Journal of Computer Science and Cybernetics, V.34, N.2 (2018), 145–159
DOI 10.15625/1813-9663/34/2/12723

CONGESTION CONTROL ALGORITHM FOR MESSAGE ROUTING IN STRUCTURED PEER-TO-PEER NETWORKS

NGUYEN DINH NGHIA[1], NGUYEN HOAI SON[2]

[1]*People's Security Academy*
[2]*VNU-University of Engineering and Technology*
[1]*nghiahvan@gmail.com*

Abstract. In structured peer-to-peer (P2P) networks, every node must perform message routing to deliver query messages to destination nodes. Therefore, they often have to process a large number of query messages quickly and efficiently. When the number of query messages sending to a node exceeds its routing capacity, a local congestion on the node occurs and reduces the information processing of the other nodes in the network. This paper proposes a congestion control algorithm for message routing in structured P2P networks by changing the routing table of nodes in the network to route the packet to the destination without going through congested nodes. The performance of the proposed the method has been evaluated and compared with the conventional Chord protocol. The result shows that our proposed method improves the query success rate significantly while minimizing the number of packets generated during congestion control process.

Keywords. Peer-to-peer, distributed hash, DHash, Chord.

1. INTRODUCTION

Distributed hash tables (DHTs) are efficient routing algorithms for storing and searching data in structured peer-to-peer networks. DHT algorithms determine data location through an identifier (i.e. a key) in the identifier space. Each node is responsible for managing a subset of identifiers. The basic operation of a DHT is to search for an identifier and return the address (i.e. IP address and port number) of the node responsible for the identifier. Each node maintains a routing table to forward queries that it does not manage. DHTs, such as P-Grid [1], Chord [2] or Pastry [3] create routing tables to move the packet to the destination node.

In recent years, DHT is used not only for file sharing applications but also to build peer-to-peer information Retrievals (P2P-IR) [4]. In P2P-IR systems, a DHT must handle concurrently a large number of messages in a short time, which significantly increases system load. Optimizing routing method in the DHT networks is an important requirement for P2P-IR systems. Furthermore, when a node receives a number of query messages that exceeds its routing capacity, a congestion occurs. If there is no proper congestion control mechanism, the next queries arriving or going through this node will create a congestion on the network that can make it collapse.

DHT congestion control belongs to an upper layer and is independent of TCP congestion control mechanism. DHTs often choose greedy algorithms to route packets in the network where a node

always selects the nearest neighbor to forward the message to the destination node. The P2P networks are heterogeneous where the ability of the nodes to process packets is not the same, the stability of the nodes is not high and the nodes have to share the CPUs and the network resources for other processes that are running at the same time, etc. Therefore, selecting neighboring nodes to forward packets to the destination may not be responsive to the dynamical change of the system load in a DHT network.

Researches in congestion control in P2P networks are either to limit the sending speed of the query sending nodes or to change the message route to avoid congestion. There are a number of studies on congestion control in P2P such as the CSCC [5], the BPCC [5] and the CCLBR [6]. Those methods have a common characteristic: they use a fixed routing table and perform congestion control by reducing the sending speed of the query node or by using alternative message route in the routing table while ignoring the nodes that have high processing capacity in the network. Therefore, they might reduce the transmission speed of the network when a congestion occurs.

In this paper, we propose a new congestion control method in a structured P2P network by adapting the routing table of each node based on congestion status of next nodes in each message route. Whenever a node receives a query message, it first checks its congestion status based on the node's routing capacity and then sends the status information back to the sending node if it is congested. Each sending node monitors congestion feedback information sent from congested nodes and replaces any congested node by a non-congested node in the message route. Our method allows nodes to quickly adapt routing to avoid overloaded parts of the network. Therefore, the method can efficiently use the throughput and resources of the non-congested nodes without affecting the routing performance of the whole network.

The contributions of this paper are in three folds. Firstly, we propose a congestion detection method which is capable of detecting the congestion status of a node based on node routing capacity before the node discards query messages due to congestion. Secondly, when the congestion status of a node is detected, a congestion avoidance solution, which changes the routing table of nodes to efficiently re-route query messages to non-congested nodes, is proposed. Lastly, a post-processing method is proposed to gradually turn the network back to normal routing when the query rate reduces and congested nodes turn back to congestion-free status.

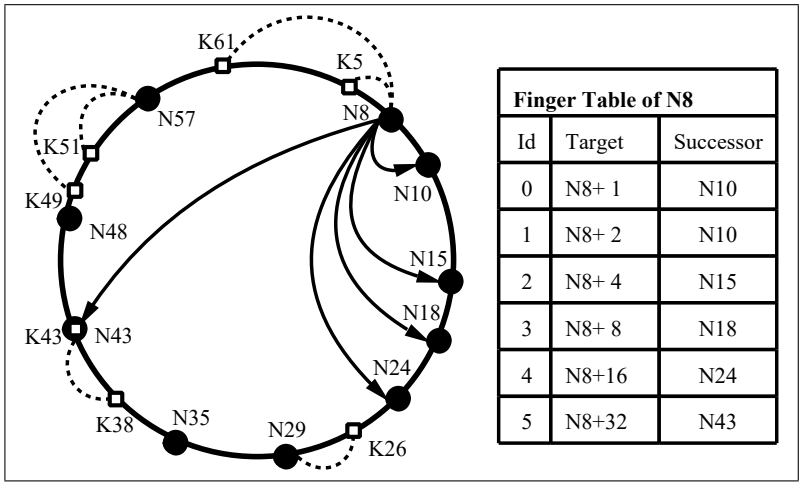
The approach was evaluated through a number of experimental simulations, which simulate a real P2P network. In our simulation, the routing capacity of the nodes to join the network is heterogeneous and node churn is based on Pareto distribution. The simulation results show that our proposed method can achieve higher successful query rate than the original Chord protocol.

