both of them own their counterpart's *pointer*, which contains one timer and the address information of the counterpart. So, the subsequent queries can be forwarded by the counterpart to the correct destination. For example, in Fig. 5(b), the query (w, o) will be redirected by node u to node v. When the nodes become stable and the timer expires, the data items will be exchanged. The initial value of the timer is related to the state of a node mentioned above. Since both the size and distribution of data items are typically application-specific, we will not go beyond the generic scheme proposed here and it will not be considered in our simulation experiments.

4 Theoretical analysis

4.1 Characteristics of peer-exchange

The basic requirement of overlay reconstruction, as mentioned above, is that the change of connections should never lead to a graph partition.

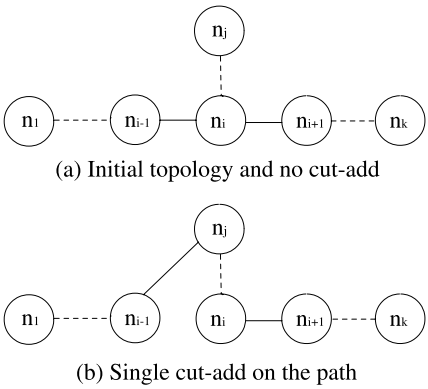


Fig. 6 A generic path where node n_j is off the path

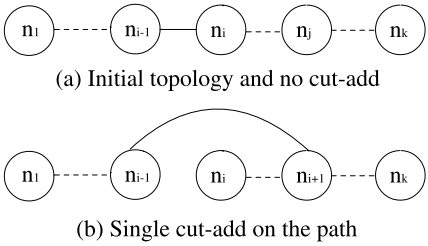


Fig. 7 A generic path where node n_j is on the path

Theorem 1 (Connectivity persistence) *Let G be an undirected connected graph, and let G′ be the graph that is derived from G by applying an exchange operation in PROP-G or PROP-O. G' is an undirected connected graph.*

Proof Both PROP-G and PROP-O consist of an exchange of several neighbor nodes. This exchange can be performed by a series of *cut-add* operations of two nodes: cut one connection of one's neighbor and add another one for its counterpart. Hence, we can restrict our attention to a single cut-add operation. It follows from induction that if the graph remains connected after a single cut-add operation, it remains connected after exchange. Let $P = \langle n_1, \ldots, n_k \rangle$ be an arbitrary sequence of nodes that forms a path in G. Node n_i will remove a connection between one of its neighbor, n_s, and itself. Then a connection between n_j and n_s is established.

Case 1: if n_i lies off the path. This means that while there may be nodes on the path whose edges change, the changed edges connect to the nodes implementing cut-add. Hence, no edges that form the path are changed, so the path remains after the cut-add is complete.

Case 2: n_i lies on, and n_j lies off the path, as in Fig. 6(a). Since nodes n_i and n_j are connected both before and after a cut-add (as mentioned in Sect. 3, exchanged neighbors should never lie on the probing path between nodes n_i and n_j, which ensures that two nodes will be still connected after the exchange), two possible scenarios occur: n_i cut no edges or one edge on the path. As can be seen in Fig. 6(b), a path between n_1 and n_k remains after the cut-add where $n_s = n_{i-1}$.

Case 3: Both nodes n_i and n_j lie on the path, as in Fig. 7(a). There are two similar scenarios as in Case 2: n_i cut no edges or one edge on the path. As can be seen in Fig. 7(b), a path between n_1 and n_k remains after the cut-add where $n_s = n_{i-1}$.

So the graph is still connected after a single cut-add operation. We can further conclude that G' is connected after a series of cut-add operations for both PROP-G and PROP-O. □

The above theorem ensures that there is no graph partition after a peer-exchange operation. Moreover, it is trivial to proof that PROP-O preserves the original degrees of each node, so it never breaks the natural Power-law-like characteristic (i.e., powerful nodes own more connections) of unstructured P2P systems.

Theorem 2 (Isomorphic characteristic) *Let graph $G(V, E)$ denote the network overlay, and let $G'(V', E')$ be the graph that is derived from G by applying an arbitrary sequence of PROP-G exchange operations. G' is isomorphic to graph G, i.e. $G \cong G'$.*

Proof It follows from induction that if the derived graph $G'' \cong G$ after a single exchange operation of PROP-G, then $G' \cong G$ based on the transitivity of isomorphism. Without loss of generality, we assume nodes u and v do a single exchange during the period t_0 to t_1. We try to find a bijection $\varphi : V \to V'$ with $xy \in E \Leftrightarrow \varphi(x)\varphi(y) \in E'$ for all $x, y \in V$. V_1 is used to denote the set of un-exchanged nodes and V_2 presents the set of exchanged ones. A mapping φ between E and E' is constructed as follows:

- For $\forall x, y \in V_1$, $xy \in E \Leftrightarrow xy \in E'$.
- For $\forall x \in V_1$, $y = u$, $xy \in E \Leftrightarrow xv \in E'$, $yx \in E \Leftrightarrow vx \in E'$. Similarly, for $\forall x \in V_1$, $y = v$, $xy \in E \Leftrightarrow xu \in E'$, $yx \in E \Leftrightarrow ux \in E'$.
- For $\forall x \in V_2$, $y \in V_1$, the proof is similar to the above one.
- For $\forall x, y \in V_2$, $xy \in E \Leftrightarrow yx \in E'$.

Observing the constructed mapping, it is easy to conclude that G is isomorphic to G'. □

Theorem 2 illustrates that PROP-G not only keeps the connectivity of logical network but also maintains the overlay topology. Therefore, as an auxiliary method, it is suitable for different topologies: ring, hypercube, tree, and so on. Moreover, the change of positions using PROP-G is not arbitrary. As an example in DHT systems, instead of regenerating its identifier, each node is only allowed to get old identifiers of other nodes. It preserves anonymity provides a certain measure of security.

However, it does not mean that PROP-G can be used in *all* P2P systems. In fact, there are several classes of P2P applications where neighbor relationships cannot be set arbitrarily. For instance, in some systems where each node has a certificate which binds its identifier to a public key for security reasons, it seems that PROP-G which exchanges node ID may not be feasible.

