Research on Cache Strategy Based on Node Weight in Self-Organizing Network Environment

Yongqiong Zhu[a]

School of Art, Wuhan Business University, Wuhan, China

[a]zyqzhuyongqiong@126.com

Keywords: Cache Strategy, Self-Organizing Network, RREQ

Abstract: One of the core issues of self-organizing networks is how quickly the network can respond to user resource requests while reducing network bandwidth consumption and ensuring system scalability. This paper presents a caching strategy based on node weights for self-organizing networks. In this algorithm, each node records the information of the cache node and the weight of the importance. Based on the weight, the node can use the node with higher weight as its direct neighbour to guide the query to quickly find the target resource. Experiments show that the algorithm has significantly improves the performance of resource search.

1. Introduction

Mobile Ad Hoc Network (MANET) [1] is a network of automatically created, automatically organized, and self-managed, consisting of a set of cooperating autonomous wireless nodes or terminals. It does not require a fixed infrastructure and uses distributed management. One of the core issues of self-organizing networks is how quickly the network can respond to user resource requests while reducing network bandwidth consumption and ensuring system scalability.

AODV [2] is a routing protocol applied to adaptive network routing, which can implement unicast routing and multicast routing. AODV is a common protocol for generating routing patterns as needed in Ad Hoc networks. The protocol is defined as: When a node needs to transmit information to other nodes in the network, if no route arrives at the destination node, then a RREQ (routing request) message must be sent in the form of multicast. The RREQ message records the network layer address of the initial node and the target node. After receiving the RREQ, the neighboring node determines whether the target node is itself. If it is, send the RREP (routing response) to the originating node; if not, it first finds in the routing table for the route to the destination node, and then, forwards the unicast RREP to the source node, otherwise it continues to forward RREQ find it. This kind of forwarding is flooded, and the routing of messages is very blind, resulting in a large number of inefficient redundancy requests, which severely limits the scalability of the MANET system. To this end, the MANET topology needs to be optimized to improve search efficiency and reduce network overhead.

This paper presents a caching strategy based on node weights for self-organizing networks. In this algorithm, each node records the information of the cache node and the weight of its importance. Based on the weight, the node can use the node with high weight as its direct neighbor to guide the query to quickly find the target resource. Experiments show that the algorithm has significantly improves the performance of resource search.

Section 1 of this paper gives the background of the paper, Section 2 gives the current state of the study, Section 3 performs the cache optimization strategy and simulation experiments, and Section 4 gives the conclusion.

2. Related work

In response to the shortcomings of self-organizing networks, researchers have proposed a variety of topology optimization methods to improve the efficiency of query forwarding. Literature [3]

proposed a two-way overlay network optimization method SBO, which allows each node to collect its own 2-hop neighbor information, compare and filter the communication delays of all 2-hop neighbors, and gradually optimize the overlay network. The literature [4] proposes the topology optimization algorithm can ensure that the expected communication delay between any two nodes in the network is a constant. Both methods are to solve the problem that the topology of the application layer overlay network and the topology of the underlying physical network are inconsistent. For the free-riding phenomenon that exists widely in self-organizing networks, the literature [5] proposes an adaptive topology evolution protocol based on the credibility of nodes. This makes high-trust nodes occupy favourable positions and low-trust nodes are in an unfavorable position. Thereby, it is possible to effectively suppress malicious nodes and improve user satisfaction.

Since the content of the request of the nodes in the self-organizing network has certain regularity and is not completely random, there are a large number of blind and inefficient requests in the system, which increases the load of the system and wastes the resources of the system. Based on the above reasons, we can optimize the node cache table and use the objective rules of the requested content in the system to provide sufficient support for more efficient target search. Reference [6] classifies nodes according to their interests, and limits the scope of the query to nodes with the same interests. The SOSPNET proposed in [7] can optimize the search according to the semantic similarity of the shared files of nodes, and achieve load balancing and fault tolerance. This paper proposes an adaptive cache optimization strategy. During the search process, the node's cache table is dynamically adjusted according to the weight of the node, and the low-level nodes can reach the height node faster. The design principle of this paper is that if a node has a heavy weight to another node, indicating that this node contributes a lot in the most recent query cycle, then they are likely to become neighbors. The characteristics of this strategy are:

1) Support semantic-based dynamic connection selection strategy, nodes can dynamically adjust neighbor relationships based on query history and mutual interest preferences

2) Nodes make full use of their own capabilities, so that high-capacity nodes have high degrees, and the number of steps from low-capacity nodes to high-capacity nodes is as short as possible.

3. Caching strategy based on node weight

3.1 Model definition and representation

Definition 1 Self-organizing network: Self-organizing network is represented as an undirected graph $G=(V,E)$. Where V is the set of nodes in the network and E is the set of edges. For any node in the network $u \in U, v \in V$, if $(u,v) \in E$,then $(v,u) \in E$. For any node $u \in V$, Its neighbor collection is recorded as $N_u = \{v \mid (u,v) \in E\}$.

