# Detecting and Reducing Partition Nodes in Limited-routing-hop Overlay Networks

Zhenhua Li, Guihai Chen
State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, 210093, P. R. China
lizhenhua@dislab.nju.edu.cn, gchen@nju.edu.cn

## Abstract

*Many Internet applications use overlay networks as their basic facilities, like resource sharing, collaborative computing, and so on. Considering the communication cost, most overlay networks set limited hops for routing messages, so as to restrain routing within a certain scope. In this paper we describe* partition nodes *in such limited-routing-hop overlay networks, whose failure may potentially lead the overlay topology to be partitioned so that seriously affect its performance. We propose a proactive, distributed method to detect partition nodes and then reduce them by changing them into normal nodes. The results of simulations on both real-trace and generated topologies, scaling from 500 to 10000 nodes, show that our method can effectively detect and reduce partition nodes and improve the connectivity and fault tolerance of overlay networks.*

## 1 Introduction

In the past few years, overlay networks have fast developed into the base infrastructures of many Internet applications. They have been widely applied to many fields, like resource sharing [1, 2, 6], multimedia streaming [7], collaborative working [5], distributed computing [4], and so on. An overlay network is usually an application-level logical network which works on top of another underlying network. The most representative utility of overlay networks may be Peer-to-Peer systems, which commonly built their overlay networks on top of the worldwide IP network.

Nodes of the overlay network are connected by logical links, each of which corresponds to a path perhaps composed of many underlying links in its underlying network. Therefore, routing one hop in the overlay network is often much more *expensive* than that in the underlying network. Besides, many overlay networks use flooding or flooding-based routing mechanism, with which one more hop will bring on tremendous extra messages. Considering the communication cost, most overlay networks set limited hops (or says $TTL$) for routing messages, so as to restrain them within a certain scope.

Although nodes in overlay networks are usually viewed as equal entities in function, in fact some of them have special meaning to the overlay topology. Nodes which behave as the single route way of other nodes play a critical role in the overlay topology. Their failures may lead the overlay to be partitioned and seriously affect its performance. We call the above-mentioned special nodes *topologically-critical* nodes. If these nodes can be effectively detected and then transformed to normal nodes, the possibility of overlay partitioning will be greatly reduced when they fail.

> *The fundamental problem is: which nodes are topologically-critical in limited-routing-hop overlay networks?*

If we regard the overlay network as an undirected graph, it is natural to think of *cut node* (or cut vertex), which can be defined as follows: node $C$ is a cut node of graph $G$ ($G$ is connected) if $G$ would be partitioned into two or more isolated connected components when $C$ is removed. However, cut node is a global and static concept, while a limited-routing-hop overlay network is marked by its routing localization and dynamic working environment. In this paper we suggest the concept of *partition node* to describe the topologically-critical nodes of overlay networks. This concept is an extension of cut node to make it comply with the characteristics of limited routing hops, which will be explained in Section 3. Figure 1 demonstrates the relations between cut node and partition node.

Figure 2 shows partition nodes' significance to seven generated overlay topologies scaling from 500 to 1500 nodes. For the same topology, compared to random node failures, much fewer partition nodes' failures would cause overlay partitioning.

Based on the partition node concept, we propose a proactive method to detect partition nodes and then reduce them

The Sixth International Conference on Grid and Cooperative Computing(GCC 2007)
0-7695-2871-6/07 $25.00 © 2007 IEEE

