A Generic Approach to Make Structured Peer-to-Peer Systems Topology-Aware

Tongqing Qiu, Fan Wu, and Guihai Chen

State Key Laboratory of Novel Software Technology, Nanjing University, China
{qtq, wufan}@dislab.nju.edu.cn, gchen@nju.edu.cn

Abstract. With the help of distributed hash tables, the structured peer-to-peer system has a short routing path and good extensibility. However, the mismatch between the overlay and physical network is the barrier to build an effective peer-to-peer system in the large-scale environment. In this paper, we propose a generic approach to solve this problem, which is quite different from other protocol-dependent methods. We reserve the structure of system and break the coupling between the node and its identifier by swap operations. We also propose several policies to reduce the traffic overhead. The policies include adaptive probing and shadow scheme. The experiment shows that our approach can greatly reduce the average latency of overlay networks and the overhead is controllable.

Keywords: peer-to-peer, overlay network, topology-aware, stretch.

1 Introduction

Several recent peer-to-peer (P2P) systems (CAN [1], Chord [2], Pastry [3], etc.) provide a self-organizing substrate for large-scale P2P applications. These structured P2P systems can be viewed as providing a scalable, fault-tolerant distributed hash table (DHT). Any item (content) can be located with in a bounded number of hops, using a small per-node routing table. However, as a node is hashed to a random identifier (node ID), the mismatch between physical topologies and logical overlays is a major factor that delays the lookup response time. In this situation, "hop" is no longer a reasonable metric to measure the delay. We usually call it *mismatching* or *topology-aware* problem. There are several methods to solve the problem. Most of methods solve it in two basic steps [4]: 1) to gather some information about network proximity, and 2) to construct or repair the overlay network using information above. In order to show the limitations of recent work, we will discuss these two steps in the following two subsections.

1.1 Collect Proximity Information

To solve the mismatching problem, some sort of proximity information of the underlying network is needed. There are two general ways which have been proposed – *landmark clustering* and *flooding or heuristic-based search.* Landmark clustering is based on the intuition that nodes close to each other are likely to

Y. Pan et al. (Eds.): ISPA 2005, LNCS 3758, pp. 816–826, 2005.

have similar distances to a few landmark nodes. S. Ratnasamy *et al* [5] utilize this idea to optimize CAN system. The main limitation of this solution is that it is a coarse method to discover the proximity of different nodes. Besides, the landmarks are like servers in the system, introducing some *single-point failure* problems [6]. Flooding or heuristic-based search is another choice to get proximity information. It is like searching method in a P2P system. Instead of getting contents, it tries to get delay information. In this way, we can gather more detailed knowledge about the physical network than landmark method. However, uncontrollable searching will be too expensive for topology matching. So the challenge is to make tradeoff between effectiveness and probing cost.

1.2 Utilize Proximity Information

When we have got some knowledge about the proximity, the next step is to utilize the proximity information to construct or repair the structured peer-to-peer system. Three basic approaches have been suggested for exploiting proximity in DHT protocols [7] – *proximity routing*, *proximity neighbor selection* and *geographic layout*. There are several systems which use one of these three policies. Topologically-Aware CAN [5] is an example with geographic layout. This approach unfortunately creates uneven distribution of nodes on the overlay. Pastry uses proximity neighbor selection to construct the routing table [8]. However, the ID prefix of Pastry is a constraint to limit the selection range. As a matter of fact, all of these have a common limitation – *protocol-dependent*. For instance, geographic layout ensures that nodes that are close in the network topology are close in the node ID space, which is only suitable for the system like CAN [9]. Because in CAN, the nearness in node ID means less hops in routing. In systems like Pastry or Tapestry, we have some degree of freedom to choose nodes in the routing table. But in Chord or CAN, the entries in routing table are deterministic. Proximity routing also has the requirement that there must be more than one choice for next hop, which is not suitable for systems like Chord.

The further problem is the dynamism in peer-to-peer systems. As nodes arrive or depart, the existing routing tables need to be repaired. Without timely repairing, the structure of overlay will digress from optimal condition as inefficient routes gradually accumulate in routing tables. So an effective overlay should be adaptive to the system's dynamic change.

In order to solve all problems mentioned above, we propose a novel method to make the structured P2P system topology-aware. This method periodically adjusts the node ID and preserves the structure of P2P systems. By iteratively reducing the average logical link latency, the overlay trends to match the physical network. This method is protocol-independent and easy to be built on any structured P2P systems. Through our experiment based on Chord, we find that this approach can greatly reduce average logical/physical link latency. Besides, the overhead of adjustment is very low when using adaptive policies.