This paper is structured as follows: Section 2 summarizes the relevant studies. Section 3 describes our proposed algorithm. Section 4 presents the analysis and evaluation of the algorithm. Finally, we provide some discussions and the conclusion in Section 5.

2. BACKGROUND

2.1. Chord protocol

Chord protocol is a representative structured P2P network protocol. Chord uses a logical identifier namespace to manage data and computers in the network. The identifier of a computer (i.e. node) in a Chord network is a n-bits integer, which can be the return value of the SHA-1 hash function [7] on



Finger Table of N8

Id	Target	Successor
0	N8+ 1	N10
1	N8+ 2	N10
2	N8+ 4	N15
3	N8+ 8	N18
4	N8+16	N24
5	N8+32	N43

Figure 1. 8-bits Chord identifier space. Dashed lines indicate that the key is stored at that node. The black lines represent the cursors in the routing table of a node with the identifier of N8. The table on the right is the finger table of a node with the identifier of N8

the IP address and the port address of the computer. Data identifiers are computed as the hash value of data name or data content. The Chord network identifiers namespace includes integers ranging from 0 to 2^n - 1 and is organized in a circle, also known as the Chord circle. Data with identifier k is stored at the node with identifier k, or node with the identifier next to k in a clockwise order. The node that manages data identifier k is called the successor of k and is denoted as Successor(k). The next node of the node with identifier n *(Id = n)* in the clockwise order of the Chord circle is called the successor of node n, denoted as Successor(n). The previous node of the node with identifier n in clockwise order in the Chord circle is called the predecessor of node n, denoted as Predecessor (n).

When a node n joins the Chord network, it will be assigned to manage data with identifier n *(k = n)* or with identifier preceding n in the clockwise order of the Chord circle. When a node n leaves the network, all data nodes managed by node n will be assigned to Successor (n).

Figure 1 shows the Chord circle drawn with 6 bits identifiers (64 identifiers), consisting of 10 nodes and 7 keys. Successors of *K5* and *K61* are *N8*, hence keys *K5* and *K61* are stored at node *N8*. Similarly, key *K26* is stored at node *N29*, keys *K38* and *K43* are stored at node *N43* and keys *K49*, *K51* are stored at node *N57*. Each node maintains a routing table (i.e. finger table) of m rows. Here m is the number of bits in the Chord identifier. In the routing table of node n, row i (the finger i^{th}) contains node s with the identifier Id = successor $(n + 2^i\text{-}1)$ where $1 \leq i \leq m$. Node s is the i^{th} finger of node n (n.finger [i]). Figure 1 also shows the finger table of node *N8*. The first finger of this node is the pointer to node *N10* (the first node after the node (8 + 20 = 9)). Similarly, the last finger in the table is the pointer to node *N43*.

2.2. Conventional congestion control methods

Studies related to congestion control in structured P2P networks have focused primarily on two directions: speed control and routing tables. In speed control, query sending rate is limited at the query node when a node in the routing path is overloaded. Typical methods include credit system

congestion control (CSCC) [5], back-pressure congestion control (BPCC) [5] and Marking method [8]. In CSCC method, when a destination node receives a packet, it responds to the query node with an ACK packet. At the intermediate packet forwarding node, a queue is maintained to store incoming queries. When the queue reaches the threshold, subsequent incoming queries are discarded without informing the query sending nodes. Each node in the network maintains a credit parameter that indicates the number of queries that have been sent but has not yet received the ACK packet. If the credit is less than the threshold, the credit value continues to increase. If the credit reaches the threshold, the credit value will be increased more slowly with each received ACK packet. If a packet loss occurs, the threshold value will be reduced. When an outgoing packet is lost, the sending node re-sends the packet.

CSCC works similarly to TCP congestion control. However, in structured P2P network CSCC operates with relatively complicated connection process with many transition nodes and low node's stability. In this environment, it is difficult to calculate the timeout value to determine the lost packets. In addition, a packet can be received multiple times, so each node needs to reserve resources to keep a list of packets received from each sending node. CSCC performs packets dropping, thus results in decreasing the throughput of the network.

In BPCC method, each node maintains a queue for incoming queries. When it receives a packet, if the queue reaches a limit, it sends its queue state to the sender. The sending node will automatically adjust the sending speed to avoid congestion. When the queue at a node P is full, it will suspend reading the packet from the TCP socket of incoming connections instead of rejecting the packet. Then the flow control algorithm in TCP automatically slows down the sending of queries to P. This method is at risk of deadlock formation. Deadlock occurs when a sending node waits for a receiving node to process packets in the queue while that receiving node has to wait for another node. BPCC does not drop packets, thus increases the throughput of the system. However, this method may raises deadlocks and it is very difficult to detect and handle those deadlocks, especially in a peer-to-peer environment that involves many free, uncontrolled nodes. In addition, the wait on the nodes increases the response time of the queries.

In Marking method, each node organizes a queue for incoming queries and returns congestion status by adding a h flag to the packet header. The goal is to keep the resource shared to bottlenecked nodes fairly without overloading the congestion node. By default, the h flag is off when the query packet is created. Flag h off indicates a non-congested network state and vice versa. Each node adds an x value into the packet to represent its current query speed. Every node stores the average number of packets xavg, which is calculated on the received values from some previously coming packets. The h flag is enabled or disabled with probability q determined by considering whether the packet arrives at a faster or slower rate than the mean xavg and the average value of the queue size. When the flag h is turned on, it will not be turned off along the way to the final destination. The destination node copies the value of h to the ACK packet and returns it to the query sending node. When a node receives an ACK packet with the h flag being off, i.e. the network has no congestion, it will speed up the query. If the h flag is on, it decreases the speed of the query. Query speed is incrementally increased and exponentially decremented.