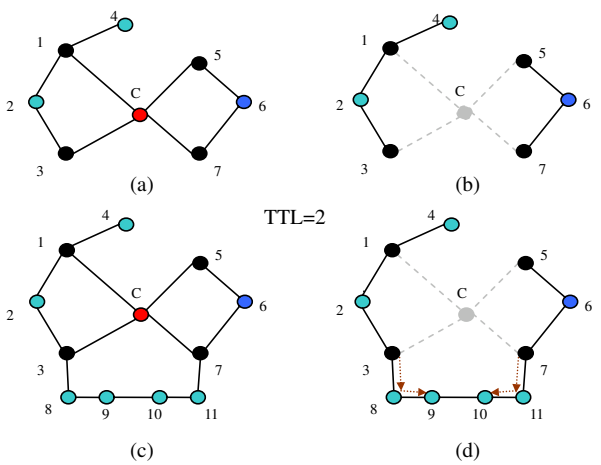


**Figure 1. Cut Node vs. Partition Node. (a) $C$ is a cut node because (b) when $C$ fails, the overlay network is partitioned; (c) $C$ is a partition node because (d) when $C$ fails, the overlay network is not partitioned, but $C$'s neighbors $1, 3, 5, 7$ can no longer find each other.**

by changing them into normal nodes. Our method needs no centralized control, and every node runs its own detection and reduction algorithm independently. For 10 real-trace overlay topologies each with 10000 nodes, our method can reduce the number of partition nodes to only $1\% \sim 3\%$.

The rest of this paper is organized as follows. In Section 2, we overview related work. Section 3 explains partition node concept. Section 4 describes partition node detection process and Section 5 presents partition node reduction. In Section 6, we evaluate the performance of our method by simulation. Finally, we conclude the paper and point out future work in Section 7.

## 2 Related Work

The overlay partitioning problem has been receiving wide attention for several years. Saroiu et al. [15] indicated that the popular unstructured P2P network, Gnutella, accords with Power Law distribution and it has high fault tolerance to random node failures. However, the failures of small amounts of high-degree nodes can efficiently partition the overlay of Gnutella [3], which makes it vulnerable in face of well-constructed, targeted attacks. Although [15] has realized the importance of high-degree nodes to overlay topology, it fails to answer which part of the high-degree nodes are topologically-critical, and whose failure may cause overlay partitioning. We consider partition nodes to be the right answer.

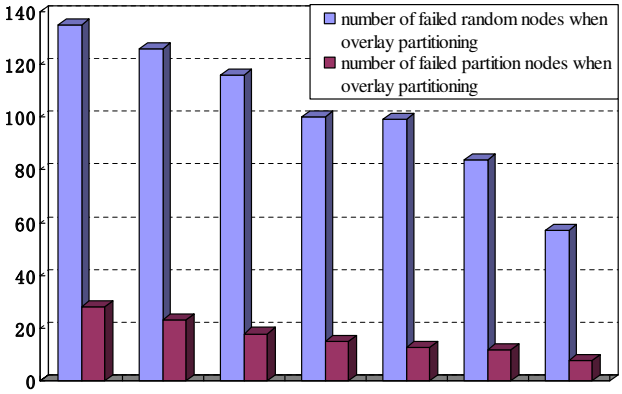


**Figure 2. Partition nodes' significance to overlay topology. Compared to random node failures, much fewer partition nodes' failures would cause overlay partitioning.**

Various methods have been proposed to alleviate overlay partitioning problem. We categorize them from two orthogonal aspects: (1)*proactive avoid* or *reactive recover*, (2)*event driven* or *periodical detect*. Proactive avoid means trying to avoid the possible occurrence of overlay partitioning, while reactive recover means to repair the overlay topology by incorporating partitions when it has been found partitioned. Event driven means operations are performed only when some prescribed events happen, e.g. nodes' arrival and departure, while periodical detect means operations are performed periodically.

### 2.1 Proactive avoid and Event driven

Pandurangan et al. [12] studied the problem of maintaining a connected overlay network as nodes join and depart according to a Poisson process. By using a central server to direct new joins to specific nodes in the network, and to update old nodes' neighbors as nodes depart, they are able to maintain the topology connectivity using only a constant amount of space per node. This centralized method requires all nodes to join and depart by notifying the central server, allowing maintenance operations to happen only at the time of arrival and departure.