This paper is organized as follows. In section 2 we describe our approach in detail, including basic policy and several overhead-reducing mechanisms. In section 3, we illustrate the results of our experiment and give some explanations.

Other related work is introduced in section 4. Finally, in section 5 we conclude the paper and point out some future work.

2 Variable Node ID

2.1 Basic Method

We propose a novel solution to the mismatching problem of distributed hash table, which is based on *variable node ID.* As we know, in a DHT system, one node is hashed into a unique identifier which is called node ID. Usually, the node ID will not change during the node's lifetime. The advantage of this scheme is obvious. It is very easy to manage a large-area system using these identifiers. Besides, the hashing process is totally random. In other words, each peer in the system is anonymous. The disadvantage of invariable node ID is also apparent. There are many constraints of routing and some constraints are unreasonable. "Mismatching problem" is an example. Figure 1 gives a mismatching situation. If node A wants to route a message to B in structure (b), the cost is 12 ($A \rightarrow C \rightarrow B$) or 14 ($A \rightarrow D \rightarrow B$), both larger than routing in (a). The essential cause of mismatching is that each node is always combined with an identifier. When one node joins into the system, its position is unchangeable. We consider whether it is possible to make the node ID more flexible, without weakening the power of DHT scheme. There are several *guidelines* we should follow when making some kind of node ID varying . First, the change of node ID should not change the structure of a P2P system. As we mentioned in section 1, the common limitation of most recent methods is that they rely on the specific protocols. If our change breaks the original structure, we can not reconstruct it without specific protocol information. In other words, it will also be protocol-dependent. Second, this change should not be arbitrary. As we know, one of the basic characteristics of the P2P system is its anonymity. If the node ID can be changed discretionarily, the system will become fragile and easily attacked by hackers. Last, the overhead of changing should be controllable. If the overhead is too expensive, the method can not achieve good performance.

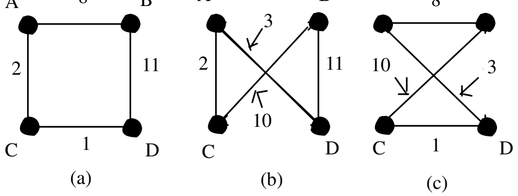

Fig. 1. A mismatching example. (a) is the physical topology with four nodes, and the latencies are marked with integer numbers. (b), (c) are both overlay structures on that physical topology. We assume that neighbors' latencies in overlay are the shortest paths between them. For example, the latency of $A \rightarrow D$ in (b) is calculate as the path $A \rightarrow C \rightarrow D$ in (a). So it equals to 3.

Table 1. The notation table

Notation	Meaning
t_0	the time before nodes a and b swap
t_1	the time after nodes a and b swap
$N_{t_i}(a)$	the neighbor set of node a at time t_i
$d(ij)$	latency between nodes i and j
L_{t_i}	the accumulated latency value of overlay at t_i

We explain our method as follows. In a structured P2P system, each item is hashed into a unique identifier. All of these identifiers constitute a "ID space" At the same time, each node also has one identifier. We call the set of these identifiers "node ID space". It is a subset of ID space. Regardless of peers' dynamism, node ID space is relatively invariable. In our method, each node can not arbitrarily choose an identifier in id space. However, each one has the freedom to choose a *better* identifier in the node ID space. In this way, the logical structure which is built on node ID space will not be broken (following guideline one). In addition, as the identifiers in node ID space is totally random, the anonymity will be preserved (following guideline two). To achieve this kind of node ID re-assignment, the basic operation is *swap*: swap the node ID, and exchange the corresponding routing tables. For example, if we want to adjust the identifiers in figure 1(b) or 1(c), we will just swap node B's id and D's or swap C's and D's correspondingly. After the adjustment, the overlay will totally match the physical network.

