Node Clustering Based on Link Delay in P2P Networks

Wei Zheng[*] , Sheng Zhang, Yi Ouyang, Fillia Makedon, James Ford
Department of Computer Science
Dartmouth College
6211 Sudikoff Laboratory
Hanover, NH 03755-3510, USA
{Wei.Zheng, Sheng.Zhang, Yi.Ouyang, Fillia.S.Makedon, James.C.Ford}@Dartmouth.EDU

ABSTRACT

Peer-to-peer (P2P) has become an important computing model because of its adaptation, self-organization and autonomy etc. But efficient organization of the nodes in P2P networks is still a challenge needs to be addressed. Node clustering is a mechanism that aims to provide an optimal infrastructure to organize the nodes in a P2P network. This paper describes an approach to implement node clustering based on link delay of node communications in the P2P network. This approach is completely distributed, in which each node only depends on its neighbors to implement node clustering. In this approach, we propose two distributed algorithms: T-closure algorithm and hierarchical node clustering algorithm to find node clusters automatically in a P2P network. We explore the node connectivity together with the connection quality. As a result, the link delay of communication between the super-node and the peer-node in node clustering can be limited, which will improve the overall performance of P2P networks.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms

Keywords

Peer-to-Peer, node clustering, link delay, hierarchical clustering

1. INTRODUCTION

In a peer-to-peer(P2P) network, two or more peers use appropriate information and communication systems to collaborate spontaneously without requiring central coordination [1][2]. In addition to the benefits as adaptation, self-organization, peer autonomy, load balancing, fault-tolerance through massive replication [3], P2P networks also introduce certain challenges that limit their usages. One challenge is the organization of peer nodes in P2P networks. The organization adopted by most existing systems are costly in terms of communication messages, which will result in message flooding[4][5][6]. Currently researchers propose to construct a super-peer network to improve the P2P network infrastructure using node clustering [7]. In a super-peer network, each node cluster is a set of nodes in the P2P network, in which the super-node handles communications of its peer-nodes and manages the node cluster. By this way, the management of the whole P2P network is distributed over the super-nodes and the overall performance will be improved[7][8][9]. For example, with node clusters, we can decrease the communication messages and limit the length of routing paths in P2P networks[10].

Although node clustering has been utilized to improve system performance such as KaZaA [11], discovering of node clusters in P2P networks is still a challenge that needs to be addressed [8][12]. In previous works, Connectivity-based Distributed Node Clustering(CDC) [10] implements node clustering based on the node connectivity in P2P networks. Through random routing of weighted messages, CDC tries to create node clusters based on the highly connected node set. In CDC, inappropriate choice of initiators may cause bad clustering result. Max-Min D-Cluster Formation [13] proposes to find node clustering based on d-hop dominating set, which is similar to k-hop clustering algorithm [14]. As proved in [13], the minimum d-hop dominating set problem is NP-complete, Max-Min D-Cluster algorithm is used as an approximation of the optimal solution for the minimum d-hop dominating set problem. In [15], MCL algorithm assumes that the global information about the entire P2P network such as the number of nodes, the number of node connections etc, is available in a central location. In [16], node clustering is constructed according to the complete connected set graph in the P2P network.

In previous works, node clustering is mainly focused on node connectivity, sometimes requiring the global knowledge such as MCL. In this paper, we propose a new node clustering approach, node clustering based on link delay, for P2P networks. In addition to node connectivity, our approach also studies the quality of node connection, which is not addressed in previous works. In a super-peer network, peer-nodes may communicate very often with super-nodes,

[*]Supported by NSF grant IDM-0308229

consequently the quality of node connection in a node cluster will affect the performance of the whole P2P network. In our approach, the link delay of communications between peer-nodes and super-nodes can be restricted to a time limit T. Our approach is implemented in a completely distributed way without the requirement of global knowledge. Another issue not addressed in previous works is the number of node clusters. If the number is too large, the performance of super-peer networks is nearly the same as that of P2P networks without node clusters. In our approach, the number of node clusters can also be restricted. It is a constant approximation of the optimal solution with high probability, which guarantees the overall performance of P2P networks.

The rest of the paper is organized as: section two describes the problem of node clustering based on link delay and the terminologies; section three describes and analyzes two algorithms used in our approach: T-closure algorithm and hierarchical node clustering algorithm; section four describes experiments to show the effectiveness of T-closure computation and tolerance of our approach to changes in P2P networks.