4.2 Effectiveness of the peer-exchange mechanism

We use the following definitions to explain the effectiveness of the peer-exchange mechanism. *Stretch* is defined as the ratio of the average logical link latency over

the average physical link latency. It is a common parameter to quantify the degree to which the physical and logical topology matches. *Average latency (AL)* is a basic parameter to quantify the property of a network. If there are n nodes in a network, then[4]

$$AL = \left(\sum_{i \in V} \sum_{j \in V} d(i, j)\right) \Big/ n^2. \tag{3}$$

Given that the physical network is usually static, only the average logical link latency affects stretch. Furthermore, supposing that the number of nodes is constant during the period t_0 to t_1, the *accumulated latency* (L_{t_i}) can be analyzed as follows. The next two equations show the accumulated latency at t_0 and t_1:

$$L_{t_0} = C + \sum_{i \in N_{t_0}(u)} \alpha_i d(u, i) + \sum_{i \in N_{t_0}(v)} \beta_i d(v, i), \tag{4}$$

$$L_{t_1} = C + \sum_{i \in N_{t_1}(u)} \gamma_i d(u, i) + \sum_{i \in N_{t_1}(v)} \delta_i d(v, i). \tag{5}$$

In (4) and (5), C represents the invariable part during the period t_0 to t_1. The coefficients of the summations $\alpha_i, \beta_i, \gamma_i, \delta_i$ represent the visited times of each neighbor-link when calculating AL. The equation $\alpha_i \approx \gamma_i \approx \beta_i \approx \delta_i$ is valid by assuming that each link has the same probability to be visited. To calculate the variation by (4)–(5), it is easy to find that if $\mathrm{Var} > 0$ then $L_{t_0} > L_{t_1}$, which implies that a peer-exchange reduces stretch. So in our simulation part, we will set $\mathsf{MIN\_VAR} = 0$.

We notice that the latency for other peers to reach a certain object on exchanged nodes might have been increased. For example, assume peer u and v exchange their identifiers and keep pointers to each other. Peer i was originally a neighbor of v, but is now a neighbor of u. If it tries to retrieve an object stored at v, it takes it two hops instead of one now. However, according to the above analysis, the average latency of all queries issued from all nodes to u and v will be decreased.

Note that this is only an approximate analysis. In fact, when the positions of the nodes change, the visited times of each node varies accordingly. This explains why not all exchange operations can reduce the average latency, as shown in the simulation experiments.

4.3 Stability

When the system are highly dynamic, it is reasonable to perform exchange operations to achieve better topology. But if the network is stable, is it possible that the exchange operation occurs continually? Will the oscillations occur?

If we only consider the exchange of two nodes, say A and B. If their neighbor condition is relatively stable, the exchange is one-way. So if nodes A and B exchange their logical positions, they will not exchange back directly. However, when we consider a chain of exchange operations involving three or more nodes, things are

[4]We assume the latency between a node and itself is zero.

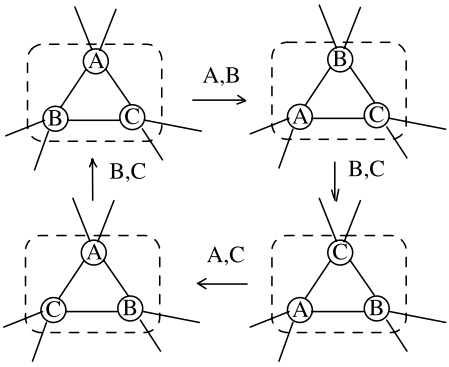


Fig. 8 Triangle churn—a chain of state transformation

different. A typical example is shown in Fig. 8. The lines between every two nodes here stands for the cooperations rather than direct connections. After four exchange operations, the topology can return to the original state. So, the exchange will never stop. We call it *triangle churn*. If $nhop = 1$, i.e., the three nodes are directly connected with each other, it is highly possible that triangle churn occurs: assuming that the average degree is c and the average probe times are n_p (the value is less than MAX_INIT_TRIAL during the warn-up procedure, and the probing frequency is lower during maintaining process), if the neighbor information determines that the topology will change according to the chain, the probability of churn when $nhop = 1$ is

$$p_1 = \begin{cases} 1 & \text{if } c \leq n_p, \\ \left(\frac{n_p}{c}\right)^4 & \text{if } c > n_p. \end{cases} \tag{6}$$

However, if the average probing time is constant, when $nhop = 2$, the probability of churn $p_2 \approx (\frac{n_p}{c^2})^4$, which is about $1/c^4$ of p_1. For example, if the average degree is around 10 in a Gnutella-like system, p_2 is 0.1% of p_1. In other words, when $nhop \geq 2$, the probability of churn is very low. Other chains involving more than three nodes will have a larger exponent number (6 for *quadrangular churn*), so they can be also omitted when $nhop \geq 2$. In the simulation part, we will also illustrate that the stretch become stable when $nhops \geq 2$.

4.4 Overhead analysis

Our method improves the overlay topology in two types of cost: the information collection between two cooperative nodes, and the reconstruction of overlay. Both of them are determined by two factors: the number of nodes involving into one potential exchange operation and the number of probing times. For an overlay network with n peers, we use c to denote the average number of neighbors. For each peer, one step of adjustment will involve $(nhop + 2c)$ for PROP-G, and $(nhop + 2m)$ for PROP-O. The overhead of PROP-O is intuitively better than PROP-G especially when c is much larger than $nhop$ and m. Our simulation will illustrate that. As for the second factor, we investigate the frequency of probing for each node, f_p. In the worst case, when each peer has to probe every time, the frequency will be $f_p = 1/\text{INIT\_TIMER}$.