### 2.2 Proactive avoid and Periodical detect

Liu et al. [10] designed a distributed mechanism called *CAM* to alleviate overlay partitioning by actively detecting cut nodes before they fail then neutralizing them into normal overlay nodes, so that the possibility of overlay partitioning is minimized after they fail. CAM mechanism efficiently discovers the routing reachability between nodes

without repeated probe work, and thus reduces the traffic cost to a low level. Taking all its strengths into consideration, in this paper we base on the probing method of CAM to find the routing reachability between nodes.

### 2.3 Reactive recover and Event driven

Many overlay networks work based on the ring-like topology, so if the ring is partitioned into several isolated sub-rings, they cannot work correctly. Mahajan et al. [11] proposed a scheme to detect and repair the ring partitions and implement their scheme on Pastry [14]. Harvey et al. [9] study the ring partitioning problem of SkipNet [8], but their solution can only be used in SkipNet, a special network which provides controlled data placement and guaranteed routing locality.

### 2.4 Reactive recover and Periodical detect

Sit and Morris [16] proposed a *cross-check* method to alleviate overlay partitioning. By asking other nodes to do random queries and comparing their results with its own, a node can verify whether its view of the network is consistent with others'. If their views are different, it is possible that the overlay has partitioned, then this node must take measures to recover the overlay. The main flaw of cross-check method lies in the randomness and uncertainty of its detection.

## 3 Partition Node Concept

We first define two relations in limited-routing-hop overlay networks: *locatability* and *reachability*, then define partition node based on the two relations.

**Definition 1 (Locatability)** *In a limited-routing-hop overlay network, node $A$ could **locate** node $B$ only if $A$ can find $B$ by sending routing messages. It is denoted by $A \rightarrow B$.*

Locatability relation is not transitive, and is not symmetric unless the topology is undirected.

**Definition 2 (Reachability)** *In a limited-routing-hop overlay network, node $A$ could **reach** node $C$ if $A$ can locate $C$, or $A$ can locate some node $B$ and $B$ can locate $C$. It is denoted by $A \rightarrow\rightarrow C$.*

Reachability relation is not transitive and is not symmetric unless the topology is undirected. Reachability is the foundation of partition node detection: whether a node $n$ is a partition node depends on whether its neighbors can still reach each other when $n$ fails. Figure 3 shows an example of locatability and reachability.

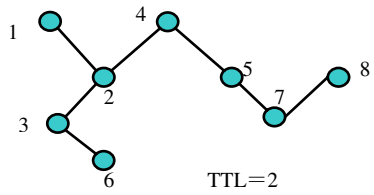


**Figure 3. Node 1 can only locate nodes 2, 3, 4, and can reach node 5, 6, 7, but cannot reach node 8.**

**Definition 3 (Partition Node)** *In a limited-routing-hop overlay network, node $C$ is a **partition node** if $C$'s neighbor set would be partitioned into two or more unreachable subsets $S_1$, $S_2$, ..., $S_n$ ($n \geq 2$) when $C$ fails. Only nodes in the same subset can reach each other. Figure 4 shows an example of partition node.*

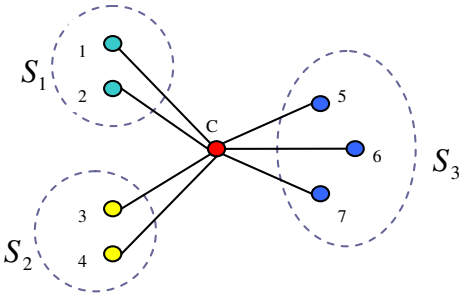


**Figure 4. Node $C$ is a partition node. Its neighbor set would be partitioned into three unreachable subsets $S_1$, $S_2$ and $S_3$ when $C$ fails.**

From the definition above, we can see that partition nodes are the critical but vulnerable elements of the overlay topology, whose failure may potentially lead the overlay topology to be partitioned so that seriously affect its performance. So if partition nodes can be effectively detected and then changed to normal nodes, the possibility of overlay partitioning will be greatly reduced when they fail.