2. PROBLEM DESCRIPTION AND TERMINOLOGIES

With the link delay of communications among different nodes, a P2P network can be represented by a connected and weighted graph. Here we define the graph as $G = (V, E, W)$. Any vertex $v \in V$ represents an actual node in a P2P network; for two vertices u and v if the corresponding nodes in the P2P network know the existence of each other(i.e., logically connected), then there is an edge $(u, v) \in E$ between them and the edge weight $W(u, v)$ is the link delay of communication between u and v. In the graph G, the sum weight of all the edges in a path is called the path distance. $\{V_1, V_2, V_3, \ldots, V_k\}(V_i \subseteq V, 1 \le i \le k)$ is a *vertices covering* of G, which satisfies $V_1 \cup V_2 \cup V_3 \cup \ldots \cup V_k = V$ and $V_i \cap V_j = \emptyset(i \neq j)$.

In a P2P network represented by the graph G, the nodes set corresponding to $V_i(1 \le i \le k)$ is called a *node cluster*. In a node cluster, there is a node that will be in charge of all the other nodes, which is called *super-node*(i.e., *cluster head*). The other nodes in the node cluster are called *peer-node*. In order to guarantee the connection quality, a parameter $T(T > 0)$ is defined to be the limit of link delay for the communication between peer-node and super-node in a node cluster. The problem of node clustering based on link delay in a P2P network is: find a vertices covering $\{V_1, V_2, V_3, \ldots, V_k\}$ with the minimum value of k and in each $V_i(1 \le i \le k)$, the distance of path between $v_i, v'_i \in V_i(v_i$ is the vertex corresponding to the super-node of the node cluster represented by V_i, v'_i is the vertex corresponding to the peer-node) is not larger than the parameter T.

3. NODE CLUSTERING BASED ON LINK DELAY

In [13], it is proved that the d-hop dominating set problem in a graph is NP-complete. The d-hop dominating set problem can be easily reduced to the problem of node clustering with the link delay limit as d. So the problem of node clustering based on link delay is also NP-complete and the objective of our approach is to find an approximation of the optimal solution.

Node clustering based on link delay includes two parts: T-closure computation and hierarchical node clustering. Correspondingly the T-closure algorithm and the hierarchical node clustering algorithm are proposed. In fact, the result of T-closure computation is a preparation for hierarchical node clustering. Both of these two algorithms are completely distributed algorithms. In node clustering based on link delay, we assume that each node has a unique identification number arbitrarily assigned in the P2P network.

3.1 T-closure Computation

The objective of T-closure computation is: given the connected and weighted graph $G = (V, E, W)$ for a P2P network and the limit T. For two vertices $u, v \in V$ in the graph G, if the distance of the shortest path between u and v is not larger than T, then we make the nodes represented by u, v in the P2P network know the existence of each other and set the link delay of communication between them as the distance of the shortest path in G. After T-closure computation, we construct a new graph $G' = (V', E', W')$. In G', we have: $V' = V$, $E \subseteq E'$ and for any two vertices u and v, if the distance of the shortest path between u, v is not larger than T in G, there must be an edge $(u, v) \in E'$ and $W'(u, v)$ is the distance of the shortest path in G.

3.1.1 T-closure Algorithm

For T-closure computation, we propose the T-closure algorithm. The intuition of T-closure algorithm is based on one observation of the shortest path: assume we create a list of the shortest paths originating from the vertex u with their distances in the ascending order, then any one of these shortest paths must be an extension of some previous one in the list. So for each vertex u in the T-closure algorithm, we search for the shortest paths beginning from u with their distances in the ascending order. And in the T-closure algorithm, each vertex u has variables as:

SL: a sorted list stores the information of the shortest paths beginning from u with their distances in the ascending order and not larger than T. For a shortest path $(u, v_1, v_2, \ldots, v_n, v)$, there is an entry $(dv, dt, pass\_ver)$ in SL. dv is the destination node v of the shortest path; $dt(dt \le T)$ is the distance of the shortest path; $pass\_ver$ is the nearest vertex v_1 to u in the path.

EL: a sorted list stores the information of the candidates for the shortest paths beginning from u with their distances in the ascending order. Each entry in EL has the same elements as the entry in SL except that the path corresponding to the entry in EL is only a candidate of the shortest path and the real shortest paths will be chosen from EL.

SC: the counter of the shortest paths beginning from u already added in SL.

EC: the counter of the candidates for the shortest paths beginning from u already added into EL.