Table 1 The choice of parameter

Symbol	Meaning	Default Value/Range
MIN_VAR	The minimum value of variation/improvement for exchange	0
MAX_INIT_TRIAL	The maximum value of warmup trials	10
INIT_TIMER	The initial value of time interval for probing	1 min
MAX_TIMER	The maximum value of timer to stop probe	128 min
n	The number of nodes in overlay networks	1200
nhop	The distance between two nodes for probing	[1, 4]
m	The number of nodes for exchange	[1, 4]

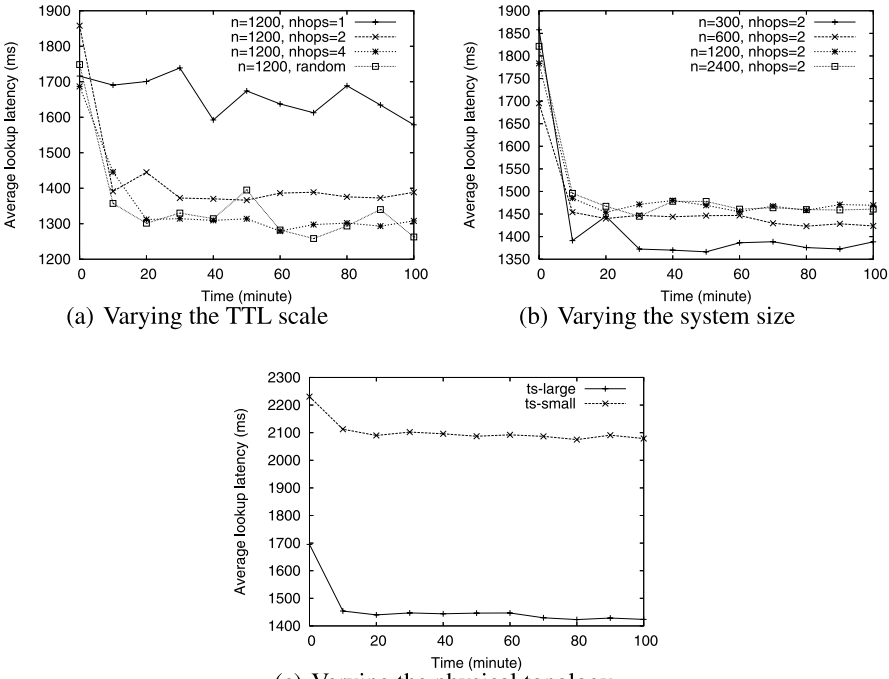
In fact, the topology will become stable after a warm-up procedure, and the frequency is very low after that because we utilize a Markov chain model to exponentially postpone the time of probing. Even when churn occurs, the frequency of probing will reduce quickly after a short period of time. Our simulation experiments regarding performance in a dynamic environment will demonstrate this idea.

5 Performance evaluation

5.1 Simulation methodology

We use the GT-ITM topology generator [25] to generate two different transit-stub models of the physical network. The first topology, *ts-large* has 70 transit domains, 5 transit nodes per transit domain, 3 stub domains attached to each transit node and 2 nodes in each stub domain. The second one, *ts-small*, differs from ts-large in that it has only 11 transit domains, but there are 15 nodes in each subdomain. Intuitively, ts-large has a larger backbone and sparser edge network than ts-small. Except in the experiment of physical topology, we always choose ts-large to represent a situation where the overlay consists of nodes scattered in the entire Internet and only very few nodes from the same edge network join the overlay. We also assign latencies of 5, 20, and 100 ms to stub-stub, stub-transit and transit-transit links, respectively. Then a number of nodes (default set to 1200), are selected from the physical network as overlay nodes.

The simulation involves three P2P infrastructures, Chord, CAN, and Gnutella; and different improving methods based on them like PNS, PIS and LTM. Table 1 shows the default value of some parameters. Among these parameters, MIN_VAR is determined by the analysis in Sect. 4.2. The value of MAX_INIT_TRIAL is based on massive experiments. It is difficult to set the value of INIT_TIMER because it is related to the dynamics of the system. In our evaluation, we simply set it as 1 minute. The choices of other parameters will be discussed in the following subsections.



(a) Varying the TTL scale

(b) Varying the system size

(c) Varying the physical topology

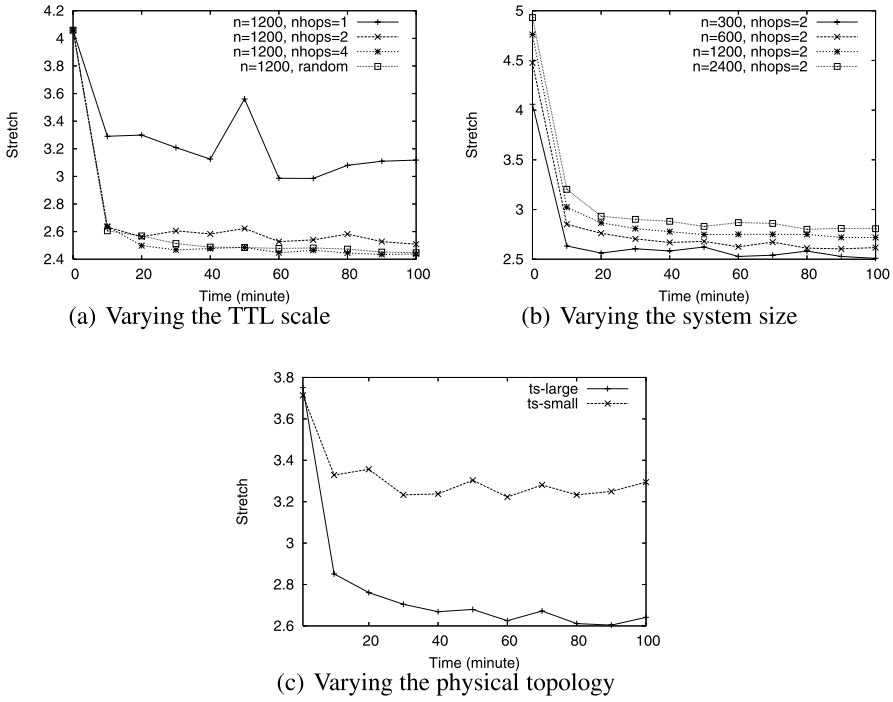
Fig. 9 Effectiveness of PROP-G in Gnutella-like environment

5.2 The effectiveness of PROP-G

5.2.1 Generic mechanism

According to the analysis in Sect. 4, PROP-G is a generic mechanism, which can be used in both unstructured and structured systems. Figures 9 and 10 show its effectiveness in both Gnutella-like and Chord environments. *Stretch* is the metric used to characterize matching degree. As messages are sent by the flooding method in unstructured P2P systems, it is not practical to calculate the latency between each pair of nodes. Therefore, the average lookup latency derived from 10,000 lookup operations is chosen in Gnutella instead. Both stretch and average lookup latency are varied according to time.