Figure 1 just illustrates a simple and ideal case. In a real P2P system, things are more difficult. Table 1 gives several useful notations for our expression. We assume there is a swapping try between nodes a and b. Node a is the *counterpart* of b, and vice versa. Two different situations t_0 and t_1 represent the time before and after a swap . In fact, t_1 is not actually the time after the swap, but the *hypothetical* time if we make the swap. In addition, $N_{t_i}(a)$ represents the neighbor set of node a at time t_i. It is worth to emphasize the fact that neighbors of one node N are not just the entries in its routing table. The nodes which point to node N should be also included. At the beginning, nodes a and b will exchange their neighbors' addresses. Then both of them probe the counterpart's neighbors and measure these latencies $d(ij)$. Node a calculates the accumulated latency of its current neighbors $\sum_{i\in N_{t_0}(a)} d(ai)$ and the one if the swap is done $\sum_{i\in N_{t_1}(a)} d(ai)$. The similar results are calculated by node b. The difference between before and after swap is shown in equation 1.

$$Diff = \sum_{i\in N_{t_0}(a)} d(ai) + \sum_{j\in N_{t_0}(b)} d(bj) - \sum_{i\in N_{t_1}(a)} d(ai) - \sum_{j\in N_{t_1}(b)} d(bj) \qquad (1)$$

If $Diff > 0$, nodes a and b will exchange their identifiers and routing tables. Unfortunately, in many structured P2P systems, it is not enough to change the state of these two nodes. The reason is that the routing in most systems is unidirectional[1]. As a result, the change of any node N will impact the nodes which have an entry $E = N$ in their routing tables. However, the unidirectional property will not complicate the implementation of our approach. Because the change of each node can be realized using *leave()* and *join()* procedures, which are already implemented in any P2P system. Until now, we just illustrate a single swap operation. In a distributed environment, every node will periodically contact a random node. The TTL-packet is used to realize this contact. At the beginning, we set $TTL = k$. When TTL becomes zero, the target node is located. Given that the method is totally distributed, each node tries to make a swap at a fixed interval. If a swap can improve the match degree, many swaps at the same time will achieve accumulated effect. In the next subsection, we will try to explain the effectiveness of node swap.

2.2 Effectiveness of Node Swap

To explain the effectiveness of node swap, we make several definitions and explain the meaning of notations first. We define *stretch* as the ratio of the average logical link latency over the average physical link latency. Stretch is a common parameter to quantify the topology match degree. *Average latency (AL)* is a basic parameter to quantify the property of a network. If there are n nodes in a network, and accumulated latency of any two nodes is $Acc(n)$, then[2]

$$AL = Acc(n)/n^2 \tag{2}$$

We analyze the change of average latency after a swap between nodes a and b. Supposing that the number of nodes is invariable during $t_0 \rightarrow t_1$, so the *accumulated latency* (L_{t_i}) is analyzed instead. Next two equations show this change:

$$L_{t_0} = C + \sum_{i \in N_{t_0}(a)} \alpha_i d(ai) + \sum_{j \in N_{t_0}(b)} \beta_j d(bj) \tag{3}$$

$$L_{t_1} = C + \sum_{i \in N_{t_1}(b)} \gamma_i d(bi) + \sum_{j \in N_{t_1}(a)} \delta_j d(aj) \tag{4}$$

In equation 3 and 4, C represents the invariable part before and after one swap operation. The coefficients of the summations $\alpha, \beta, \gamma, \delta$ represent the times each neighbor link used. We notice that nodes a and b just exchange their neighbors, so $N_{t_1}(b) = N_{t_0}(a)$ and $N_{t_0}(b) = N_{t_1}(a)$. Besides, assuming that each link has the same probability to be visited, then $\alpha_i \approx \gamma_i$ and $\beta_j \approx \delta_j$. To calculate the variation by (3) – (4), we get that if $Diff > 0$ then $L_{t_0} > L_{t_1}$, which implies that a swap makes the stretch reduced. It is worth to mention that it is an approximate analysis. In fact, when the positions of the nodes changed, the times each neighbor link visited are variable. In other words, those coefficients are different, that is why not all swaps can reduce the average latency. We will see that in our experiment.

[1] CAN is an exception. Its routing is bidirectional.

[2] We assume the latency between one node and itself is zero.

2.3 Controllable Overhead

In section 2.1, we have given three guidelines to change node ID. However, we have not given the method to control the overhead yet. The overhead of our approach includes four aspects: (1) the probing of neighbors, (2) the probing of random nodes, (3) exchanges of the routing tables, and (4) exchanges of the contents. We believe that the cost (1) is limited as it can be realized as a piggyback process when constructing the P2P system. So we just give solutions to reduce cost of (2-4).