VS: the set of nodes that for any node $v \in VS$, the shortest path from u to v has been added to SL.

Flag: $Flag = 1$ indicate that all the shortest paths from u with distance not larger than T have been found, otherwise $Flag = 0$.

Following is the details of T-closure algorithm for vertex u, which includes three processes: P1, P2 and P3. P1 is the initialization of the algorithm; P2 is the process that will deal with the incoming requests; P3 is the process that will deal with the replies from the neighbors. In the algorithm description and the following discussion, for a vertex u we use $u.variable$ to represent $variable$ of u; we use $SL_i.dv$, $SL_i.dt$ and $SL_i.pass\_ver$ to represent the fields of the ith entry in SL, which is the same for EL. So $u.SL_i.dv$ means the field dv of $u.SL$'s ith entry.

```
P1: {Initialization of the $T$-closure algorithm}
1: $SL \Leftarrow null$
2: $EL \Leftarrow null$
3: $VS \Leftarrow null$
4: $Flag \Leftarrow 0$
5: $EC \Leftarrow 0$
6: for any vertex $v \in$ neighbors of node $u$ do
7:   if $W(u,v) \le T$ then
8:     add an entry $(v, W(u,v), v)$ into $EL$
9:     $EC \Leftarrow EC + 1$
10:  end if
11: end for
12: if $EL = null$ then
13:   $SC \Leftarrow 1$
14:   add the entry $(u, 0, u)$ into $SL$
15:   $Flag \Leftarrow 1$
16: else
17:   $SC \Leftarrow 2$
18:   $VS \Leftarrow \{u, EL_1.dv\}$
19:   add the entry $(u, 0, u)$ into $SL$ {the first shortest path}
20:   add the entry $(EL_1.dv, EL_1.dt, EL_1.pass\_ver)$ into $SL$ {the
      second shortest path}
21:   send a request (tell me your shortest path after $EL_1.dv$) to
      the neighbor $EL_1.pass\_ver$
22:   delete $EL_1$ from $EL$
23:   $EC \Leftarrow EC - 1$
24: end if
25: wait
```

```
P2: {processing a request (tell me your shortest path after ver)
from the neighbor $v$}
1: if cannot find $SL_k$ in $SL(2 \le k \le SC)$ such that $SL_{k-1}.dv = ver$
   then
2:   if $Flag = 1$ then
3:     send the reply (all shortest paths have been found) to $v$
4:   else
5:     put the request into request queue
6:     wait
7:   end if
8: else
9:   send the reply $(SL_k, SL_{k+1}, \ldots, SL_{SC}, EL_1, EL_2, \ldots, EL_{EC})$
     to $v$
10: end if
```

```
P3: {processing a reply $RP$ from the neighbor $v$}
1: if $RP \ne$ (all shortest paths have been found) then
2:   assume there are $n$ entries in $RP$
3:   for $i = 1$ to $n$ do
4:     if there is an entry $EL_k$ in $EL(1 \le k \le EC)$ such that
       $EL_k.dv = RP_i.dv$ then
5:       if $EL_k.dt > W(u,v) + RP_i.dt$ then
6:         create a new entry $E = (EL_k.dv, W(u,v) + RP_i.dt, v)$
7:         delete $EL_k$ from $EL$
8:         add the new entry $E$ into $EL$
9:       end if
10:    else if $RP_i.dv \notin VS$ and $W(u,v) + RP_i.dt \le T$ then
11:      add into $EL$ the entry $(RP_i.dv, W(u,v) + RP_i.dt, v)$
12:      $EC \Leftarrow EC + 1$
13:    end if
14:  end for
15: end if
16: if $EL = null$ then
17:   $Flag \Leftarrow 1$
18:   Find all the requests in the request queue. Assume the request
      is from the neighbor $v$, send to $v$ the reply (all shortest paths
      have been found)
19: else
20:   add an entry $(EL_1.dv, EL_1.dt, EL_1.pass\_ver)$ into $SL$;
21:   $VS \Leftarrow VS + \{EL_1.dv\}$
22:   $SC \Leftarrow SC + 1$
23:   Find all the requests (tell me your shortest path after ver)
      in the request queue such that we can find the entry $SL_k$ in
      $SL(2 \le k \le SC)$ satisfying $SL_{k-1}.dv = ver$. Assume the re-
      quest is from node $v$, send to $v$ the reply $(SL_k, SL_{k+1}, \ldots, SL_{SC}$
      $, EL_1, EL_2, \ldots, EL_{EC})$
24:   send the request (tell me your shortest path after $EL_1.dv$) to
      the neighbor $EL_1.pass\_ver$
25:   delete the entry $EL_1$ from $EL$
26:   $EC \Leftarrow EC - 1$
27:   wait
28: end if
```