Figures 9(a) and 10(a) show the impact of the *TTL* on stretch in two different systems. There are four typical scenarios as far as probing is concerned. In the main scenario, instead of TTL packets, a random node is selected as the probing target. In other three conditions, TTL value *nhop* is set to 1, 2, and 4, respectively. Neighbors' exchange ($nhop = 1$) is not suitable because it cannot reduce the stretch significantly (which is consistent with our analysis of stability), while other three different ways have nearly the same impact on stretch reduction. Given that random probing is not practical in a distributed system, only when $nhop \geq 2$ can a good performance be attained in a P2P system. In order to minimize the cost, $nhop = 2$ may be a better



(a) Varying the TTL scale

(b) Varying the system size

(c) Varying the physical topology

Fig. 10 Effectiveness of PROP-G in Chord environment

choice, and it will be used in the following experiments. Figures 9(a) and 10(a) also illustrate that stretch is not reduced all the time, which is consistent with our approximate analysis.

Figures 9(b) and 10(b) demonstrate the impact of system size. The effectiveness of the schemes is slightly reduced as the system size becomes larger. There are at least two reasons. First, when the system has a larger size and PROP-G fixes *nhop* as 2, the collected information is relatively limited. Second, as we choose the nodes from the same physical network, the overlay is getting closer to the physical topology when it is larger. Fortunately, PROP-G is still effective even when almost all physical nodes are chosen.

The impact of physical topology is presented in Figs. 9(c) and 10(c). We have generated two different types of topologies: *ts-large* and *ts-small* by GT-ITM tools, both of which contain about 2400 nodes. It is obvious that the ts-large topology has much better performance. In the ts-large topology, only a few stub domains are attached to transit nodes. As a result, the probability that two stub nodes belong to different transit domains is relatively high. In other words, two *far* nodes can execute the exchange operation with a high probability, and this kind of exchange will greatly improve the performance of the system. PROP-G is more efficient in ts-large topology, which is much like the Internet as we mentioned above. Finally, comparing structured and unstructured systems, the average lookup latency in Gnutella fluctuates more markedly.

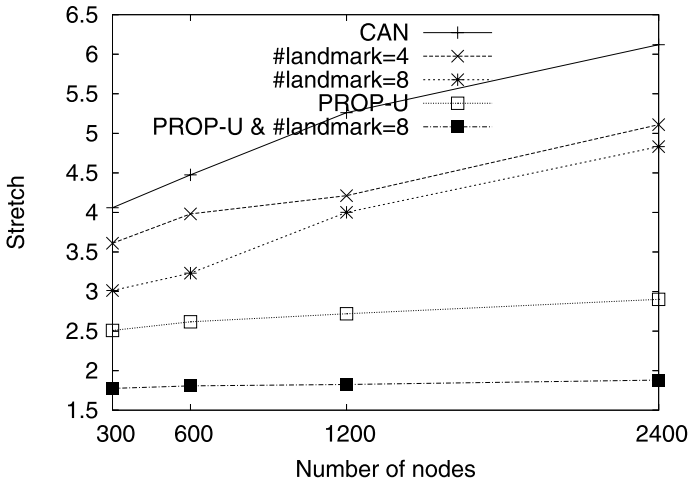


Fig. 11 A comparison between PIS and PROP-G

This is because Gnutella owns more random logical connections, and it is harder to find the better candidate nodes to exchange.

5.2.2 Comparison with PIS

As mentioned in a previous section, there are three kinds of optimization methods for structured P2P systems: PIS, PRS, and PNS. Since PNS and PRS share many common features, we will merely compare PROP-G with PIS and PNS. Landmark clustering, which is widely used in the PIS method, is based on the intuition that nodes close to each other are likely to have similar distances to a few landmark nodes. Ratnasamy et al. utilize this idea to optimize the CAN system [13]. By measuring the distance between some landmarks and itself, a new node joins CAN and locate a specific position. Figure 11 shows the comparison between PIS and PROP-G based on CAN. We choose two different number of landmarks: 4 and 8. It is obvious that PROP-G produces a better topology because landmark clustering is a coarse-grained approximation which is not effective in differentiating nodes within close distance. Furthermore, PIS seems more sensitive to changes in system size. The reason is that when the number of landmarks is fixed, the precise degree tends to decrease, which in turn lead to an increase in stretch. Moveover, since PROP-G is a protocol-independent method, it can be easily combined with landmark clustering. Figure 11 also shows that this kind of combination can further reduce stretch.

5.2.3 Comparison with PNS

We choose PNS-Chord, the proximity-aware version of Chord [26] to represent PNS schemes for comparison. In the original version of Chord, the ith finger table entry of the node with ID a refers to the first node in the ID-space range $a+2^i$ to $a+2^{i+1}-1$, while PNS(x) considers up to the first x nodes in that range, and routes lookups through the node with the lowest latency. We choose $x=4$ and $x=16$.