## 4 Partition Node Detection

Every node runs the detection process independently to decide whether itself is a partition node. For a node $C$, it first sends probe messages to collect reachability information between its neighbors, and then divides its neighbor set into several unreachable subsets. If it has only one such

subset, $C$ is not a partition node; otherwise it is. The steps are as follows:

1. **Initialize Detection**
   To initialize the detection, $C$ sends a *Msg_Init* message to each of its neighbors $N_1, N_2, \ldots, N_n$. Each $N_i$ must return a *Msg_Response* message which contains $N_i$'s information such as IP address, degree, bandwidth and so on (see Figure 5(1)). The information above is necessary for later work (partition node reduction). If $C$ does not receive *Msg_Response* from $N_i$, it would consider $N_i$ to have failed and remove $N_i$ from its neighbor set.

2. **Probe Reachability**
   To probe reachability between its neighbors, $C$ sends a *Msg_Probe* message to each of its neighbors (see Figure 5(2a)). *Msg_Probe* contains $C$'s IP address, a $TTL$ limit and the neighbor's ID: $N_i$. Every node in the system maintains a connection list, in which every candidate has an entry like { candidate's address, neighbor 1's ID, neighbor 2's ID, ...}. The candidate $C$'s entry in node $N$'s connection list means which of $C$'s neighbors can reach each other. When a node receives a *Msg_Probe* and the neighbor's ID in *Msg_Probe* is new, it will forward this message to all its neighbors except the message sender (see Figure 5(2b)), and add the neighbor's ID to the corresponding candidate entry in its connection list. After that, if this entry contains two or more neighbors, a *Msg_Arrival* message will be sent to the candidate to tell it which of its neighbors can reach each other (see Figure 5(3)). The probe process above is based on the probe method used in [10].

3. **Partition Subsets**
   Node $C$ gathers *Msg_Arrival* messages continuously to get reachability information between its neighbors. When the probe step has finished (or the timeout has expired), $C$ partitions its neighbors by assigning reachable neighbors into the same subset.

4. **Make Decision**
   When node $C$ finishes partitioning subsets, it makes decision whether itself is a partition node: if all its neighbors are in the same subset, $C$ is not a partition node; otherwise $C$ is and it will enter the next step — partition node reduction.

The partition node detection process is shown in Figure 5.

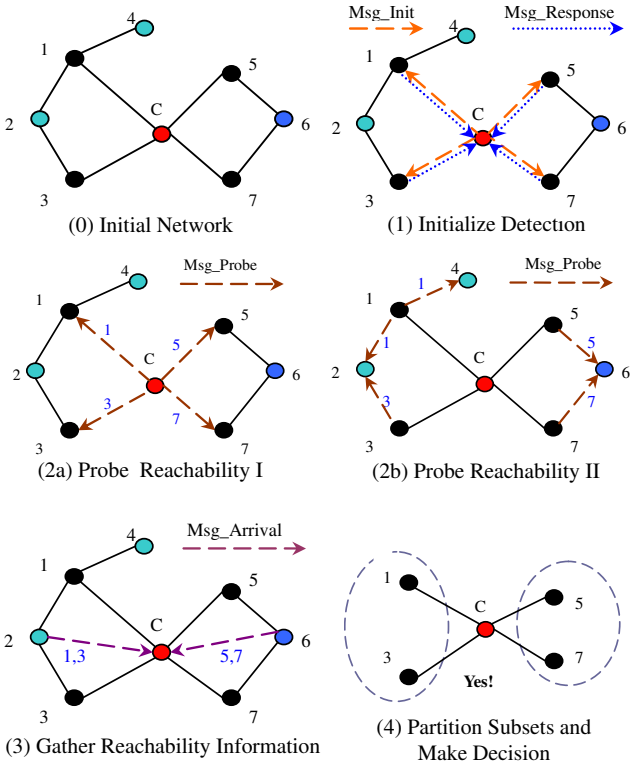


**Figure 5. Partition Node Detection. $C$ is the node who wants to decide whether itself is a partition node.**

## 5 Partition Node Reduction

### 5.1 Add edges to reduce partition nodes

After node $C$ has made sure that it is a partition node, in order to change itself into a normal node, it needs to add edges for its neighbors so as to incorporate unreachable subsets to a single one. Supposing $C$'s neighbors are partitioned into $S_1, S_2, \ldots, S_n$, $C$ will choose an appropriate *delegate node* $N_i$ from each subset $S_i$, and then connects all the delegate nodes in some way. To find appropriate delegate nodes, we consider the following criteria:

1. In order to improve the system's fault tolerance, we try to make every node's degree above a constant lower bound as much as possible. Ripeanu and Foster [13] have pointed out if every node in the overlay network can have a degree above a constant lower bound, resistance to malicious attacks can be greatly strengthened. In graph theory, for every graph $G$ there is a simple but important inequation illustrating the relations between *vertex-connectivity* $\kappa(G)$, *edge-connectivity* $\lambda(G)$ and *minimum-degree* $\delta(G)$ : $\kappa(G) \leq \lambda(G) \leq \delta(G)$, so we can see high vertex/edge-connectivity requires a large minimum vertex degree. We think the lower bound [13] mentioned is indeed the minimum vertex degree $\delta(G)$.

2. In order to improve load balance of the overlay

network, we make every node's *load factor* ($= \frac{node\_degree}{node\_capability}$) be the same as much as possible.

Based on the criteria above, the partition node $C$ first finds which node of subset $S_i$ has the smallest degree. If the smallest degree is lower than the constant lower bound $Min\_Degree$, $C$ takes this smallest-degree node as the delegate $N_i$ of subset $S_i$; otherwise, $C$ computes each node's load factor, and then takes the node with the lowest load factor as the delegate $N_i$.

When delegate nodes have been chosen, how to connect them? The simplest method is *Linear Chain Connection*: for each delegate node $N_i$, it is added an edge to $N_{(i+1) \mod n}$ (see Figure 6). However, linear chain connection is vulnerable. If some nodes fail later, this chain is likely to disconnect again, which will cause the emergence of new partition nodes. Therefore, we consider using a more complicated method *Chord Ring Connection*: for each delegate node $N_i$, it is added not only an edge to $N_{(i+1) \mod n}$, but also an edge to $N_{(i+n/2) \mod n}$ (see Figure 7). Simulation results show that the latter is usually better than the former in many aspects.

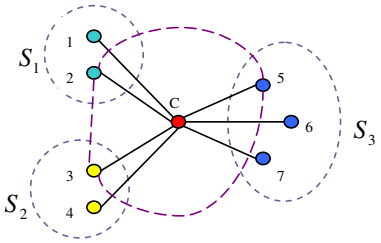


**Figure 6. Linear Chain Connection.**

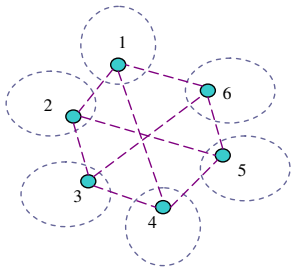


**Figure 7. Chord Ring Connection.**

### 5.2 Remove edges to limit node degree

Maintaining more connections usually causes much more cost of updating node state, so each node can only maintain a limited number of connections according to their capabilities. Therefore, when adding edges to a node causes its degree to exceed the upper bound, that node should remove some of its edges. Then which edge should be removed from that node? We use the following criteria:

1. The new edges added to reduce a partition node cannot be removed;

2. For the other edges, compute the load factor of each node they connect to, then remove the edge whose corresponding node has the highest load factor.

The reason for the criteria above is straightforward, so we omit it here.