In fact, T-closure algorithm is a controlled depth-first search algorithm for the shortest paths. Every time T-closure algorithm extends the shortest path found most latterly and compares the distance of the extension with those of all the candidates. Based on the comparison, T-closure algorithm decides whether to go on with the extension or to choose a candidate as the new shortest path. In the T-closure algorithm, every time vertex u chooses the first entry EL_1 in EL, adds $(EL_1.dv, EL_1.dt, EL_1.pass\_ver)$ as a new entry into SL. Then u sends a request to its neighbor $EL_1.pass\_ver$(extension of the shortest path found most recently). When u gets the reply from $EL_1.pass\_ver$, u updates its EL(comparing the distance of the extension with that of the candidates) and repeats the computation. When u gets a request from its neighbor, if the request can be processed, u sends the reply including the corresponding entries in SL and all the entries in EL to the neighbor, otherwise the request is put into the request queue. There is one thing needs to be mentioned that when u finds that its *Flag* is 1, u will send *finish message* to all the vertices that the shortest paths distances between u and these vertices are not larger than T. When u gets *finish message* from all these vertices, it will terminate the T-closure algorithm. Then the entries stored in $u.SL$ give the information for all the shortest paths beginning from u with distance not larger than T.

3.1.2 Analysis of the T-closure Algorithm

THEOREM 1. *There is no deadlock in the T-closure algorithm.*

PROOF. Suppose there is a deadlock when the T-closure algorithm is executed for a graph $G = (V, E, W)$. Let $u_1, u_2, u_3, \ldots, u_n$ be the sequence of nodes involved in the deadlock. According to the T-closure algorithm, u_i must be waiting for a reply from $u_{i\oplus 1}(1 \le i \le n, \oplus$ is the modulo n addition operator) and $u_{i\oplus 1}$ is the neighbor of u_i. Assume for u_i, its request for $u_{i\oplus 1}$ is (*tell me your shortest path after* v_i).

Because u_i sends the request (*tell me your shortest path after* v_i) to $u_{i\oplus 1}$, the entry $(v_i, d_i, u_{i\oplus 1})$ is the most resent entry added into $u_i.SL$ and $d_i = W(u_i, u_{i\oplus 1}) +$ *distance from $u_{i\oplus 1}$ to v_i*. Correspondingly $(v_{i\oplus 1}, d_{i\oplus 1}, u_{i\oplus 2})$ is the most resent entry added into $u_{i\oplus 1}.SL$. Because the request of u_i cannot be processed by $u_{i\oplus 1}$, then we must have *distance from $u_{i\oplus 1}$ to v_i* $\ge d_{i\oplus 1}$. As a result, $d_i > d_{i\oplus 1}$ and we have $d_1 > d_2 > \ldots > d_n > d_1$. It is contradictory. So the deadlock sequence $u_1, u_2, u_3, \ldots, u_k$ cannot happen.

□

THEOREM 2. *When the T-closure algorithm terminates for a graph $G = (V, E, W)$, for any node $u, v \in V$, if the distance of the shortest path from u to v is not larger than T, u must know the existence of v and the distance of the shortest path.*

PROOF. Assume there is a shortest path from v_1 to v_k $(v_1, v_2, \ldots, v_k)$ and the distance is not larger than T. We prove that when the T-closure algorithm terminates, v_1 must be able to know v_k and the distance of $(v_1, v_2, \ldots, v_k)$.

It is obvious that when doing initialization, the entry $(v_2, W(v_1, v_2), v_2)$ is added to $v_1.EL$ or $v_1.SL$ directly. If the entry is added into $v_1.EL$, because (v_1, v_2) is also the shortest path, the entry will never be changed afterward and added to $v_1.SL$ finally. Then v_1 know the existence of v_2 and the distance of (v_1, v_2). Assume for the shortest path $(v_1, v_2, \ldots, v_i)(1 < i \leq k-1)$, v_1 knows the existence of v_i and distance of $(v_1, v_2, \ldots, v_i)$, we prove that v_1 must be able to know the existence of v_{i+1} and the distance of $(v_1, v_2, \ldots, v_i, v_{i+1})$.