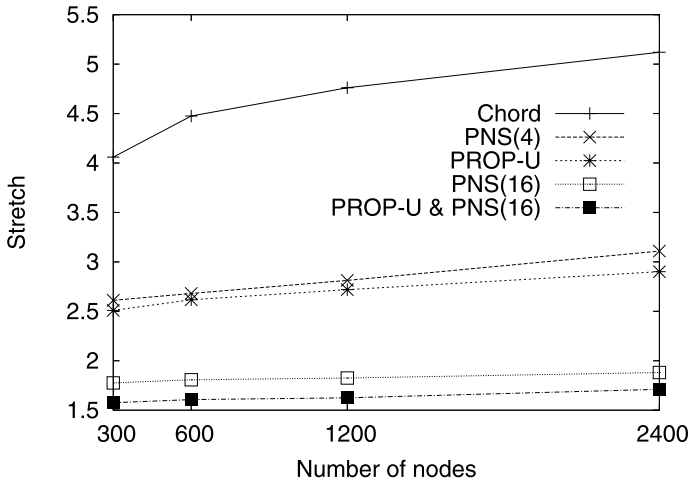


Fig. 12 A comparison of efficiency between PNS and PROP-G

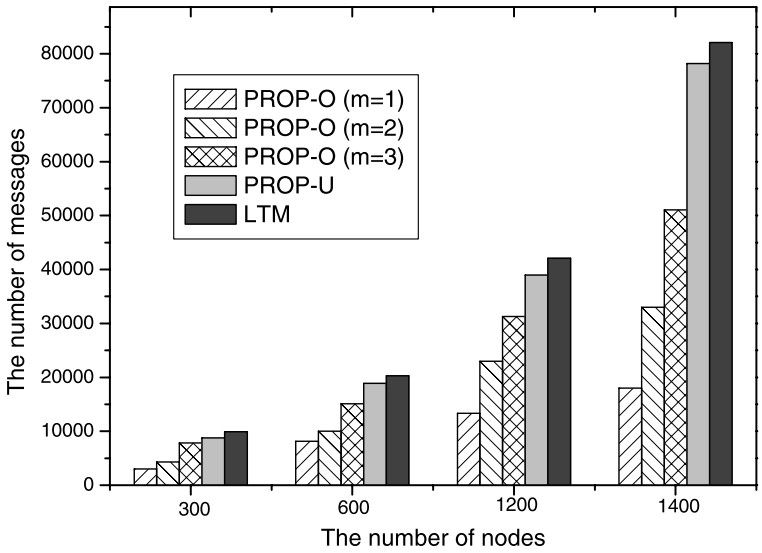


Fig. 13 A comparison of overhead between PNS and PROP-G

Figure 12 shows that PROP-G outperforms PNS(4), but is worse than PNS(16). However, PNS incurs a much higher overhead than PROP-G, especially when $x = 16$, as shown in Fig. 13. This is because in PNS-Chord, each node actually extends the finger table to maintain much more state information (though less than x times), which unavoidably introduce complexity into the system management process. Consequently, a lot more messages are required for join requests and forwards, distance

measurements, as well as status updates. PROP-G, on the other hand, simply tries to make adjustment in a limited area.

Moreover, just as with PIS, PROP-G can further reduce system stretch when combined with PNS(16), even though PNS(16) already achieves near optimal results for the PNS scheme as discussed in [26]. The selection range of the PNS method is limited (see [11] for a similar conclusion). PROP-G, on the other hand, can choose peers' positions to reduce the global stretch without constraints like ID prefix. Since the basic idea of PNS is to achieve global optimization in a greedy way to satisfy the requirement of each single node, whereas the peer-exchange method performs reconstruction based on a series of cooperation between pairs of nodes, so it is not surprising that exchange can complement PNS to a certain extent.

Although only PNS-Chord is presented in the experiment, we believe that the results will be similar when compared with other PNS schemes like Pastry or Tapestry because they share common characteristics with PNS-Chord.

5.3 PROP-O

5.3.1 Effectiveness in unstructured systems

In this section, we will compare PROP-O with PROP-G and LTM under a Gnutella-like environment. m is allowed to vary from 1 to 4, where 4 is the minimum average degree in the system. In order to illustrate the features of PROP-O in a heterogeneous environment, we further introduce node heterogeneity in our simulations. There are many resource factors which result in node heterogeneity, including process speed, storage and bandwidth supported. We merely use processing delay to represent node heterogeneity because we are more interested in lookup latency in this paper. To simulate processing delay, the *bimodal* distribution is used. There are two kinds of nodes—fast and slow. The processing delay of the fast nodes is 10 ms, while the delay of the slow ones is 100 ms. The fraction of fast nodes is 5% of the total population: the overall setting is similar to that in [27]. Since the total delay is just the sum of the link delay plus processing delay of nodes, the resulting absolute delay will be much larger than the corresponding results for PROP-G. To avoid any confusion, we choose a normalized value instead of real lookup delay which is measured by millisecond.

Figure 14 shows an important feature of PROP-O. In real-life P2P systems, powerful nodes provide much more services than poor ones. Accordingly, the destination of lookup operations will be concentrated on the powerful nodes. We simulate this phenomenon by increasing the fraction of lookups whose destination is a fast node in the *bimodal distribution* environment. When all queries are directed to slow nodes, LTM shows best routing performance. However, when more queries are directed to fast nodes, the delay of both PROP-G and LTM increase. On the other hand, the delay for PROP-O keeps decreasing. We will explain it from the following two perspectives. On the one hand, given that the physical network we construct is "Internet-like," only a few nodes from the same edge network will join the overlay. So, the query with largest latency is usually from a slow node to another slow one in different areas. As a result, it is likely that the latency of a query to slow node tends to be larger than the one to fast node. However, if the two slow end-nodes are connected by a number

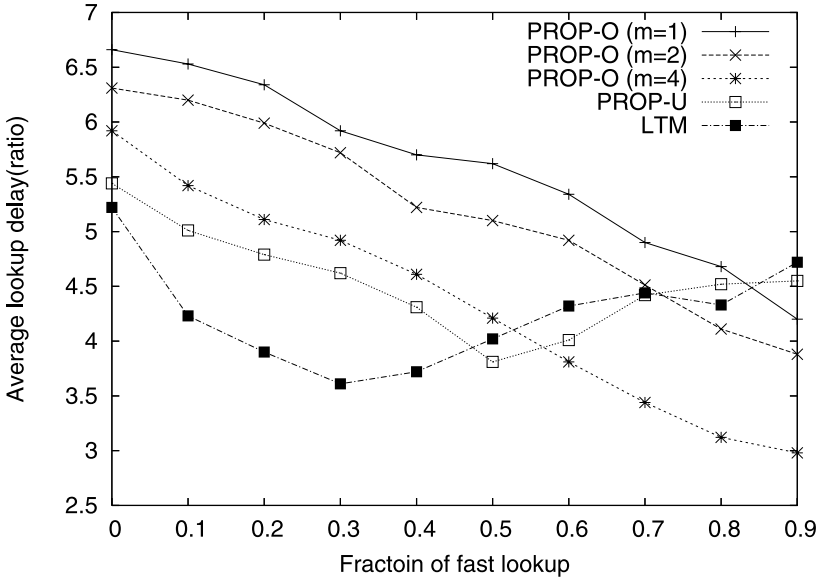


Fig. 14 Average lookup latency for bimodal processing delay distribution, when varying the fraction of fast node lookup

of fast intermediate nodes, maintaining the positions of these fast nodes will have a more significant impact on the performance of the system. Because fast nodes have more connections, it is more likely that they will be located in better positions after a peer-exchange, and PROP-O is able to maintain the connection number of each node so that the fast nodes can keep this kind of priority.

Another reason that we use PROP-O instead of PROP-G or LTM in unstructured systems is that it brings less overhead. For an overlay network with n peers, we use c_n to denote the average number of neighbors. For each peer, one step of adjustment will involve c_n^2 nodes for LTM, $(nhop + c_n)$ for PROP-G, and $(nhop + m)$ for PROP-O. The overhead of PROP-O is intuitively better especially when c_n is much larger than $nhop$ and m. We use the number of additional messages to represent the overhead. Figure 15 shows that PROP-O introduces less overhead, and the differences among the three methods become significant when the system size increases, in other words, when c_n is larger.