Adaptive Probing. Cost (2) and (3) are relative to swap times. In our basic method, we do probing periodically at a fixed interval. However, as the system trends to be steady, this periodic adjustment becomes costly and not necessary. The ideal time to stop the periodic adjustment is when the system's average latency doesn't change obviously. Due to the limitation of distributed systems, we can only make decisions based on the local information. So we propose an *adaptive policy* to reduce the operations of probing and swapping. From a local view, every node lives in an *environment* consisting of its neighbors. If neighbors of one node change continually, this node lives in an *unstable* environment. So it will try to do probing and make swapping. Oppositely, if the node's neighbors do not change at a relatively long interval, we can believe that this node is *stable*. To realize this idea, a parameter *activity* is used as the description of node's state and the criterion of periodic probing. At the beginning, the activity parameter is set as an initial number. If one node makes a swap operation, it will move to a new environment, so this parameter will increase to make probing continue. Besides, it will also notify its neighbors to increase activity number. As the fixed intervals pass, the activity number will be reduced. Algorithm 1 is the pseudo code of adaptive probing. Tow parameters – initial number and threshold both have an effect on the number of probing operations. Appropriate value of the two parameters will make the system achieve a better performance. In our experiment, both of them are zero. The results show that this adaptive method greatly reduce the number of the nodes' probing and swap operations without sacrificing the effectiveness of stretch reduction too much.

Shadow Scheme. In a real P2P system, all contents reside on different nodes. In other words, each node owns one part of id space. After exchanging the identifiers of nodes a and b, the contents that they owns should be exchanged respectively. This process may be most expensive one among four aspects mentioned above. Inspired by Baumann et al's work in mobile agent area [10], we propose a *shadow scheme* to reduce the overhead. We view the nodes a and b as mobile agents. After they swap their identifiers, they will not exchange the contents immediately. Instead, both of them own their counterpart's *shadow*, which records the specific lifetime of the shadow and the address information of their counterpart. So before the lifetime becomes zero, the content queries will be forwarded by the counterpart to the correct destination. When the nodes become *stable* and the lifetime is over, the contents will be exchanged. The value

Algorithm 1. Adaptive probing of each node

```
$activity = initial\ number$
while $activity \geq threshold$ do
  probe one node
  if swap is necessary then
    exchange the node ID and routing tables
    $activity = activity + 1$
    notify neighbors to increase activity
  end if
  $activity = activity - 1$
  wait for an fixed interval
end while
```

of the lifetime is related to the state of a node which we describe above. Utilizing shadow scheme, the content distribution times are reduced. Obviously, it will take a longer path to locate the content. So it's necessary to consider the tradeoff between the query latency and the distribution overhead in a real P2P application. As the content distribution is relative to specific applications, we propose a generic scheme here and will not consider it in our experiments.

3 Performance Evaluation

3.1 Simulation Methodology

We use the GT-ITM topology generator [11] to generate transit-stub models of the physical network. In fact, we generate two different kinds of topologies. 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 sub domain. 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 in which 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 100ms to stub-stub, stub-transit and transit-transit links respectively. Then, several nodes are selected from the topology as overlay nodes, with the node number $n = \{300, 600, 1200\}$. Chord is chosen as the platform of our simulation because the limitation of Chord makes it unsuitable for many mismatching solutions. We have discussed the limitation in section 1.

3.2 Effectiveness of Swap

The *stretch* is used to characterize the match degree of the overlay to the physical topology. The time interval is fixed as one minute. Figure 2 shows the impact of the TTL scale on stretch. We choose node number $n = 600$ and four typical scenes of probing node. In a centric scene, we can just choose a random node as

