On the Reliability of Large-Scale Distributed Systems - A Topological View

[1]Yuan He, [2]Hao Ren, [1]Yunhao Liu, and [3]Baijian Yang

[1]*Hong Kong University of Science and Technology*
{heyuan, liu}@cse.ust.hk

[2]*National University of Defense Technology*
renhao1973@gmail.com

[3]*Ball State University, U.S.A.*
byang@bsu.edu

Abstract

In large-scale, self-organized and distributed systems, such as peer-to-peer (P2P) overlays and wireless sensor networks (WSN), a small proportion of nodes are likely to be more critical to the system's reliability than the others. This paper focuses on detecting cut vertices so that we can either neutralize or protect these critical nodes. Detection of cut vertices is trivial if the global knowledge of the whole system is known but it is very challenging when the global knowledge is missing. In this paper, we propose a completely distributed scheme where every single node can determine whether it is a cut vertex or not. In addition, our design can also confine the detection overhead to a constant instead of being proportional to the size of a network. The correctness of this algorithm is theoretically proved and a number of performance measures are verified through trace driven simulations.

1. Introduction

Large-scale and distributed systems, such as Peer-to-peer (P2P) and wireless sensor networks (WSN) are proliferating due to their high flexibility, easy implementation, and high autonomy. However, such systems suffer from node failures because P2P nodes might leave the system arbitrarily, and sensor nodes may suddenly die from running out of energy or being attacked. Hence, providing reliable services is a very critical and challenging problem for those systems.

For a large-scale distributed system, it is often not cost-effective to protect all the nodes. Moreover, nodes in a distributed system are usually not equally important from the reliability point of view. For example, S. Saroiu et al. [1] observe that after randomly removing 30% of the nodes from a 1771 nodes Gnutella network, 1106 of the remaining 1300 nodes stay connected, and are likely to exchange and share information. In contrast, if 4% of carefully selected nodes are removed from the network, the overlay is fragmented into hundreds of pieces. Other researchers also notice the existence of critical nodes or critical links in the context of P2P or sensor network systems [2, 3, 4].

Figure 1(a) illustrates a topology representing a part of a P2P overlay or a WSN. The dashed arrows depict the data/information flow. Clearly, the failure of any white node in Fig. 1 (a) has not much impact on the overall reliability of the system. However, if the red node *P* fails, the system is broken into two pieces. For a P2P system, the critical node *P* can be protected by inserting edge *AB* or *AC* to the system, as illustrated in Fig. 1(b). As a result, the network will remain connected and the services will not be interrupted when node *P* leaves. If it is a sensor network, we can give the critical node *P* a higher priority in competing channels or exempt it from unnecessary sensing jobs to save its power and prolong its lifetime. The motivation of our work is to answer the following two questions. 1) If we cannot afford to protect every node, is it effective to just concentrate on the critical nodes? 2) How do we identify the critical nodes?

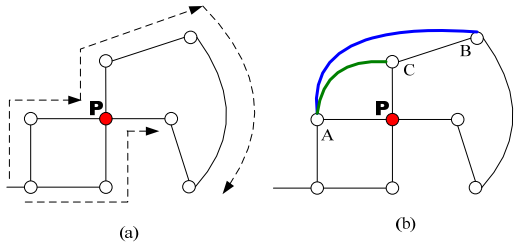

Figure 1. Cut vertex in a P2P overlay or a WSN

We model a distributed system, such as a P2P overlay or a WSN, by an undirected graph $G = (V, E)$ where the vertex set V represents units such as hosts or sensors, and the edge set E represents physical links. Intuitively, all the cut vertices in a graph are critical nodes. For ease of expression, we use terms "node" and "vertex", "edge" and "connection" interchangeably in the remainder of this paper. It is trivial to compute the cut vertices if we have the complete information of a graph G [5]. However, collecting global information in a dynamic, large-scale distributed system is extremely difficult, if not impossible. While existing approaches usually depend on the global knowledge, they usually incur huge traffic overhead and are impractical in large-scale distributed systems [6, 7]. In this paper, we propose a novel approach to identify cut vertices in a distributed manner. A tool is designed for each node to determine if it is a critical node. Corresponding actions are then proposed to protect those critical nodes to achieve higher reliability.