5.3.2 Limitations in structured systems

The above simulations show that PROP-O is better than PROP-G in unstructured P2P systems. So, another problem is that whether PROP-O can be applied to structured systems. Theoretically speaking, because all structured systems have certain requirements about the identifications of the neighbors, peers cannot always select their m neighbors to make an exchange. In practice, however, most structured systems allow for some flexility in the selection of neighbors. We use the average number of neighbors that each pair of nodes can exchange to show this kind of flexibility. Table 2

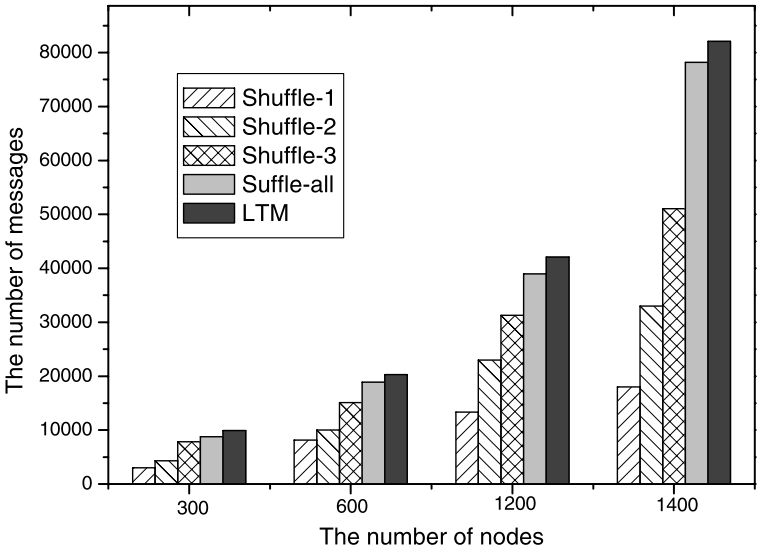


Fig. 15 Overhead comparison between PROP-O, PROP-G and LTM

Table 2 The flexibility results for PNS-Chord

System size	No PNS	PNS(4)	PNS(8)	PNS all
300	0.3115	1.1145	1.3170	4.2759
600	0.2001	1.0367	1.1949	4.7972
1200	0.1238	1.0333	1.1398	5.2981
2400	0.7480	1.0122	1.1377	5.7988

illustrates the flexility of PNS-Chord. "PNS All" represents an ideal state where any exchange can be performed. It is obvious that the more candidates exist, the greater the flexibility. We can also see that nodes in a sparse network with fewer nodes have more freedom to choose neighbors because a small inexactness will not matter when most neighbors of nodes have similar identifications (e.g., 1111 and 1110), mapped to the same physical machines. In short, PROP-O cannot be directly applied in structured systems, and its effectiveness relies heavily on the degree of flexibility allowed in the specific system.

5.4 Performance in a dynamic environment

Dynamism is a very important property in P2P systems [28]. In this section, we will examine the impact of dynamic changes in the system on the peer-exchange mechanism. We use the trace in [29] which recorded the lifespans of more than 500,000 peers over 7 consecutive days. Figures 16 and 17 show the effectiveness of PROP-G and PROP-O in dynamic environment respectively. The ascending lines in both figures present the Lost Cumulative Distribution Function (LDCF) of the nodes, which

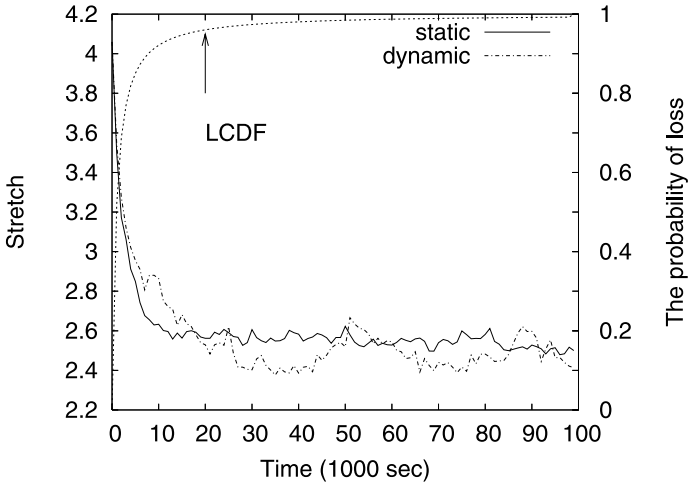


Fig. 16 Effectiveness in dynamic environment, PROP-G

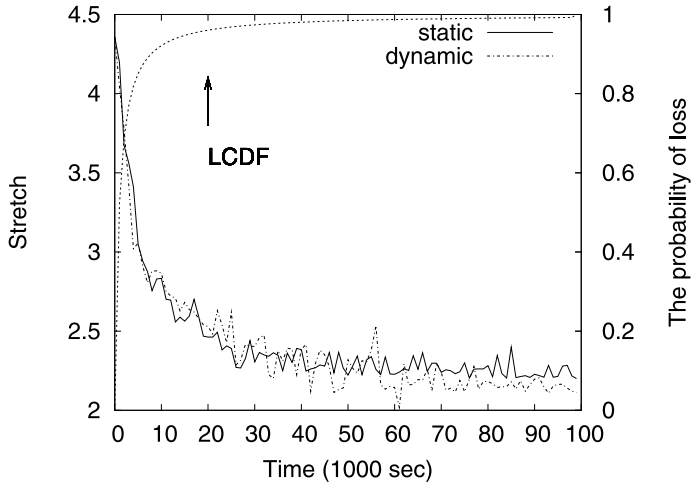


Fig. 17 Effectiveness in dynamic environment, PROG-O ($m = 4$)

show that most nodes stay in the system for only a short period of time. Generally speaking, peer-exchange is as effective in dynamic environment as in static one, and sometimes can achieve even better results because not all join/leave are harmful to the overlay. Moreover, we found that PROP-O can achieve better performance due to its flexibility to choose neighbors.