Each node also stores an estimated RTT (round trip time) value for the query. The RTT value is calculated based on the time it is sent and the time its response is received. RTT values used for the entire network are independent of the destinations of the queries and are often very large.

After the duration of RTT, if a node does not receive the response for a query, it decreases the query sending speed and resends the query.

Because the Marking method does not eliminate packets as well as the lightweight mechanism of sending congestion status to the source node, it results in better throughput and better query response time than the two methods CSCC and BPCC.

In another direction, the routing table method is based on the status of each node. This approach changes the path of the packet to the less congested nodes. In adaptive routing method [9], instead of selecting the closest node to the key, it selects k nodes near to the key and monitors the performance of those nodes to determine the best path. The query sending node uses the congestion flag h to determine the usability of each routing path via the k node near the selected key. From the received h value, each node calculates the probability (called op) of observing h on each path. Based on this probability value, each node will move routing traffic through different paths to increase throughput across the network. A node updates the op values every time a new query is made. When a node forwards queries to other nodes, it divides traffic in different directions based on the calculation of the capacity of each route.

A disadvantage of the adaptive routing method is that, when we want to increase the number of routing paths, we must increase the size of the routing table, which will consume resources of the system. In addition, the abandonment of greedy routing methods will increase the number of packets transmitted over the network.

Other approaches include the CCLBR [6], the methods in [8], [10] and [11]. The CCLBR [6] proposed a resource grouping and a rewiring method to spontaneously organize and cluster the peers having similar resources together. Then, a collaborative Q-learning method is proposed to balance the query loads among the intragroup peers in order to intelligently avoid queries being forwarded to the congested peers in the network. The work [8] can achieve high network throughput, but each node has to spend its resources to manage the increase in the size of the routing table. The work [10] proposed a DHT-based routing mechanism to improve the performance of the smart grid. In another direction, the work [11] proposed a P2P traffic optimization model with congestion distance and introduced an optimization scheme based on congestion distance and Distributed Hash Table (DHT). Those methods have a common limitation: they use a fixed routing table and perform congestion control by reducing the sending speed of the query node or by using alternative message route in the routing table. Therefore, they do not make full use of the capacity of nodes that have high processing capacity in the network for congestion avoidance.

3. THE CA-CHORD METHOD

In this section, we propose a congestion control method called CA-Chord, which can avoid message routing via congested nodes in structured P2P. In our method, when a congested node receives a query, it will notify the sending node (i.e. the source node or the forwarding node) so that the sending node will change the routing table by replacing the congested node with a neighboring node that is not in the congestion status. When a congested node changes back to the normal status, it will inform the nodes that already change their routing tables due to its congestion status to change the routing tables back to original entries.

Our proposed method can avoid local congestion by utilizing the capacity of non-congested nodes for message routing, but does not increase hop count of routing message as well as affect the Chord

network's model and the size of the routing table. The CA-Chord method works as follows:

Step 1. Congestion detection

Each node in the system has a routing capacity C, which corresponds to the maximum number of query messages that a node can route per unit time. The C value of a node can be estimated based on the size of query messages and the bandwidth that can be used for message routing. For example, if the size of query message is 200Byte and the bandwidth of a node is 100Mps but a user sets the bandwidth used for message routing to be 10Mbps, then the routing capacity of the node is 6250 messages per second.

If the incoming message rate to a node, which is calculated by counting the number of incoming messages per second, is over the routing capacity of the node, it will not process any more packets and the incoming packets are discarded. This state is called *"hard congestion state"*. In other words, the hard congestion state of a node indicates that the node is completely congested.

To avoid packet loss due to hard congestion state, each node monitors the incoming message rate for early congestion detection. We define a parameter T called soft congestion threshold of a node as an indicator of congested status of the node. The soft congestion threshold is calculated as follows

$$T = p * C, \tag{1}$$

where p is a system parameter and $0 < p < 1$.

When the incoming message rate of a node reaches the soft congestion threshold, we consider that the node is congested. In this case, the node will trigger the congestion avoidance process but the incoming packets are handled as usual. This state is called *"soft congestion state"*.

The value of the soft congestion threshold T affects the "sensitivity" of the congestion detection. If T is too small, the nodes will easily be considered as congested even if they are still capable of serving. On the other hand, if T is too large, the congestion processing is performed too late, leading to hard congestion state of the node and the drop of query messages. Hence, the value of the soft congestion threshold must be determined carefully based on experiments.

Each node maintains a list of successor nodes that are not in the congestion status. When a node changes its state from non-congested state to congestion status or vice versa, it will inform nodes in its predecessor node list (as in conventional Chord's protocol) about its state. These nodes will update their list of non-congested successor nodes when they receive the notification.

Step 2. Congestion avoidance

When a node in the *"soft congestion"* state receives a new message, it will carry out the following tasks to avoid message routing via the congested node.

- The congested node sends a congestion notification message to the node that sends the packet to notify its congestion status and the information of the alternative node. Here, the first node in the list of non-congested successor nodes is selected as the alternative node. This selection method reduces the number of nodes (i.e. hop count) that the packet must pass through. Whenever a packet is sent to a congested node, if the sending node does not received any congestion notification message, the congested node will send a congestion notification message to the sending node and store the information of the sending node into its database to eliminate the redundancy of sending notification messages.