Definition 2 The weight of the node: The weight of a node is determined according to the number of resources that the forwarding node can obtain. Assuming that node v is a neighbor of node u, the greater the number of resources obtained by node v forwarding, the less the number of search hops, the higher the weight of node v to u.

$$W(u,v) = \sum_{v=1\cdots ttl} \frac{F^{v-1}}{N_u[v]} \tag{1}$$

$N_u[v]$ is the number of hops that the neighbor node v of the node u forwards. F^{v-1} is the file found by the $v-1$-hop of the query to the node. Ttl is the message lifetime.

Each node cache stores information about all hit nodes that initiate queries from the node, including node IDs and weights. The identifier field is used to determine whether the node is a neighbor node, and the weight represents the degree of positive feedback recently obtained by the node. The greater the weight of the node hit, the more likely it is to forward directly. Cache tables are sorted by weight. The size of the cache table for each node is fixed. The maximum number of entries is M. A replacement policy is required when the cache is full.

Definition 3 The maximum connection degree L of the node: Assuming that the capacity of node P_a is C_a, the average bandwidth required to establish a connection is W, then the maximum number of connections of the node is

$$L = \frac{C_a}{W} \tag{2}$$

Once the cache list is full, you need to delete the old cache node for replacement. The weight represents the importance of the cache node. The more important the cache node is, the less important it is to be replaced by the new cache node.

3.2 Cache table update

Since the cache table stores the information of the hit node, whenever a node hits, it needs to be inserted into the cache table for updating. When the node first appears in the cache table, you need to record the node ID and weight. If the node already exists in the cache table, you only need to update its weight. The weight of the node is related to the number of hits of the node, and can dynamically reflect the importance of the node relative to the query initiation node in the current cycle, and the nodes with larger weights are more similar. The cache weights of nodes are defined as follows:

Definition 4 The cache weight of the node: The cache weight of a node is related to the cache weight of the node in the previous cycle and the number of hits in the current cycle.

$$W_T(u, p_a) = \alpha \times W_{T-1}(u, p_a) + \beta \times \frac{Hit(p_a)}{\sum_T QueryNum_u} \tag{3}$$

Where $W_{T-1}(u, p_a)$ represents the weight value of the node in the last week T-1, $\sum_T QueryNum_u$ is the number of queries initiated by the node u in the current period, and $Hit(p_a)$ represents the number of queries initiated by u in the current period T. α, β is the parameter that satisfies $\alpha + \beta = 1$. The value of α is related to the frequency of interest transfer of the node and the network status. When the network is very dynamic, α takes a value between (0.5, 1), otherwise it takes a value between (0, 0.5).

If the entry in the node cache table has not reached its upper limit L, it is inserted directly into the cache node.If the capacity of the cache node has reached its upper limit, you will need to replace the connection with the replacement strategy below.

3.3 Replacement connection

Once the node's total cache reaches its maximum connection limit, the node must replace an old cache when adding a new cache. At this time, the selection of the nodes in the cache table cannot consider only one party's interests, but it is necessary to consider whether adding such a connection to the two parties is more profitable than the connection to be replaced. If the newly added connection does bring more benefits to both parties, you can create this connection to replace the old one, otherwise you won't have to establish this connection. The global satisfaction of defining nodes is as follows:

Definition 5 Global satisfaction of the node:

$$SatAll(u, P_k, v) = \frac{W(u, P_k) + W(P_k, v)}{2} \tag{4}$$

Selecting the node P_k method to be established in the cache table of the node u is the same as described in the previous section, and selecting the neighbor node to be deleted is selecting a node whose weight value is smaller than C_Threshold and having the smallest weight in the neighbor node list, and is assumed to be v. To replace the old connection $u \to v$ with the new connection

$u \to p_k$, you must ensure that the node P_k is added to the node u, v, and the benefit from the original connection is greater than that of the original connection.

$$SatAll(u, P_k, v) > SatAll(u, v, P_k) \tag{5}$$

If the node with the smallest weight in the neighbor list does not satisfy the condition, the node with the second smallest weight is selected to judge. If a node that satisfies the condition is not found, then the newly established connection does not bring more benefits and does not have to establish a new connection.