### 5.3 Total cost of partition node detection and reduction

During the partition node detection, the traversal scopes of probe messages for different neighbors will not overlap (just $Msg\_Arrival$ messages are returned when they intersect), so it avoids repeated work for reachability probe. Considering an overlay network with $n$ nodes, let $c$ be the average node degree, and let $t$ be the $TTL$ threshold, the communication cost for each node's detection is $min(O(c^t), O(nc))$ (more detailed analysis can be found in [10]). The cost of partition node reduction is trivial compared to that of partition node detection, so finally the total cost of all nodes' detection and reduction is $min(O(nc^t), O(n^2c))$. In particular, if the average degree $c$ and the $TTL$ value $t$ are both small, the total cost would be $O(n)$, which is very low.

## 6 Performance Evaluation

### 6.1 Simulation Methodology

We perform simulations on 30 real-trace *sparse* overlay topologies and 30 generated *dense* topologies. The real-trace data was collected from Dec. 7th 2000 to June 15th 2001 on http://dss.clip2.com. The trace topologies we used scale from 500 to 10000 nodes, with average node degree from less than 1 to 3.5 (so-called *sparse*). The generated topologies scale from 500 to 2500 nodes, with average node degree from 5 to 7 (so-called *dense*). The $TTL$ threshold is set from 2 to 5 respectively, especially 3 and 4 always used. Every node is assigned its capability value randomly ranging from 1 to 20, which represents the number of connections it can burden, and the constant lower bound $Min\_Degree = 3$. To simulate a highly dynamic network environment, we let nodes continuously fail, and meanwhile perform partition node detection and reduction to see the effect.

### 6.2 Effectiveness of our method

To check the effectiveness of our method, we first measure the number of partition nodes in the initial network, and then perform partition node detection and reduction for many rounds to see how the number of partition nodes changes. A clear observation is that partition nodes are greatly reduced after the 1st and 2nd round, usually left only $1\% \sim 3\%$, then tend to be very few and stable. Figure 8 shows the simulation results on 5 real-trace topologies, each with 10000 nodes, $TTL$=4.

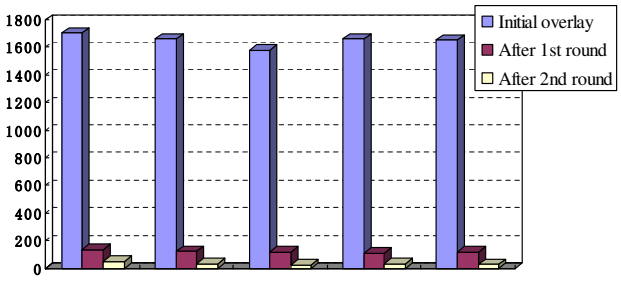


**Figure 8. Effectiveness of our method. Y-axis shows the number of partition nodes in initial network, after the 1st round and after the 2nd round.**

### 6.3 Enhancing resistance to overlay partitioning

To evaluate how the network's resistance to overlay partitioning is enhanced by reducing partitioning nodes, we first let nodes gradually fail with no recovery mechanism, then use the same initial topology and let the same nodes gradually fail, but every time when $T$ more nodes fail we perform partition node detection and reduction, each time measuring the number of failed nodes when overlay partitioned. Figure 9 shows the simulation result on several generated topologies, each with 1000 nodes, $TTL$=3, 4.

As $T$ increases, recovery is performed less frequently, and the overlay topology could not get repaired in time, so the number of failed nodes decreases when the overlay is partitioned. When $TTL$ increases, the number of partition nodes detected decreases, so the overlay topology gets less repaired.

### 6.4 Overlay's fault tolerance improvement

We measured the overlay network's query success rate under a dynamic environment in which nodes continuously fail. First we let nodes gradually fail and measure the query success rate every time $T$ more nodes fail. Then we use the same initial topology and let the same nodes gradually fail, but perform partition node detection and reduction every time $T$ more nodes fail, and do the same queries. The $TTL$ threshold for query is the same as that for detection.