The highlights of our approach are summarized as follows:

1. Local: without the help of global knowledge, a node can identify whether it is a cut vertex based solely on local information;

2. Traffic lightweight: the proposed adaptive detection does not insert any extra traffic into the network, because in both P2P and WSN systems, message transmissions are considered more resource-consuming than local computing;

3. Adaptive-accuracy: we use relatively limited information to identify cut vertices. Note that a 100% accurate algorithm is not necessary as long as all the cut vertices are detected and the number of wrongly identified cut vertices is small. Furthermore, corrective actions can be taken by requesting more information when necessary.

4. Dynamic: as P2P nodes come and leave frequently, and sensor nodes might run out of energy at any time, the algorithm is able to run at each node at any time to identify itself if it becomes a cut vertex at the given time.

To simplify the discussion, we focus on P2P systems in this paper. The remainder of this paper is organized as follows. Section 2 briefly discusses the related work. Section 3 presents the distributed approach. We then elaborate the correctness of our approach in Section 4, and introduce our simulation methodology as well as the results in Section 5. Our work is concluded in Section 6.

2. Related Work

The reliability of a system relies on many aspects, such as the resilience and stability of network topology, the availability of the data and service, and the efficiency of content distribution and transmission etc. The dynamic and self-organizing nature of large-scale distributed systems brings more challenges to this issue. Over the past few years, numerous methods and techniques have been proposed for monitoring, modeling, detecting failures [8, 9, 10], and various schemes have been developed to improve the availability and the system resiliency [11, 12, 13, 14, 15] to provide reliable services.

D. Dumitriu et al [11] sort the patterns of Denial-of-Service attacks and observe node behavior in P2P file sharing systems. Misbehaving "super-nodes" and power-law network topology are regarded as the contributing factors to the resilience of such systems. Instead of identifying vulnerable peers, they propose counter-strategies to provide immunity to the attacks which sacrifice performance in the absence of such attacks. In [8, 9], novel techniques are proposed for network anomaly detection and traffic measurement. This school of approaches mainly focuses on network anomaly and dynamic failure and is not beneficial to normal cases. Some other recently proposed algorithms also address the reliability problems of overlay networks [11, 12, 13, 14], but they treat all the nodes equally without distinguishing the “critical” nodes from the ordinary ones. Indeed, critical nodes do exist in all sorts of distributed systems, such as critical nodes in service [16], in data or resources [11], and in topology. They are extremely important to the reliability of network systems.

In this paper, we focus on the critical nodes from the topological perspective, specifically, cut vertices. Cut vertex is a concept in graph theory which has been studied extensively. Existing algorithms to find cut vertices in graph theory include the DFS-based algorithm and bi-connected component sorting algorithm [5, 7]. Both algorithms require global information of the topology. As we know, it is extremely difficult, if not impossible, to obtain such topology information in large scale distributed systems.

P. Keyani et al. [17] address distributed recovery from attacks in P2P systems. They propose to modify the P2P overlay so as to reduce the number of high degree nodes. High degree nodes are more vulnerable to attacks, which are similar to the role of critical nodes in our paper. Because detecting high degree nodes is less difficult than locating cut vertices, their method tries to reduce high degree nodes and is invoked only when an attack occurs. However, their method cannot improve the reliability of topology for the normal cases when there are no attacks.

Critical nodes in WSNs are studied more than those in P2P and other distributed systems, probably because they have limited power energy and communication ranges. W. Zhang et al. [17] propose a node placement algorithm in WSNs. Their approach focuses on a small portion of critical nodes to support fault-tolerant communication and the computation complexity is approximated to $O(1)$. From the topological point of view, the relay nodes they defined are very likely to be cut vertices. In comparison with the work in [17], our scheme concentrates more on cut vertex detection and topology optimization. Moreover, our scheme can be applied to both P2P overlays and WSNs. Proposals in [6, 15] are related to the work in this paper, but both rely on active detections to identify cut vertices, resulting in additional overhead. Although many overlay optimization approaches have been proposed for P2P systems [18, 19, 20], these studies mainly aim to reduce the searching overhead instead of improving the system reliability. L. Ramaswamy et al. [21] propose a mechanism for detecting and constructing clusters in a P2P system. But their approach cannot be applied to identify cut vertices.