Figures 18 and 19 show what happens when the network experiences a sudden surge in the change of membership. We start with a nonoptimized network, and the members of the network stay roughly the same except for two bursts of activity in the intervals [20:40] and [60:80]. During these intervals, 100 nodes join and another

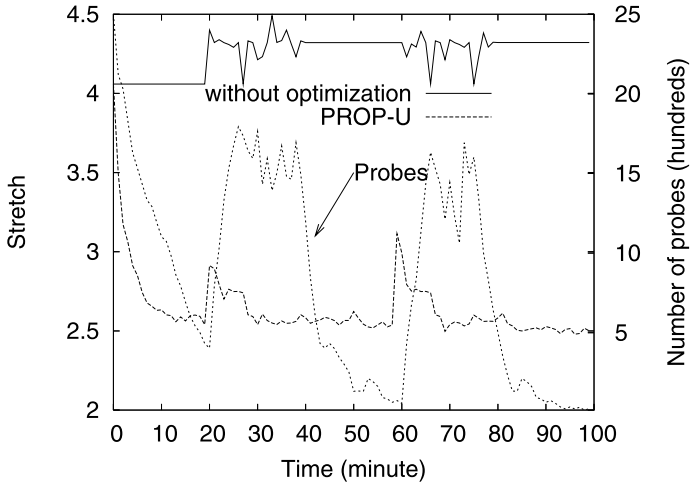


Fig. 18 Effectiveness of PROP-G in a bursty network environment

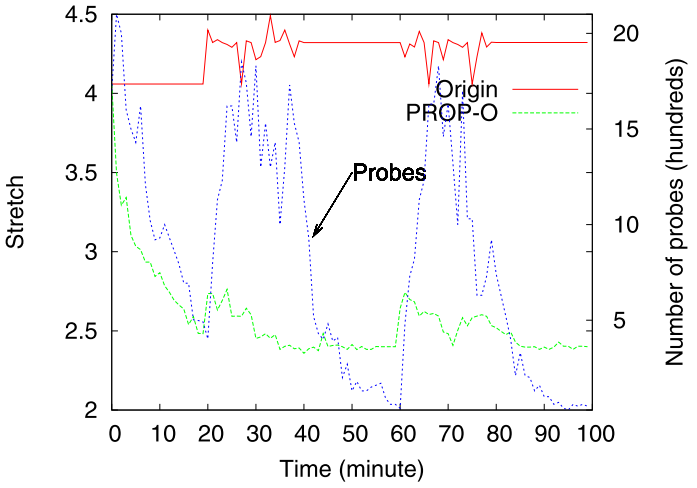


Fig. 19 Effectiveness of PROP-O in a bursty network environment

100 nodes leave each minute. The two figures show network quality under these conditions: without optimization, and when PROP-G (PROP-O) used. It further records the number of probes initiated when PROP-G (PROP-O) is applied. We can see that the number of probes increases dramatically when bursts occur, which illustrates the short response time of overlay reconstruction. On the other hand, stretch tends to stay at a low level for PROP-G (PROP-O) after a short period. After each bursty interval, the number of probes decreases accordingly. In other words, the peer-exchange scheme can adapt effectively to changes in the overlay without significant overhead.

6 Conclusion

This paper proposes a family of peer-exchange methods called PROP to solve the mismatching problem in P2P systems. PROP is adaptive scheme which can be easily embedded into most P2P systems without affecting the characteristics of the original systems. Simulation experiments show that PROP is an efficient way to match the physical network. By combining it with other recent methods, the overall performance can be further improved. It is also adaptive to dynamic change of peers. Unlike most previous studies that try to discover a better structure for P2P overlay, we accept the fact that many different structures coexist and our approach is another choice to make P2P systems more efficient.

A number of issues regarding PROP are still under study. For example, instead of using purely random contact which fixes *nhop*, we could find a better candidate node on the path of probing. Moreover, we merely focus on distance consideration. Actually, mismatch also includes other factors such as processing speed, storage capacity, and so on. Moreover, it will be very interesting to test out our methods in a live system, something that we are currently not equipped to do. We leave these issues for future work.

Acknowledgements The work is partly supported by China NSF grants (60573131, 60673154, 60721002), Jiangsu High-Tech Research Project of China (BG2007039), and China 973 project (2006CB303000) and a grant from CityU Project No. 7002115.

References

1. What is Gnutella. http://rfc-gnutella.sourceforge.net
2. Kazaa v3.2.5. http://www.kazaa.com
3. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for Internet applications. In: SIGCOMM '01: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications. ACM Press, New York, pp 149–160
4. Rowstron A, Drusche P (2001) Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware 2001: IFIP/ACM international conference on distributed systems platforms, vol 2218/2001. Springer, Berlin, pp 329–350
5. Zhao BY, Huang L, Stribling J, Rhea SC, Anthony, Joseph D, Kubiatowicz JD (2004) Tapestry: a resilient global-scale overlay for service deployment. IEEE J Sel Areas Commun 22:41–53
6. Castro M, Costa M, Rowstron A (2004) Should we build Gnutella on a structured overlay? SIGCOMM Comput Commun Rev 34(1):131–136
7. Liu Y, Xiao L, Liu X, Ni LM, Zhang X (2005) Location awareness in unstructured peer-to-peer systems. IEEE Trans Parallel Distributed Syst 16(2):163–174
8. Ripeanu M, Iamnitchi A, Foster I (2002) Mapping the Gnutella network. IEEE Internet Comput 6(1):50–57
9. Liu Y, Xiao L, Ni L (2004) Building a scalable bipartite P2P overlay network. In: IPDPS '04: 18th international parallel and distributed processing symposium. IEEE Computer Society, pp 46–55
10. Liu Y, Zhuang Z, Xiao L, Ni L (2004) Distributed approach to solving overlay mismatching problem. In: ICDCS '04: 24th international conference on distributed computing systems. IEEE Computer Society, pp 132–139
11. Gummadi K, Gummadiy R, Gribblez S, Ratnasamy S, Shenker S, Stoica I (2003) The impact of DHT routing geometry on resilience and proximity. In: SIGCOMM '03: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications. ACM Press, New York, pp 381–394