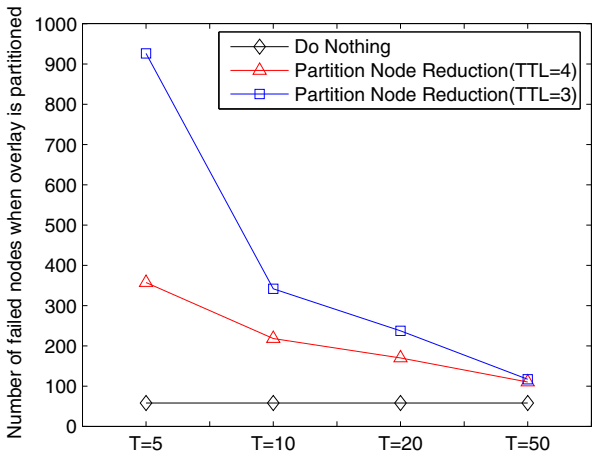


**Figure 9. Enhancing resistance to overlay partitioning. It shows the number of failed nodes when overlay is partitioned, with recovery $T = 5, 10, 20, 50$ and $TTL = 3, 4$.**

Figure 10 shows the simulation result on a generated topology with 1000 nodes, $TTL$=3. We can observe that with partition node reduction, query success rate is not only higher, but also more stable and would not decline sharply.

## 7 Conclusion and Future Work

In this paper, we investigate the overlay partitioning problem of limited-routing-hop overlay networks. We first suggest the concept of partition node whose failure may potentially lead the overlay topology to be partitioned, then propose a proactive, distributed method to detect partition nodes and reduce them at low cost. Simulation results prove the effectiveness of our method, and that our method can improve the connectivity and fault tolerance of overlay networks.

## References


[1] Bittorrent website. *http://www.bittorrent.com*.
[2] edonkey website. *http://www.edonkey.com*.
[3] Gnutella website. *http://rfc-gnutella.sourceforge.net*.
[4] Gpu website. *http://gpu.sourceforge.net*.
[5] Groove website. *http://www.groove.net*.
[6] Kazaa website. *http://www.kazaa.com*.

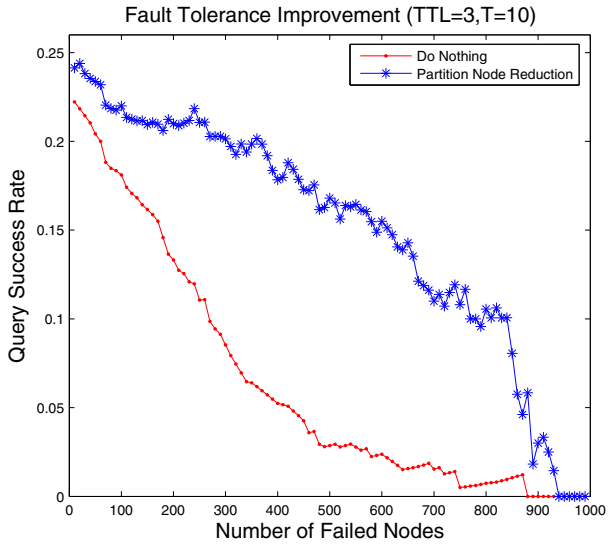


**Figure 10. Overlay's fault tolerance improvement. It shows the network's query success rate when nodes continuously fail.**


[7] Skype website. *http://www.skype.com*.

[8] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. *Proceedings of USITS*, 3, 2003.

[9] N. Harvey, M. Jones, M. Theimer, and A. Wolman. Efficient Recovery From Organizational Disconnects in SkipNet. *Second International Workshop on Peer-to-Peer Systems (IPTPS03)*, 2003.

[10] X. Liu, L. Xiao, A. Kreling, and Y. Liu. Optimizing Overlay Topology by Reducing Cut Vertices. *Proceedings of ACM NOSSDAV*, 2006.

[11] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. *IPTPS03*, pages 21–32, 2003.

[12] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 492–499, 2001.

[13] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. *Proceedings of International Conference on Peer-to-peer Computing*, 101, 2001.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 11:329–350, 2001.

[15] S. Saroiu, K. Gummadi, and S. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[16] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*.