Because v_1 knows the existence of v_i and the corresponding shortest path distance, then when the entry $(v_i, \textit{distance of}\,(v_1, v_2, \ldots, v_i), v_1')$ is added to $v_1.SL$, v_1 sends a request to v_1' according to v_i. When v_1 gets the reply, in $v_1'.SL$ there should be an entry $(v_i, dist, v_2')$ and v_1' gets reply from v_2' according to v_i. Use the same deduction, when v_1 gets the reply, we get a sequence of vertices $v_1', v_2', \ldots, v_p', v_i (1 \leq p)$. v_j' sends request to $v_{j+1}' (1 \leq j \leq p-1)$ according to v_i and v_p' sends the request to v_i. It is obvious that when v_1 gets reply from v_1', v_j' must get reply from $v_{j+1}' (1 \leq j \leq p-1)$ and v_p' gets reply from v_i. The distance of $(v_1, v_1', v_2', \ldots, v_p', v_i)$ is equal to the distance of $(v_1, v_2, \ldots, v_i)$, otherwise there can not be the entry $(v_i, \textit{distance of}\,(v_1, v_2, \ldots, v_i), v_1')$ in $v_1.SL$. Then $(v_1, v_1'), (v_j', v_{j+1}')(1 \leq j \leq p-1)$ and (v_p', v_i) must all be the shortest paths. Because in initialization, the entry $(v_i, W(v_i, v_{i+1}), v_{i+1})$ is added to $v_i.EL$ or $v_i.SL$ directly. So when v_p' gets the reply, v_p' must be able to know the existence of v_{i+1} and the distance of (v_p', v_i, v_{i+1}), which will not be changed because (v_p', v_i, v_{i+1}) is the shortest path between v_p' and v_{i+1}. Use the same deduction, we can get when v_1 gets reply from v_1', the entry $(v_{i+1}, \textit{distance of}\,(v_1, v_1', v_2', \ldots, v_p', v_i, v_{i+1}), v'')$ can be created in $v_1.EL$. Because the distance of $(v_1, v_1', v_2', \ldots, v_p', v_i)$ is equal to the distance of $(v_1, v_2, \ldots, v_i)$, then the distance of $(v_1, v_1', v_2', \ldots, v_p', v_i, v_{i+1})$ is equal to the distance of $(v_1, v_2, \ldots, v_i, v_{i+1})$. As a result, the entry $(v_{i+1}, \textit{distance of}\,(v_1, v_1', v_2', \ldots, v_p', v_i, v_{i+1}), v'')$ won't be changed and added to $v_1.SL$ finally. Then v_1 knows the existence of v_k and the distance of $(v_1, v_2, \ldots, v_i, v_k)$.

$\square$

It is obvious for any vertex $u \in V$, there are at most $N-1$ shortest paths beginning from u with the distance not larger than T(N is the number of vertices in the graph G). In the T-closure algorithm, for any $u \in V$, after each round of communication with the neighbors, a shortest path from u can be added correctly. So for any vertex $u \in V$, u only needs communicate with it neighbors at most $N-1$ times to know all the shortest paths beginning from u with the distance not larger than T. So the total number of communication messages needed in T-closure algorithm is $O(N^2)$.

3.2 Hierarchical Node Clustering

In [17], hierarchical clustering algorithm is proposed to find discrete mobile centers in a wireless network that can be represented by a graph. In the graph, there is an edge between two vertices if and only if a disk of radius R centered at one node contains the other one. The weight of each edge is the distance of the corresponding two nodes in space. The discrete mobile centers problem is equal to finding the minimal dominating set with visible range radius as R in the graph, which is a NP-complete problem. With the hierarchical clustering algorithm, the number of discrete mobile centers we get will be a constant approximation of the optimal solution with high probability [17]. Because of this property, we adopt the hierarchical clustering algorithm in node clustering based on link delay to control the number of node clusters. The graph in [17] is not a general graph, in which if there is a path $(v_1, v_2, \ldots, v_n)$ such that the path distance is not larger than the visible range radius R, there must be an edge between v_1 and v_n and the edge's weight must be no larger than R. It is obvious that the original graph $G = (V, E, W)$ we use to represent a P2P network may not satisfy this requirement. So we propose the T-closure algorithm. After T-closure computation, if we take the visible range radius as T, the graph $G' = (V', E', W')$ we get can satisfy the requirement. Now node clustering based on link delay in G' with the limit T is equal to the discrete mobile centers problem with visible range radius T.