Table 1. The initial routing table of node P_i

Idx	Target ID	Active Route	Origin Route
1	$N_i + 1$	P_a	P_a
2	$N_i + 2$	P_b	P_b
...	...	...	...
$k-1$	$N_i + 2^{k-1}$	P_{k-1}	P_{k-1}
k	$N_i + 2^k$	P_k	P_k
...	...	...	...
m	$N_i + 2^m$	P_x	P_x

Table 2. The routing table of node P_i after being modified

Idx	Target ID	Active Route	Origin Route
1	$N_i + 1$	P_a	P_a
2	$N_i + 2$	P_b	P_b
...	...	...	...
$k-1$	$N_i + 2^{k-1}$	P_{k-1}	P_{k-1}
k	$N_i + 2^k$	P_t	P_k
...	...	...	...
m	$N_i + 2^m$	P_x	P_x

- When the sending node receives the congestion notification message, it replaces the congested node in the entry of its finger table with the alternative node.

- New packets originally routed to the congested node now will be passed through a non-congested node in the new route.

To change the routing table while retaining the information of original destination nodes for post-congestion process, each node maintains a routing table which contains information of both active route and origin route. The active route is an alternative used for message routing in case the origin route, which is inititally designed route, is congested.

An example of congestion avoidance process is given as follows. Suppose that node P_i has an identifier of N_i, the initial routing table of node P_i is as shown in Table 1.

When the node P_k is congested, if node P_i forwards a packet to node P_k, node P_k will notify its congestion status to node P_i. The routing table of node P_i then will be modified as shown in Table 2.

Here, node P_t is the alternative node of node P_k in the finger table of node P_i. If there is a packet arriving at node P_i and the message will be routed to node P_k according to the original finger table, the query message will be forwarded to (according to the original Chord protocol):

- P_t if the query key is not in the range of P_k and P_t,
- P_{k-1} if the query key is in the range of P_{k-1} and P_t.

Step 3. Post-processing after dealing with congestion

When the state of a node changes from a congestion status to a normal state (i.e. the number of

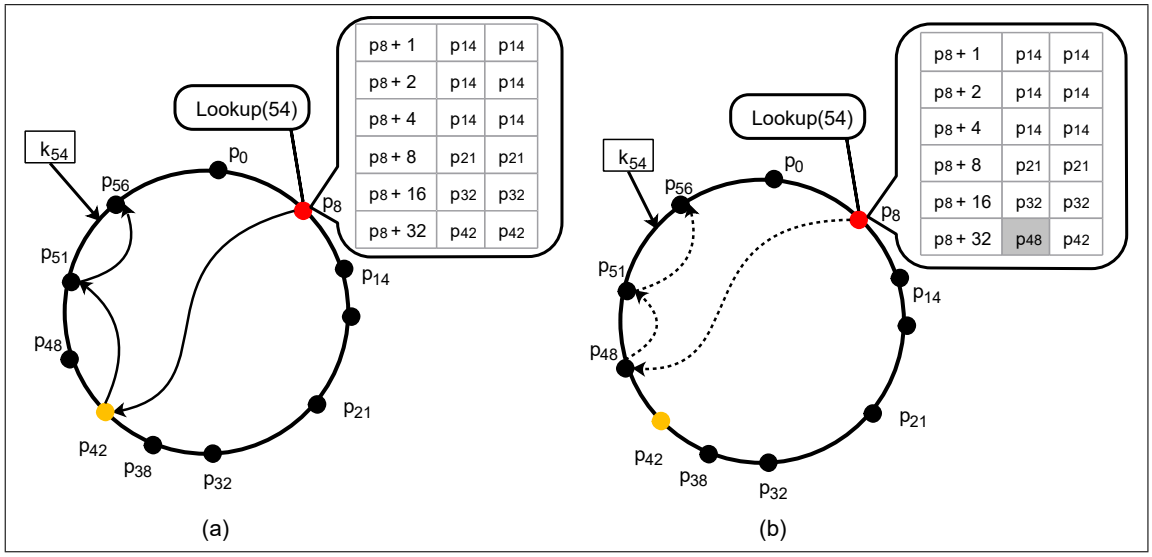


Figure 2. A normal query in Chord network ($m = 8$)

incoming messages in a time unit is smaller than soft congestion threshold), the following operations will be performed.

- The node that has just left the congestion status will send a congestion-free notification to the nodes that already received its congestion notification.

- The node that receives the congestion-free notification will change the corresponding active route in the routing table to the original route. For example, when P_t receives a congestion-free notification from P_k, it then switches the entry of active route in its routing table back to P_k.

When a node, say P_k, leaves the congestion status, it only selects a certain number (say z) of nodes that already received its congestion notification to send the congestion-free notification in one time interval. That number z of nodes will directly affect the level of load recovery of the node. If the number z is set too small, it will cause the receiving node to return to normal routing state slowly. If it is set too high, it can make the non-congested node congested again easily.

Consider a Chord network with 8-bit key space (64 identifiers) as shown in Figure 2.

Node P_8 performs a query with the key $k = 54$. When the system is operating normally, the finger table of node P_8 is as shown in Fig. 2 (a). When node P_{42} is congested, it will announce the congestion status to node P_8. P_8 will change its finger table as shown in Fig. 2(b). On node P_8, all routing paths via node P_{42} will be changed to P_{48} (the successor of P_{42}). The number of nodes that the query must pass through is still 3, that is the number needed in the initial case.

Assuming node P_{48} is in the congestion status. In the routing table of P_8, node P_{48} will be replaced by P_{51}. On node P_8, assume that we need to query the key 49. Since key 49 belongs to the range [42, 51], the query will have to go through node P_{42} and be processed as normal.