3. The Distributed Approach

This section describes our distributed approach to identify the critical nodes in large-scale distributed systems. We assume the communication between peers/sensors is mainly by flooding. With the help of information interchanged among peers/sensors, we then design a zero-overhead adaptive detection method to identify critical nodes. The accuracy of the adaptive detection can be reinforced by an active detection method with a fairly low cost. Both the adaptive and active detection will be executed periodically to reflect the dynamics of large-scale P2P overlays and WSNs.

We begin our discussion from essential definitions and principles, followed by the proposed adaptive detection and active detection. A straightforward method of cut vertex neutralization is then introduced to enhance the system reliability. The last part of this section discusses the overhead of our scheme.

3.1. Definitions and Principles

As stated in Section 1, we model a distributed system, such as a P2P overlay or a WSN, by an undirected graph $G = (V, E)$ where the vertex set V represents units such as hosts or sensors, and the edge set E represents physical links. The following are the formal definitions used in this discussion [5].

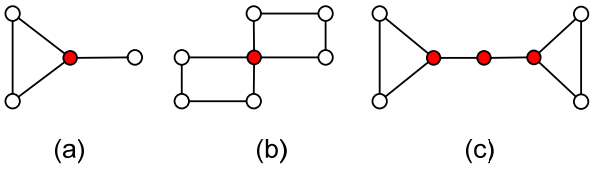

Figure 2. Three cases of cut vertex

Cut vertex: a vertex of a graph such that removal of it causes an increase in the number of connected components.

Bridge: an edge whose removal disconnects a graph.

Component: a maximally connected subgraph.

K-connected: if it is always possible to establish a path from any vertex to all others even after removing any $(k - 1)$ vertices, then the graph is said to be k-connected.

Block: a maximally connected subgraph having no cut vertex. A block in a graph can be either a 2-connected subgraph or a bridge with its end-vertices.

Because a cut vertex partitions one component into many, we come to the following **theorem**: *a vertex is a cut vertex if and only if it is the joint of multiples blocks*. The correctness of this theorem is proved in Section 4.

Without loss of generality, three possible cases are illustrated in Fig.2, where two blocks are connected by one or more cut vertices. In Fig. 2(a), the cut vertex (red point) is the joint of a 2-connected subgraph and a bridge; in Fig. 2(b), the cut vertex is the joint of two 2-connected subgraphs; in Fig. 2(c), the cut vertex in the middle is the joint of two bridges. Indeed, all cut vertices fall into these three cases.

3.2. Adaptive Detection

A P2P overlay or a WSN usually adopts the flooding mechanism where a message is propagated from a node to others hop by hop. The forwarding path of a flooded message also contains the connectivity information from the starting node to the current receiving node. Our basic idea is to utilize such information to find cut vertices by sorting the neighbors of a node into one or more blocks. According to the aforementioned **theorem**, a node is a cut vertex if its neighbors belong to multiple blocks.

In our scheme, we assume the following characteristics in a distributed system:

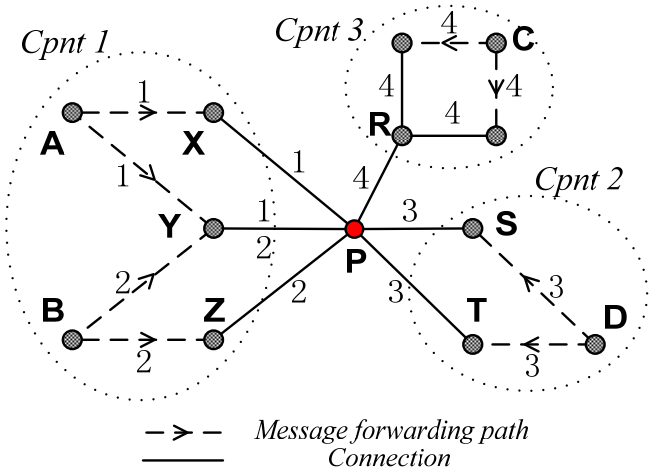

Figure 3. Example of adaptive detection

```
=====================================
/* P:    the node receiving a flooding message
msg:     the flooding message node P receives */
On_Flooding ()
{
  if (P receives msg for the 1st time)
  {
    P creates a new entry for msg in P.MsgList;
    if (msg.TTL>0)
    P forwards msg;
  }
  else   // P has received same msg before
  {
    P creates a new entry for msg in P.MsgList
    /* Find entries in P.MsgList which have same Message
    ID with msg;*/
    FindEntries(P.MsgList);
    /* Merge the neighbor where msg comes from and the
  neighbors in the entries just found; */
    MergeBlocks(P.BlockSet);
    P drops msg;
  }
}
=====================================
```

Figure 4. Pseudo code of the function *On_Flooding*()

1) A node forwards a message it receives to all its neighbors except the node which the message comes from;

2) Each message is assigned a globally unique message ID;

3) A node forwards each message only once. If it receives the same message later, it simply drops the duplicates.

Now let us examine the example in Fig. 3. In this figure, node *P* is a cut vertex that connects three blocks. The removal of node *P* causes the whole graph to be partitioned into three components, labeled as *Cpnt 1*, *Cpnt 2*, and *Cpnt 3*, as denoted in the figure with dotted circles.

In the adaptive detection, each node keeps track of recently received messages. A list of records is cached on each node, called *MsgList*, with each entry representing a message received recently in the format of <*Message ID, Neighbor where the message comes from*>. A node also keeps the data-structure called *BlockSet* that records the block information of its neighbors. At the beginning of an adaptive detection period, *MsgList* is empty and all the neighbors of a node are assumed to be in different blocks.

During the periodical executions of the adaptive detection, node *P* receives messages from the nodes in all the three components. For instance, when node *A* gets message 1, it broadcasts message 1 to its neighbors. When nodes *X* and *Y* receive message 1 through disjoint paths and forward it to node *P*, node *P* will discover that there exist two disjoint paths in which message 1 is delivered. Though node *P* perhaps does not know which node originally issues the message, it is able to find a cycle (*A*, *X*, *P*, *Y*) containing nodes *X*, *Y*, and itself. As a result, nodes *X* and *Y* are 2-connected and they are in the same block (refer to the **Theorem 2** in Section 4). Node *P* merges the blocks containing nodes *X* and *Y* into one single block. Similarly, after receiving message 2 from nodes *Y* and *Z*, node *P* puts nodes *X*, *Y*, and *Z* in the same block (refer to

the **Lemma 1** in Section 4). Nodes *S* and *T* are merged into one block after node *P* receives message 3. Node *R* remains isolated in its block. There is no other node except *R* that could possibly forward message 4 to node *P*.

Figure 4 provides the pseudo code of the function *On_Flooding*(). As more messages arrive, node *P* keeps merging the blocks in its *BlockSet*. At the end of an adaptive detection period, all the neighbors of node *P* will probably be merged into a few blocks. If only one block remains in the *BlockSet*, node *P* is not a cut vertex. Otherwise, an active detection process is triggered for further determination, which we introduce in Section 3.3. Note that the adaptive detection is executed passively during flooding search. It does not incur any additional traffic overhead.

3.3. Active Detection

With the adaptive detection, a node knows for sure that it is not a cut vertex if all the neighbors belong to a single block. However, having two or more blocks remaining at the end of adaptive detection does not mean a node is a cut vertex. For example, a node might not be able to receive all the flood messages from its neighbors because the requested item of a search is found or the *TTL* threshold is reached. Therefore we believe that an active detection is necessary to further identify cut vertices. Compared to the adaptive detection, the active detection achieves shorter convergence time at the cost of additional but acceptable traffic overhead.