3.2.1 Hierarchical Node Clustering Algorithm

Similar to [17], hierarchical node clustering algorithm depends on a basic clustering operation. As we discussed in the beginning of this section, in node clustering based on link delay, we assume that each node has an unique identification number arbitrarily assigned in the P2P network. So each vertex $v \in V'$ has an unique identification number. The basic clustering operation is: given the graph $G' = (V', E', W')$ and the parameter d, for each vertex $v \in V'$, v nominates another vertex $v' \in V'$ with the largest identification number such that the edge between v and v' is not larger than d. v can nominates itself if there are no other vertices with larger identification number than v. As a result, in the P2P network, the node that its corresponding vertex v in G' is nominated becomes the super-node, all the nodes that their corresponding vertices nominate v become the peer-nodes belonging to the cluster controlled by this super-node.

In fact, the hierarchical node clustering algorithm is a repeated execution of the basic operation with an increasing value for the parameter d. In node clustering based on link delay, for simplicity we assume the link delay limit T is equal to $2^k * t$, t is called *delay base* and k is called *delay exponent*. The hierarchical node clustering algorithm executes the basic operation for $k-1$ rounds and in the rth$(1 \leq r \leq k-1)$ round, the value of the parameter d is set to $2^r * t$. The hierarchical node clustering algorithm is executed in every vertex $v \in V'$ and v will store the identification numbers of the vertices it controls(the vertices that v is nominated by) or the identification number of the vertex it is controlled by(the vertex that v nominates). In the rth$(1 \leq r \leq k-1)$ round, for $v \in V'$, if v is not nominated, v will terminate the execution of hierarchical node clustering algorithm; for the vertex it nominates, v will send message to tell the identification numbers of the vertices it controls; for the vertices it controls, v will send messages to tell the identification number of the vertex it nominates. In the $r+1$th$(1 \leq r \leq k-2)$ round, the hierarchical node clustering algorithm is executed only in the vertices that are nominated in the rth round. In each round, the nodes of the vertices that are nominated become the super-nodes of this round. After the $(k-1)$th round, the nodes of the vertices that are nominated become the super-nodes of the final result.

3.2.2 *Analysis of Hierarchical Node Clustering Algorithm*

THEOREM 3. *With hierarchical node clustering algorithm, we can get a set of node clusters such that every peer-node is within link delay limit T from its super-node.*

PROOF. The proof is similar to the Lemma in [17]. We prove that after the ith round, the link delay between a peer-node and its super-node is not larger than $2^{i+1} * t$. When $i = 1$, this is obviously true. Assuming it is true for i, we consider the $(i+1)$th round. If we take a peer-node v and its super-node v' in the ith round, then the link delay between v and v' is not larger than $2^{i+1} * t$. If v' does not nominate any other nodes, then v' nominate itself as super-node in $(i+1)$th round and v is still the peer-node of v', so the link delay between v and v' is clearly not larger than $2^{i+2} * t$. If v' nominates v'' as super-node, then the link delay between v and v'' is not larger than $2^{i+1} * t + 2^{i+1} * t = 2^{i+2} * t$. So after $(k-1)$th round, the link delay between the peer-node and its super-node is never larger than $2^{k-1+1} * t = T$. $\square$

It is proved in [17] that in hierarchical clustering algorithm, if the basic operation is executed for $\lfloor loglog(N_w) - 1 \rfloor$ rounds (N_w is the number of nodes in the wireless network), the discrete mobile centers in the result will be a constant approximation of the optimal solution with high probability. In fact, $loglog(N_w)$ is a small value. For example, if N_w is 2^{30} (roughly 10^9), $loglog(N_w)$ is less than 5. As we discussed, the node clustering based on link delay problem in G' with the limit T is equal to the discrete mobile centers problem with visible range radius T. So for a P2P network with N nodes, if we set the value of k, t to satisfy: $2^k * t = T$ and $k > \lfloor loglog(N) - 1 \rfloor$, then the node clusters we get will also be a constant approximation of the optimal solution with high probability. In a P2P network with N nodes, for each round in the hierarchical node clustering algorithm, the number of communication messages is $O(N)$. So the total number of communication messages is $O(kN)$

4. EXPERIMENTS AND RESULTS

The first experiment shows the effectiveness of our T-closure computation, the second experiment shows the tolerance of our approach to the changes in the P2P network.