During operation, we assume that some nodes with routing table passing through node P_{42} have to change to other nodes when nodeP_{42} is congested. When P_{42} gets out of congestion status, the nodes whose routing table have been changed will be restored to the original status one by one.

4. ALGORITHM EVALUATION

To evaluate the effectiveness of the proposed algorithm, we have performed a number of experiments and compared the result with the Chord algorithm. The experiments were evaluated on a simulated network that is similar to the real-world network. The simulated network consists of 4096 physical nodes and operates in discrete time using the input parameters of J. Ledlie [12]. Each time step includes scenarios: node entering/leaving the system, updating the route table, routing query messages.

The node in and out of the system: At each step, nodes arrive and depart with the lifetime of the nodes based on Pareto birth/death distributions. We generated several Pareto birth/death distributions with average lifetime of a node set to be 15 minutes, 30 minutes, 1 hours, 2 hours and 3 hours. The simulation time is set to be 3 hours but we recorded statistics only for the second half of a simulation to avoid instabilities of simulation results. The routing capacity of a node is generated randomly from 1 to 399,999 query messages per second. The average routing capacity of a node is 8,000 query messages per second and is fixed for all the experiments.

Update path finding table: each node in the system uses the Chord mechanism [2] to carry out its path finding table update.

Query data: we generate random sets of query keys according to Uniform distribution and Zipf distribution with the query speed being 0.01, 0.1, 1, 10, 20 or 100 queries per second accordingly and store them in different query key files. Set of query keys is generated based on the Zipf distribution, which reflects the popularity of a query key based on a parameter called a rank. The probability that a key appears in the set of query keys is in proportion to $1/r^{\alpha}$. Here, r is the rank of the key and is a random number between 1 and the total number of query keys, α is a constant number. In each simulation, we set the value of α among 0.8, 1.2, 2.4, 4.8.

In each simulation, query keys are selected from query key files and randomly assigned to a query node. A query node sends the query message through multiple intermediate nodes on the routing path to the destination node responsible for the query key. If an intermediate node is in hard congestion, the query message is discarded and the query is considered failed. A query is considered successful if the query message reaches the destination node.

We present and discuss the experimental results in following subsections.

4.1. Evaluation of query success rate

This section presents our evaluation on the success rate of queries when we vary different parameters, including the average lifetime of nodes in a network, the average number of queries set in a node, the T, and the Zipf distribution parameter.

In the first experiment, the query keys set in each node are initialized as Uniform and Zipf distributions with the parameter $\alpha = 0.8$. For each round of a simulation, we perform 20 queries per node and count the number of routing queries successfully by query nodes. The average lifetimes of a node are 15 minutes, 30 minutes, 60 minutes, 120 minutes and 180 minutes, respectively. For each lifetime of a node, a data file is created to keep track of the node's entering and leaving. The parameter T of a node is set to 50% of the node's routing capacity.

The result is depicted in Figure 3. It shows that the success rate of our CA-Chord algorithm is higher than that of conventional Chord routing algorithm by 42% for Uniform queries and 37% for

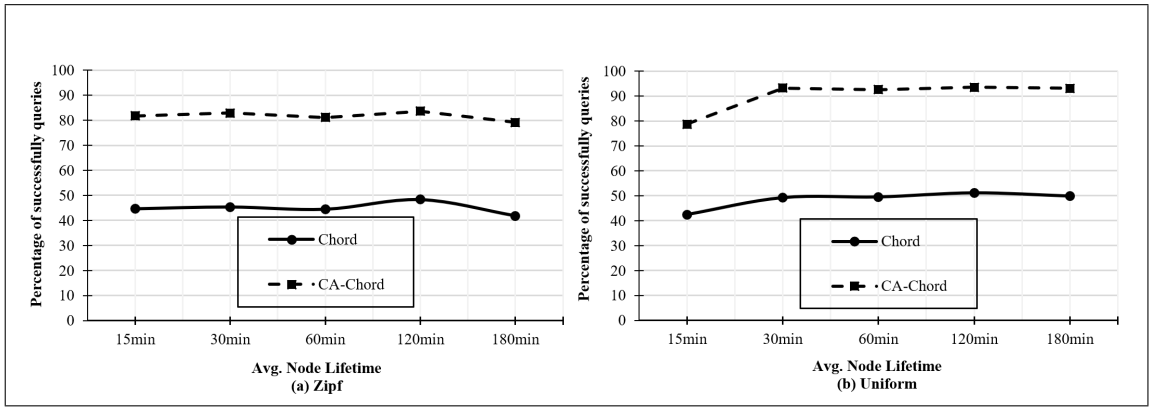


Figure 3. Percentage of successfully queries for varying rates of churn

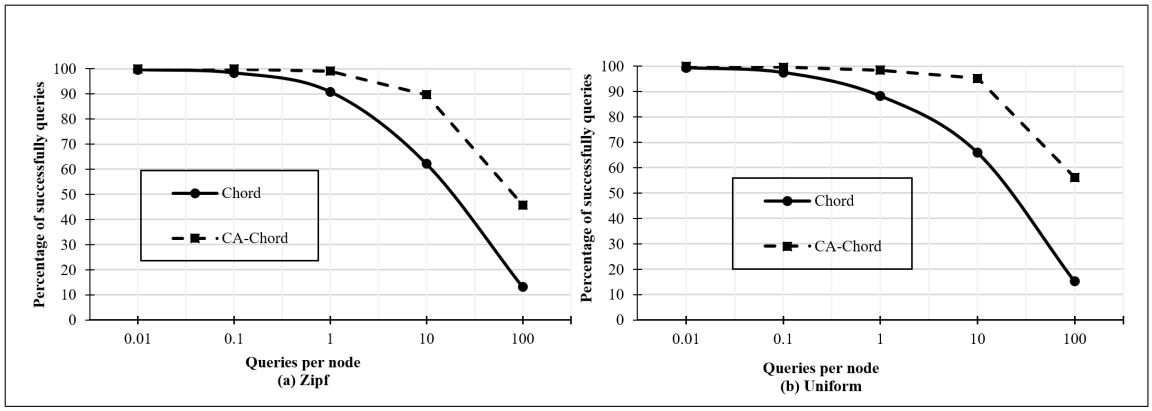


Figure 4. Percentage of successfully queries with varying loads

Zipf queries.