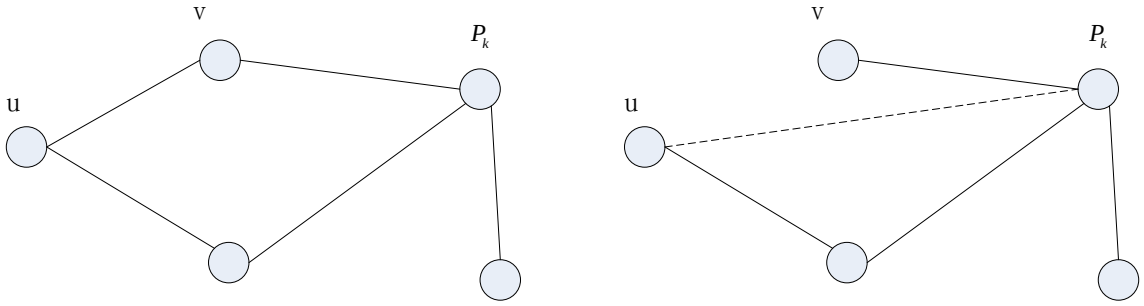

Figure 1. Replacement connection of nodes

3.4 Algorithm

Below we give the strategy algorithm.

Cache-based Algorithm

The current node is assumed to be u .
1. Select the non-neighbor node p_k that has the largest weight exceeding the threshold C_Threshold from the cache list of the node;
2. If the total number of current neighbors of node u does not exceed the upper limit of connection L, go to 3, otherwise go to 4;
3. If node p_k is not in the neighbor list, p_k is inserted into the neighbor list as a direct neighbor of node u;
4. A node v having the lowest weight less than the threshold C_Threshold is selected from the cache table;
5. Calculation formula 5;
6. Replace node p_k with node v if the condition is met;
7. Otherwise, look for the next node with the lowest weight, go to 5;
8. If a node that satisfies the condition is not found, the replacement operation is not performed and the algorithm terminates.

3.5 Simulation

This experiment uses Matlab to achieve simulation. Set the size of the network node to 100. Set up TTL is 20. Set 20% of the nodes to share 80% of the files, 80% of the nodes share 20% of the files, which indicates the node's contribution to the system imbalance. We set up several rounds of experiments for the experiment. Each node randomly sends out 1 to 20 search requests from each node, and the search target file is selected from its neighbors and cache set with the same probability. During the period, the node collects various types of information required by the algorithm. When all the searches are completed, each node first performs an algorithm for adding connections, and then performs an algorithm for deleting the connection. Although this experimental method is not possible in the actual application environment due to the lack of an effective synchronization mechanism between nodes, it can fully reflect the effect produced by this algorithm.

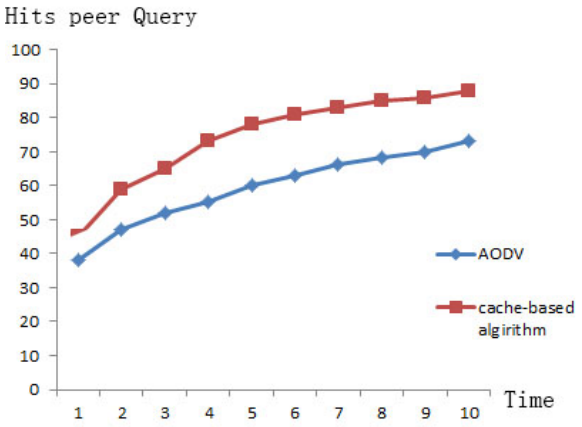

Figure 2. Comparison of the hits peer query.

As can be seen from Fig. 2, cache-based algorithm is increased by about 10% compared with AODV.

4. Conclusion

Compared with AODV, the cache-based topology optimization scheme established by this algorithm has better retrieval efficiency. However, the cache also needs to consume a certain amount of network capacity, and further improvements can be made.

Acknowledgements

This work was financially supported by fund of the Science Research Project of Wuhan Business University (Grant No. 2015KA011) -Research on the Current Situation and Development Strategy of Chinese 3D Animation Films.

References

[1] DAVID R, IGNAS G N. Ad hoc networking in future wireless communications [J]. Computer Communications, 2003 (1): 36-40.

[2] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc OnDemand Distance Vector (AODV) Routing. RFC 3561, July 2003.

[3] Liu, Y., X. Li, and L.M. Ni, *Building a Scalable Bipartite P2P Overlay Network.* Parallel and Distributed Systems, IEEE Transactions on, 2007. **18** (9): p. 1296-1306.

[4] Hung-Chang, H., L. Hao, and H. Cheng-Chyun, *Resolving the Topology Mismatch Problem in Unstructured Peer-to-Peer Networks.* Parallel and Distributed Systems, IEEE Transactions on, 2009. **20** (11): p. 1668-1681.

[5] zhangqian, An Effective Peer-to-Peer Adaptive Topological Evolution Co-operation. Journal of software, 2007.

[6] Xin-Mao, H., C. Cheng-Yue, and C. Ming-Syan, *PeerCluster: A Cluster-Based Peer-to-Peer System.* Parallel and Distributed Systems, IEEE Transactions on, 2006. **17** (10): p. 1110-1123.

[7] Garbacki, P., D.H.J. Epema, and M. van Steen, *The Design and Evaluation of a Self-Organizing Superpeer Network.* Computers, IEEE Transactions on, 2010. **59** (3): p. 317-331.