4.1 Effectiveness of T-closure Computation

As we discussed, the original graph G for a P2P network may not satisfy the requirements of hierarchical clustering algorithm. So in the experiment, we execute hierarchical node clustering algorithm with and without T-closure computation to see the difference. In our experiment, we modeled a P2P network with 1000 nodes and the weight of the edge between two nodes is randomly generated. In the experiment, we have delay exponent k as 5 because $\lfloor loglog1000 \rfloor < 5$. We compared the value of $\frac{number of clusters}{1000}$ in each round of hierarchical node clustering with and without T-closure computation respectively. Figure 1 is the result of the experiment. As we can see from the figure, with T-closure computation, the number of clusters in hierarchical node clustering will decrease dramatically compared with that without T-closure computation. In fact only with T-closure algorithm, the node clusters we get can be a constant approximation of the optimal solution with high probability.

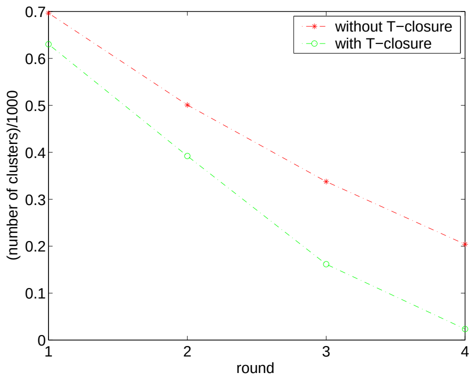

Figure 1: Effectiveness of T-closure Computation

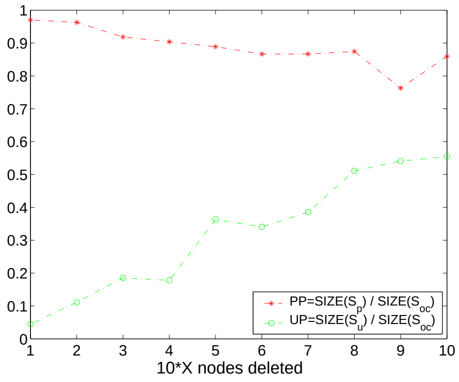

Figure 2: Adaptation for Node Deletion

4.2 Tolerance to Changes

With this experiment, we want to see if some changes take place in the P2P network, how the new set of super-nodes will differ from the original set of super-nodes. In this experiment, we also modeled a P2P network of 1000 nodes and the edge weights are randomly generated. We firstly did node clustering in the original P2P network. After that, we made some random changes in the P2P network, which include changing the value of link delay between nodes and deletion of some nodes. Then we did node clustering again and compared these two clustering results. Assume the set of super-nodes in the first node clustering is S_{oc} and the set of super-nodes in the second node clustering is S_{nc}. We define the preserving set S_p and the updating set S_u as: $S_p = S_{oc} \cap S_{nc}$; $S_u = S_{nc} - S_{oc}$. Consequently we define preserving percentage PP and updating percentage UP respectively as:

$$PP = \frac{SIZE(S_p)}{SIZE(S_{oc})}; UP = \frac{SIZE(S_u)}{SIZE(S_{oc})}.$$

PP describes the percentage of nodes in S_{oc} that are still super-nodes in the second clustering result; NP describes the percentage of nodes that are newly added after the changes according to S_{oc}.

Firstly we randomly chose $10 * i (i = 1, \ldots, 10)$ nodes in the original P2P network, deleted them from the P2P network and did node clustering. The result of PP and UP is showed in Figure 2. Secondly we randomly chose 10, 50 and 100 nodes in the original P2P network, increased their edge weights by $2^i * t (i = 1, 2, 3, 4)$ and then did node clustering.