The second experiment studies the success rate of data queries when we change the average number of queries set in a node. The average lifetimes of a node is kept constant for 1 hour in the experiment. The parameter T of a node is set to 50% of the node's routing capacity. Query keys are generated according to the Uniform and Zipf probability distributions with $\alpha = 0.8$.

Experimental results are depicted in Figure 4. When the number of queries per node is low, the number of congested nodes in the network is small. Therefore, the success rate of both routing algorithms is near 100%. When the number of queries per node increases (i.e. there exists overloaded nodes in the network) the conventional Chord algorithm has no mechanism for balancing processing, therefore the success rate decreases substantially. Our algorithm chooses another route to forward the query to the destination, therefore it increases the success rate of the queries. With the average number of queries set in a node of 100, the success rate of our algorithm is 32% and 41% better than Chord algorithm for Zipf and Uniform queries, respectively.

The third experiment was conducted to evaluate the effect of soft congestion thresholds on the success rate of queries. The number of queries set in a node being 20 queries per second, and the

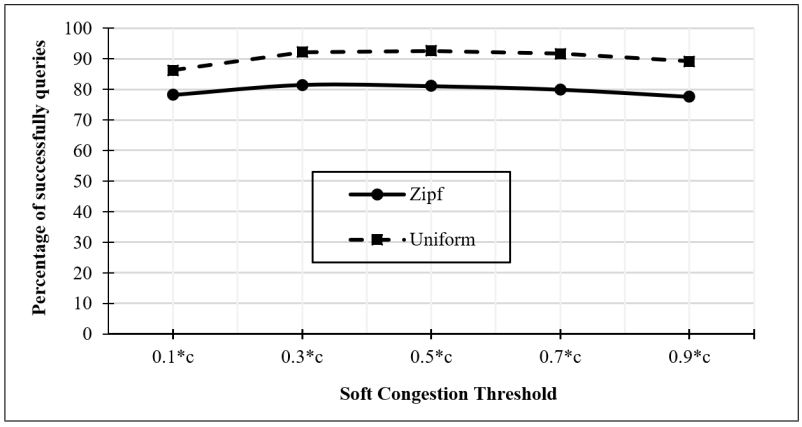


Figure 5. Percentage of successful queries with a varying soft congestion threshold

average lifetime of a node being one hour. The T parameter of a node was changed to handle bottlenecks in a node. Query execution takes two forms: Uniform and Zipf distributions with $\alpha = 0.8$. The results of the experiment are shown in Figure 5.

When the T parameter of a node is set at a low value (lower than 30% routing capacity of a node) and high value (greater than 70% routing capacity of a node), the success rate of the queries is lowered when the T parameter of a node is between 30% and 70% of the routing capacity of a node. When the T parameter is set to a low value, nodes will switch to early congestion processing state, affecting the performance of the system. If the T parameter is set to a high value, the node will be in the late congestion processing state. In this case, the number of deleted packets increases. The CA-Chord achieves the best performance when the value of T parameter is 50%.

The fourth experiment was conducted to evaluate the responsive ability of algorithms to the Zipf queries. The number of queries set in a node being 20 queries per second, the T parameter of a node being 50% of a node's routing capacity, the lifetime of a node being one hour, and changed the Zipf parameter of the query with the values α of 0.8, 1.2, 2.4 and 4.8 respectively. The results of the experiment are shown in Figure 6.

The results show that our algorithm performs better than the regular Chord algorithm on Zipf queries, with an average response rate of about 20% higher. When the parameter α is low ($\alpha = 0.8$ and $\alpha = 1.2$), the keys are equally distributed. Therefore, the conventional Chord and the CAChord algorithms achieve a high success rate. However, when the parameter α is high, the popularity of some keys increases. In this case, there are some keys that are frequently queried, while other keys are less frequently queried. Therefore, there exists the bottleneck in the nodes where they cannot process the queries, leading to a low query success rate.

4.2. Evaluation of query hop count

In this section presents our evaluation on the hop count needed to forward a query. The average lifetime of a node being one hour, the T parameter of a node being 50% of a node's routing capacity, and changed the number of queries set in a node to 0.01, 01, 1, 10, 100 queries per second. Query execution takes two forms: Uniform and Zipf distributions with $\alpha = 0.8$. The results are shown in