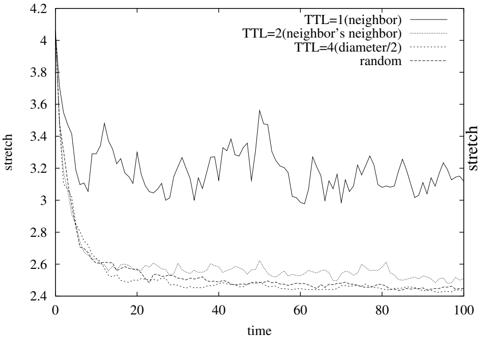

Fig. 2. Varying the TTL scale

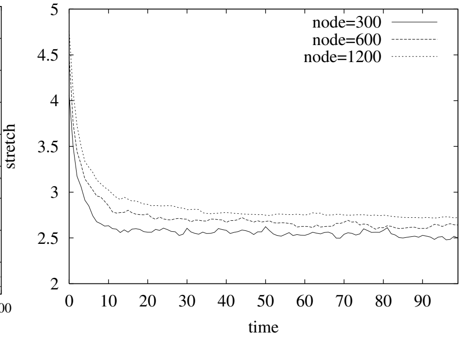

Fig. 3. Varying the system size

the probing target. In a distributed system, we use $TTL = \{1, 2, 4\}$. $TTL = 1$ means probing neighbors; $TTL = 2$ means probing neighbors' neighbors and $TTL = 4$ means probing the node half of diameter away from the original node[3]. We can find that neighbors' swap is not suitable as it can't greatly reduce the stretch, while other three different ways have nearly the same impact on stretch reduction. The reason is obvious, as $TTL = 1$ gets only neighbor information which is too limited. Given that random probing is not practical in a distributed system, only when $TTL \geq 2$ can achieve a good performance in a P2P system. In order to minimize cost, $TTL = 2$ may be a better choice, and it will be used in next several experiments. In figure 2, we can also discover that the stretch is not reduced all the time, which is consistent with our approximate analysis in section 2.2.

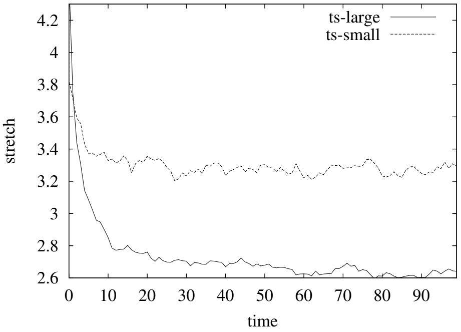

Fig. 4. Varying the physical topology

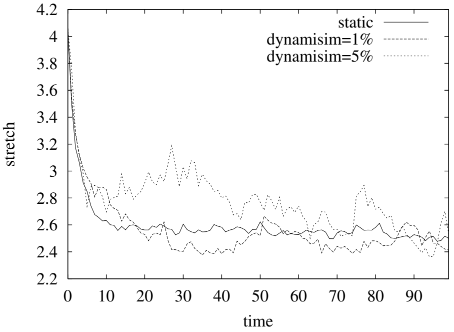

Fig. 5. The stretch in dynamical environment

Figure 3 illustrates the impact of system size. We choose $n = 300, 600, 1200$. The effectiveness is reduced as the size becomes larger. This situation can be

[3] As node number is 600, we suppose that the diameter $d = \log_2 n \approx 8$.

explained in two different directions. First, when the system has a large size and probing method fixes TTL as 2, the information we get is relatively limited. Second, as we choose the nodes from the same physical network, when the overlay becomes larger, it is closer to the physical topology. And the effectiveness will be not so obvious.

The impact of physical topology is presented in figure 4. We have generated two different types of topologies *ts-large* and *ts-small* by GT-ITM tools. Both of them contain 2200 nodes. It is obvious that ts-large topology has much better performance. In ts-large topology, only a few stub nodes attach to transit nodes. So the probability that two stub nodes belong to different transit nodes is relatively high. Accordingly, the probability that these nodes exchange is also great. It means that two *far* nodes make adjustment to match the physical topology with a high probability. This kind of swap will greatly improve the performance of the system. As we mentioned above, ts-large topology is much like the Internet, so our method will significantly improve performance in a real large-scale system.

3.3 Dynamic Environment

Dynamism is a very important property in P2P systems. In this part, we try to discover the impact of dynamism on our approach. Although people do several searches about dynamism of the P2P system [12], there is not a standard model to describe it. In our simulation, we just set a very simple dynamic environment. There are δ percent of nodes join and δ percent of nodes leave at a time interval t. $\delta = \{0, 1, 5\}$ and $t = 1min$. Figure 5 shows the results. It is obvious that stretch fluctuates greatly when the system is under a dynamic situation in which 5 percent of nodes change per minute. However, we can see that nodes' arrival and departure may not lead the system to a poor match degree. It's possible that nodes' changes have the similar effect as our swap operation which reduces the stretch of the system. Although there is a fluctuation, our method can be still effective in dynamic environment.