12. Ratnasamy S, Stoica I, Shenker S (2002) Routing algorithms for DHTs: some open questions. In: IPTPS '01: Revised papers from the first international workshop on peer-to-peer systems. Springer, London, pp 45–52
13. Ratnasamy S, Handley M, Karp R, Shenker S (2002) Topologically-aware overlay construction and server selection. In: INFOCOM 2002: Twenty-first annual joint conference of the IEEE computer and communications societies, vol 3, pp 1190–1199
14. Waldvogel M, Rinaldi R (2003) Efficient topology-aware overlay network. SIGCOMM Comput Commun Rev 33(1):101–106
15. Han J, Watson D, Jahanian F (2005) Topology aware overlay networks. In: INFOCOM 2005: 24th annual joint conference of the IEEE computer and communications societies, vol 4, pp 2554–2565
16. Fan J, Ammar MH (2006) Dynamic topology configuration in service overlay networks: a study of reconfiguration policies. In: INFOCOM 2006
17. Zhao BY, Duan Y, Huang L, Joseph AD, John, Kubiatowicz D (2002) Brocade: Landmark routing on overlay networks. In: Peer-to-peer systems: first international-workshop, IPTPS 2002, vol 2429. Springer, Berlin/Heidelberg, pp 34–44
18. Xu Z, Mahalingam M, Karlsson M (2003) Turning heterogeneity into an advantage in overlay routing. In: INFOCOM 2003: Twenty-second annual joint conference of the IEEE computer and communications societies, vol 2, pp 1499–1509
19. Li J, Stribling J, Morris R, Kaashoek MF (2005) Bandwidth-efficient management of DHT routing tables. In: NSDI 05: 2nd symposium on networked systems design and implementation. USENIX, pp 99–114
20. Hu J, Li M, Zheng W, Wang D, Ning N, Dong H (2004) SmartBoa: Constructing P2P overlay network in the heterogeneous Internet using irregular routing tables. In: IPTPS 2004: International workshop on peer-to-peer systems, vol 3279. Springer, Berlin, pp 278–287
21. Chawathe Y, Ratnasamy S, Breslau L (2003) Making Gnutella-like P2P systems scalable. In: SIGCOMM '03: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications. ACM Press, New York, pp 407–418
22. Ren S, Guo L, Jiang S, Zhang X (2004) SAT-Match: a self-adaptive topology matching method to achieve low lookup latency in structured P2P overlay networks. In: IPDPS '04: 18th international parallel and distributed processing symposium. IEEE Computer Society, pp 83–91
23. Rowstron A, Druschel P (2001) Storage management and caching in past, a largescale, persistent peer-to-peer storage utility. In: SOSP '01: Proceedings of the eighteenth ACM symposium on operating systems principles. ACM Press, New York, pp 188–201
24. Dabek F, Kaashoek M, Karger D, Morris R, Stoica I (2001) Wide-area cooperative storage with CFS. In: SOSP '01: Proceedings of the eighteenth ACM symposium on operating systems principles. ACM Press, New York, pp 202–215
25. Zegura EW, Calvert KL, Bhattacharjee S (1996) How to model an internetwork. In: INFOCOM'96: Fifteenth annual joint conference of the IEEE computer societies, vol 2, pp 594–602
26. Dabek F, Li J, Sit E, Robertson J, Kaashoek MF, Morris R (2004) Designing a DHT for low latency and high throughput. In: USENIX symposium on networked systems design and implementation
27. Chun S-G, Zhao BY, Kubiatowicz JD (2005) Impact of neighbor selection on performance and resilience of structured P2P networks. In: IPTPS 2005: 4th international workshop of P2P systems. Springer, Berlin, pp 264–274
28. Rhea S, Geels D, Roscoe T, Kubiatowicz J (2004) Handling churn in a DHT. In: USENIX annual technical conference
29. Bustamante F, Qiao Y (2003) Friendships that last: peer lifespan and its role in P2P protocols. In: Web content caching and distribution: proceedings of the 8th international workshop. Kluwer Academic, Norwell, pp 233–246
30. Qiu T, Wu F, Chen G (2005) A generic approach to make structured peer-to-peer systems topology-aware. In: Proceedings of ISPA, pp 816–826
31. Qiu T, Chen G, Ye M, Chan E (2007) Towards location-aware topology in both unstructured and structured P2P systems. In: Proceedings of ICPP

Tongqing Qiu received his B.Sc. and M.Sc. degrees in computer science from Nanjing University, China. He served as a research assistant in the Department of Computer Science at City University of Hong Kong from April 2006 to February 2007. He is now a Ph.D. student in the Department of Computer Science at Georgia Tech, USA. His research interests are in the areas of network measurement, peer-to-peer computing and distributed systems.

Edward Chan received his B.Sc. and M.Sc. degrees in Electrical Engineering from Stanford University, and his Ph.D. in Computer Science from Sunderland University. He worked in the Silicon Valley for a number of years in the design and implementation of computer networks and real-time control systems before joining City University of Hong Kong where he is now an Associate Professor. His current research interests include performance evaluation of high speed networks, mobile data management, power-aware computing, and network management.

Mao Ye received his B.Sc. degrees in Department of Computer Science of Nanjing University, in 2004. Since then, he was enrolled in the Ph.D. program in Department of Computer Science of Nanjing University. His research interests include distributed system and pervasive computing, particularly in the area of peer-to-peer network and wireless sensor network.

Guihai Chen obtained his B.Sc. degree from Nanjing University, M. Engineering from Southeast University, and Ph.D. from University of Hong Kong. He visited Kyushu Institute of Technology, Japan in 1998 as a research fellow, and University of Queensland, Australia in 2000 as a visiting professor. During September 2001 to August 2003, he was a visiting professor in Wayne State University. He is now a full professor and deputy chair of the Department of Computer Science, Nanjing University. Prof. Chen has published more than 120 papers in peer-reviewed journals and refereed conference proceedings in the areas of wireless sensor networks, high-performance computer architecture, peer-to-peer computing and performance evaluation. He has also served on technical program committees of numerous international conferences. He is a member of the IEEE Computer Society.

Ben Y. Zhao is an Assistant Professor in the Computer Science department at UC Santa Barbara. Professor Zhao completed his M.Sc. and Ph.D. degrees in Computer Science at UC Berkeley. His research spans the areas of large scale networks and distributed systems, security and privacy, and mobile and wireless networks. He received his B.Sc. degree from Yale University (BR'97). Additionally, Professor Zhao is a recent recipient of the National Science Foundation's CAREER award, MIT Tech Review's TR-35 Award (35 Young Innovators Under 35), and ComputerWorld's Top 40 Technology Innovators.