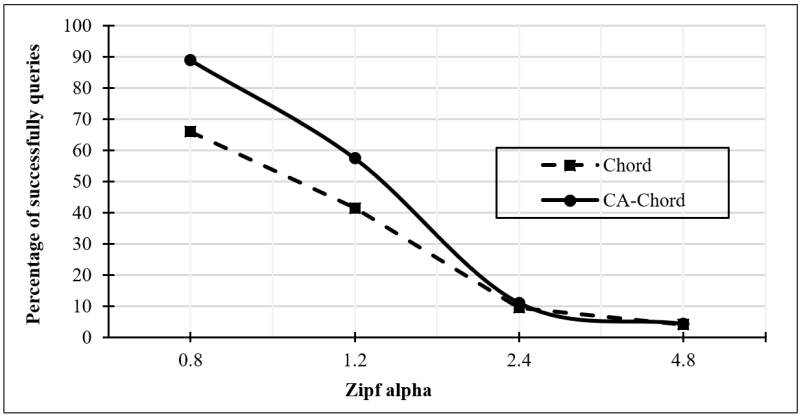


Figure 6. The effect of Zipf parameters on the query success rate

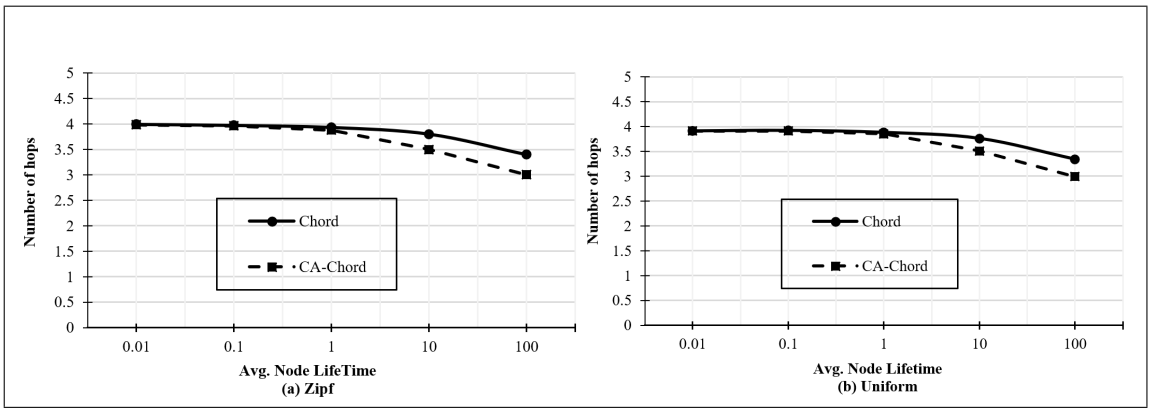


Figure 7. The effect of the number of queries on the number of steps for forwarding the query

Figure 7.

When the number of queries set in each node is low, nodes in the network are not congested and the CA-Chord processes the queries similarly to the conventional Chord algorithm. The average number of steps to forward a query of the two algorithms is the same. When the number of queries increases, some nodes in the network are in the congestion status. The CA-Chord algorithm replaces the congested nodes by the non-congested nodes close to the key management node in the query path, therefore decreasing the number of hops to forward the query to the destination. Our algorithm performs slightly better than the conventional Chord algorithm.

Second, we fixed the number of queries set in a node being 20 queries per second, the T parameter of a node being 50% of a node's routing capacity, and changed the node's lifetime. Query execution takes two forms: Uniform and Zipf distributions with $\alpha = 0.8$. The results of the experiment are shown in Figure 8.

When the lifetime of a node increases, i.e. the stability of the network increases, the number of steps for forwarding a query decreases. In both Zipf and Uniform queries, the number of steps to forward the query in our algorithm is smaller than that of the standard Chord algorithm.

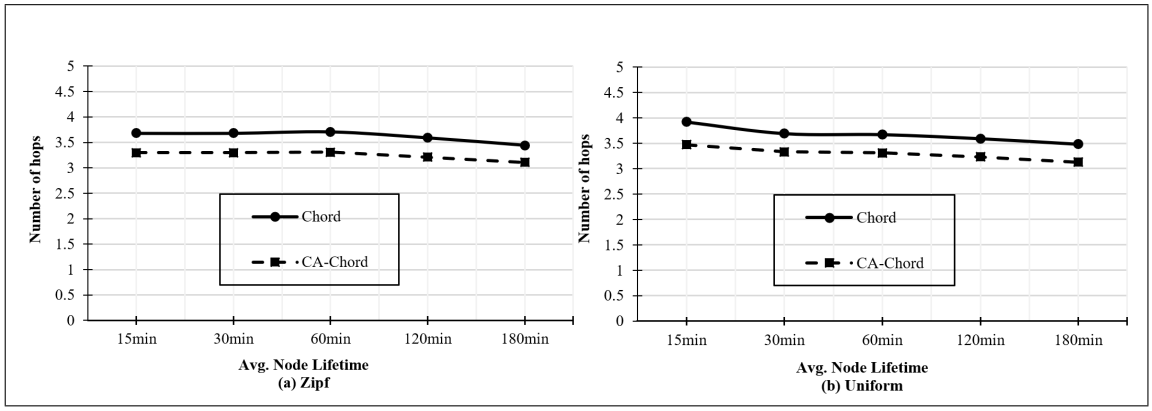


Figure 8. The effect of a node's lifetime on the number of steps to forward the query

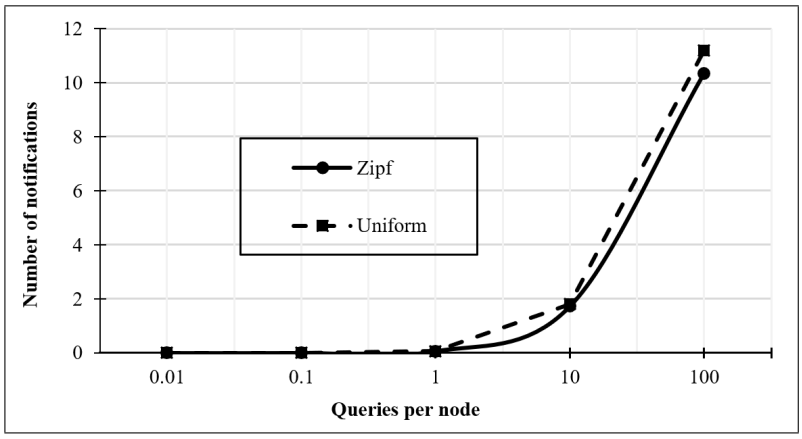


Figure 9. The effect of the number of queries on the number of congestion notifications