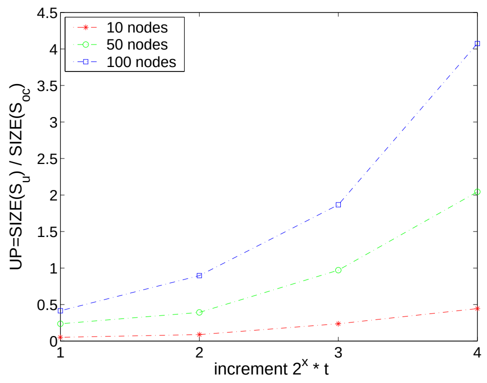

Figure 3: UP according to edge increment

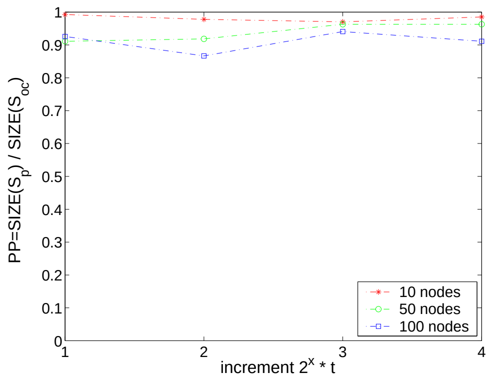

Figure 4: PP according to edge increment

The experiment results of PP and UP for each weight increment are showed in Figure 3 and 4. From these figures, we can see that when not too many changes take place in the original P2P network, the new set of super-nodes will not differ too much from the original one. As a result, the original clustering result can still be used when the changes are not dramatic.

5. CONCLUSION

In this paper, we propose an approach implementing node clustering based on link delay in P2P networks. With our approach, node clustering in P2P networks is implemented in a completely distributed way and does not require the global knowledge of P2P networks. In our approach, not only node connectivity but also the quality of node connection are studied. As a result, the link delay of communication between super-nodes and peer-nodes can be limited. With hierarchical node clustering, our approach can get the set of node clusters that is a constant approximation of the optimal solution for node clustering based on link delay with a high probability. Consequently the overall performance in P2P networks can be improved.

6. REFERENCES

[1] J. Wu and I. Stojmenovic, "Ad hoc network," *Computer:Ad Hoc Network*, vol. 37, no. 2, 2004.

[2] M. Singh, V. K. Prasanna, J. Rolim, and C. S. Raghavendra, "Collaborative and distributed computation in mesh-like sensor arrays," in *the IFIP-TC6 8th International Conference on Personal Wireless Communications(PWC)*, 2003.

[3] N. Daswani, H. Garcia-Molina, and B. Yang, "Open problems in data-sharing peer-to-peer systems," in *9th International Conference on Database Theory(ICDT)*, 2003.

[4] J. Li and P. Mohapatra, "A novel mechanism for flooding based route discovery in ad hoc networks," in *Wireless Communications Symposium, GLOBECOM*, 2003.

[5] P. Mohapatra, C. Gui, and J. Li, "Group communications in mobile ad hoc networks," *IEEE Computer, Special Issue on Ad Hoc Networks*, 2004.

[6] S.-J. Lee, J. Hsu, R. Hayashida, M. Gerla, and R. Bagrodia, "Selecting a routing strategy for your ad hoc network," *Computer Communications, special issue on Advances in Computer Communications and Networks: Algorithms and Applications*, vol. 26, no. 7, pp. 723–733, 2003.

[7] B. Yang and H. Garcia-Molina, "Designing a super-peer network," in *the 19th International Conference on Data Engineering (ICDE)*, Bangalore, India, 2003.

[8] A. McDonald and T. Znati, "A mobility based framework for adaptive clustering in wireless ad-hoc networks," *IEEE Journal On Selected Area of Communications*, vol. 17, no. 8, pp. 1466–1487, 1999.

[9] B. Das and V. Bharghavan, "Routing in ad-hoc networks using minimum connected dominating sets," in *IEEE International Conference on Communication*, 1997, pp. 376–380.

[10] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis, "Connectivity based node clustering in decentralized peer-to-peer networks," in *the 3rd International Conference on Peer-to-Peer Computing*, 2003.

[11] Kazaa website. [Online]. Available: http://www.kazaa.com

[12] S.Basagni, "Distributed clustering for ad hoc networks," in *99' International Symp. On Parallel architecture, algorithms, and Networks (I-SPAN'99)*, 1999, pp. 310–315.

[13] A. Amis, R. Prakash, T. Vuong, and D. Huynh, "Max-min d-cluster formation in wireless ad hoc networks," in *IEEE INFOCOM'2000*, 2000.

[14] D. Kim, S. Ha, and Y. Choi, "K-hop cluster-based dynamic source routing in wireless ad-hoc packet radio networks," in *IEEE Vehicular Technology Conference*, 1998, pp. 224–228.

[15] S. van Dongen, "A cluster algorithm for graphs," in *Technical Report INS-R0010*, National Research Institute for Mathematics and Computer Science in the Netherlands, Amsterdam, May 2000.

[16] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan, "A cluster-based approach for routing in dynamic networks," *ACM Computer Communications Review*, vol. 27, no. 2, pp. 49–64, 1997.

[17] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu, "Discrete mobile centers," in *the 17th ACM Symposium on Computational Geometry (SoCG'01)*, 2001, pp. 188–196.