3.4 Adaptive Probing

Regardless of the distribution of the content, the largest overhead is related to times of swapping. In section 2, we introduce an adaptive method to reduce swapping times. Figure 6 illustrates the effect of this method. We compare two different policies. The first one is to probe at a fixed interval, while the second one is to probe with an additional parameter – *activity*. Initial number and threshold are both zero. In this figure, x axis represents stretch and y axis represents the swap times. One point records the swap times in one minute and the stretch value after these swap operations. The adaptive method significantly reduces the swap times. At the same time, it sacrifices the effectiveness of stretch reduction. The points at the high-stretch interval of fixed method are less than adaptive one. However, it is not as significant as the reduction of swap operation. So we choose the adaptive method to reduce the overhead.

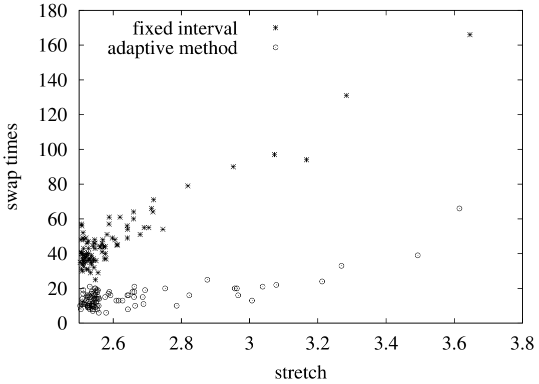

Fig. 6. Tradeoff between swap and stretch

4 Related Work

There are several methods that try to solve the mismatching problem. The most related one to our work is a method called "SAT-Match" [13]. The basic operation in this system is *jump*. When one node discovers several nearby nodes by flooding, it will jump to the nearest node in the nearby area. In fact, it is one kind of variable node ID. However, this method has several limitations. First, the node ID space changes after jumping. So the original overlay structure is broken. Arbitrary change of ID also violates the anonymity of the P2P system. One node which is controlled by a hacker can easily jump to a specific area. Second, although author mentioned the impact of the dynamism, we can not find detail evaluation in different dynamic environments. Last, with respect to the overhead, SAT-Match didn't give a solution to reduce the cost of the content movement.

5 Conclusion

This paper proposes a novel method to solve the mismatching problem in structured P2P systems. This method is totally protocol-independent, which can be easily used on any P2P system based on DHT. Besides, we propose a series of solutions to minimize the overhead cost, including adaptive probing and shadow scheme. Our experiment has shown that node swap greatly reduces the stretch of overlay networks, and the number of swap operations is also greatly reduced when using adaptive probing. In the near further, we will try to combine our method with other different solutions like *proximity neighbor selection* (PNS) together to achieve better performance.

Acknowledgement. This work is supported by the China NSF grant, the China Jiangsu Provincial NSF grant (BK2005208), the China 973 project (2002CB312002) and TRAPOYT award of China Ministry of Education.

References

1. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the ACM SIGCOMM. (2001)
2. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the ACM SIGCOMM. (2001)
3. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001)
4. Xu, Z., Tang, C., Zhang., Z.: Building topology-aware overlays using global soft-state. In: Proceedings of ICDCS 2003. (2003)
5. Ratnasamy, S., Handley, M., Karp, R., Shenker., S.: Topologically-aware overlay construction and server selection. In: Proceedings of INFOCOM 2002. (2002)
6. Winter, R., Zahn, T., Schiller, J.: Random landmarking in mobile, topology-aware peer-to-peer networks. In: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS04). (2004)
7. Ratnasamy, S., Shenker, S., Stoica., I.: Routing algorithms for DHTs: Some open questions. In: 1st International workshop on P2P Systems(IPTPS02). (2002)
8. Castro, M., Druschel, P., Hu, Y., Rowstron, A.: Exploiting network proximity in distributed hash tables. In: Proceedings of FuDiCo 2002. (2002)
9. Waldvogel, M., Rinaldi., R.: Efficient topology-aware overlay network. In: Proceedings of HotNets-I. (2002)
10. Baumann, J., Hohl, F., Rothermel, K., StraBer, M.: Mole – concepts of a mobile agent system. In: Proceedings of World Wide Web. (1996)
11. Zegura, E.W., Calvert, K.L., Bhattacharjee., S.: How to model an internetwork. In: Proceedings of INFOCOM. (1996)
12. Ge, Z., Figueiredo, D.R., Jaiswal, S., Kurose, J., Towsley, D.: Modeling peer-to-peer file sharing systems. In: Proceedings of IEEE INFOCOM. (2003)
13. Ren, S., Guo, L., Jiang, S., Zhang, X.: SAT-Match: A self-adaptive topology matching method to achieve low lookup latency in structured P2P overlay networks. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS04). (2004)