4.3. Evaluation of congestion notification message number

The next experiment was conducted to evaluate the number of messages sent by a node in case of congestion. We fixed the T parameter of a node being 50% of a node's routing capacity, the node lifetime being one hour and the number of queries set in a node to 0.01, 01, 1, 10, 100 queries per second. Query execution takes two forms: Uniform and Zipf distributions with $\alpha = 0.8$. The results of the experiment are shown in Figure 9.

The experimental results show that when the number of query messages set in a node increases, the number of congested nodes in the network increases, therefore the number of congestion notifications increases as well. The number of congestion notifications of Zipf queries is higher than that number of Uniform queries.

5. CONCLUSIONS AND FUTURE RESEARCH

We presented a new solution for congestion avoidance in structured peer-to-peer networks. The proposed solution brings better query success rates and limits the amount of traffic within the system. It thereby demonstrates the ability to resolve local congestion and increase the throughput in the network.

The advantage of our proposed solution is its simplicity in controlling congestion in Chord networks. Changing routing avoids bottlenecks, therefore increases the likelihood of success in query execution. Those changes affect only a small number of nodes (the congested nodes and the source nodes), therefore the number of packets and the secondary information needed to perform the changes and recovery processes is not large. Furthermore, selecting a node to replace a congested node as described above does not increase the number of nodes that each query must go through.

Our proposed method was evaluated by simulations of a P2P network that is similar to real-world networks where the capacities of nodes are not identical, the queries set in nodes are uneven and the lifetime of nodes are not similar. The evaluation results show that our proposed solution performs better than the conventional Chord algorithm.

However, our method still limits the replacement nodes to the adjacencies of a congested node without considering the node's responsive capacity when selecting the replacement node. In addition, the changes may continue if the new destination node is congested as well.

Therefore, if there is no mechanism to limit the number of changed routing table, the number of message packets may increase considerably when all the nodes of the network are in the congestion status. Moreover, changing the routing of packets cannot solve the congestion problem if the nodes of the network continue to push the query too quickly comparing to the network's responsiveness.

Despite some drawbacks, our approach can solve the problem of light congestion, especially in the case of local congestion, thereby increasing the responsive capacity of the network. Moreover, our approach can be combined with other congestion control methods using existing traffic control mechanisms to eliminate the possibility of network collapse when higher levels of congestion occur.

REFERENCES

[1] K. Aberer. P-Grid, "A self-organizing access structure for P2P information systems", *Sixth International Conference on Cooperative Information Systems*, 2001.

[2] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord, "A scalable peer-to-peer lookup service for internet applications", *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications*, San Diego, California, United States, SIGCOMM '01. ACM, New York, NY, 2001 (pp. 149–160).

[3] A. Rowstron and P. Druschel. 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] C. Tang and S. Dwarkadas, "Hybrid global-local indexing for efficient peer-to-peer information retrieval", *In NSDI*, 2004 (pages 211224).

[5] F. Klemm, J.-Y. Le Boudec, and K. Aberer, "Congestion control for distributed hash tables", *In The 5th IEEE International Symposium on Network Computing and Applications (IEEE NCA06)*, 2006.

[6] X. Shen, Q. Chang, L.Liu J. Panneerselvam, and Z. Zha: CCLBR, "Congestion control-based load balanced routing in unstructured P2P systems", *IEEE Systems Journal*, vol. 12, no. 1, 2018, 802–813.

[7] "Securehashstandard," *NIST, U.S. Dept. of Commerce, National Technical Information Service FIPS 180-1*, April 1995.

[8] F. Klemm, Jean-Yves Le Boudec, Dejan Kostic, and Karl Aberer, "Handling very large numbers of messages in distributed hash tables", *Proceeding COMSNETS'09 of the First International Conference on COMmunication Systems And NETworks*, 2009.

[9] F. Klemm, J.-Y. Le Boudec, D. Kostic, and K. Aberer, "Improving the throughput of distributed hash tables using congestion-aware routing", *In International Workshop on Peer-to-Peer Systems (IPTPS), Ecole Polytechnique Federale de Lausanne (EPFL)*, Lausanne, Switzerland, 2007.

[10] Z. Rehman, N. Shah, H. Rehman, and S. Kashan, "Implementation of DHT-Based Routing in Smart Grid", *International Journal of Open Information Technologies, ISSN: 2307-8162*, vol. 6, no.1, 2018.

[11] Q. He, Q. Dong, B. Zhao, Y. Wang, and B. Qiang, "P2P traffic optimization based on congestion distance and DHT", *Journal of Internet Services and Information Security (JISIS)*, vol. 6, no. 2, pp. 53–69, May 2016.

[12] J. Ledlie, and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity, and churn", *In Proceedings of 24th Annual Joint Conference of the IEEE Computer and Comunications Societies, INFOCOM 2005*, vol. 2, pp 1419-1430, March 13-17, 2005.

[13] F. Bustamante and Y. Qiao. Friendships that last, "Peer lifespan and its role in P2P protocols", *In Eighth International Workshop on Web Content Caching and Distribution, Hawthorne, NY*, October 2003.

Received on July 06, 2018
Revised on August 21, 2018