If the neighbors of a node are sorted into two or more blocks at the end of adaptive detection, it regards itself as a cut vertex candidate and immediately starts an active detection process. At first, it randomly selects a neighbor from each block and numbers the connection to each neighbor with a unique index (e.g. 1, 2, 3 ...). Then the node sends probe messages along these connections. The format of the probe message is <*ID, timestamp, TTL, connection_index*>, where *ID* is the candidate's node ID, *timestamp* records the time the probe message is generated, *TTL* is a pre-configured threshold and *connection index* represents the connection between the neighbor and the candidate. Each node keeps a *connection list*. There is one entry for each candidate in the connection list with the format of <*candidate's ID, timestamp, connection_index* 1*, connection_index* 2 …>. Upon receiving the probe messages from candidates, a node will take corresponding actions according to the information stored in its *connection list.*

For the ease of illustration, we still use the example in Fig.3 but remove unrelated nodes and symbols, as shown in Fig. 5. We further assume node *X*, *R*, and *S* are selected from each block. When other nodes receive the probe messages originating from candidate *P*, they compare the probe messages with the information stored in their *connection lists*. Nodes that receive the probe messages also keep the latest records of those cut vertex candidates. If there is no entry for a candidate, a new entry is created. After that the *TTL* value is reduced by 1 and the probe messages are then forwarded to the downstream nodes.

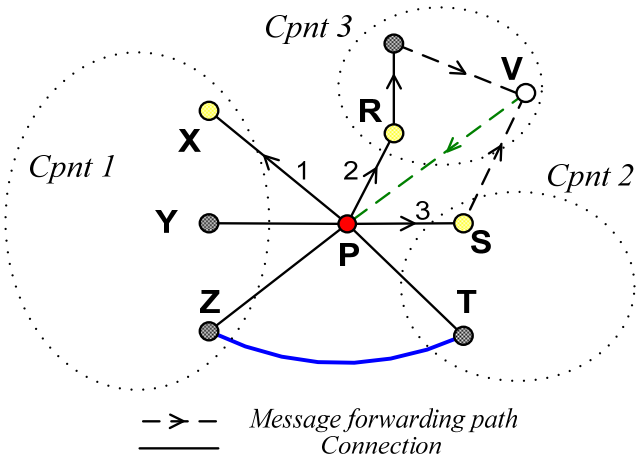

Figure 5. Example of active detection

In Fig. 5, node *V* receives two messages from the same candidate *P* but through different connections (connection 2 and connection 3). Therefore node *V* generates an arrival message containing node *P*'s *ID*, a *timestamp* and two *connection indexes* (2 and 3). The arrival message is sent to the candidate *P* at once, which is depicted as the green dashed arrow from *V* to *P* in Fig. 5.

On receiving the arrival message from node *V*, the candidate *P* is able to find a cycle (*P*, *R*, *V*, *S*) containing nodes *R*, *S*, and itself. That is, nodes *R* and *S* are 2-connected and they should be in the same block (refer to the **Theorem 2** in Section 4). So node *P* merges the blocks containing nodes *R* and *S* into one block. In other words, *Cpnt 2* and *Cpnt 3* are actually in one block.

In this way, the neighbors of a cut vertex candidate can be merged into fewer and fewer blocks. If only one block remains in the *BlockSet*, node *P* is not a cut vertex. Otherwise, node *P* must be a cut vertex. Since the initial *TTL* value of a probe message is usually small, we are able to get the result of the active detection much sooner than the adaptive detection. In other words, the active detection can be applied as a useful complement to the adaptive detection. It can also be utilized as an independent approach to identify cut vertices if we value speed over cost.

3.4. Cut Vertex Neutralization

The goal of cut vertex neutralization is to enhance the system reliability with respect to topology connectivity. Cut vertex neutralization is relatively easy to achieve by building extra connections between nodes in different blocks. For example, node *P* selects a neighbor from each block, such as *Z* and *T*. Then node *P* sends requests to nodes *Z* and *T* and asks them to connect with each other. After the new connection is built (depicted as the blue line in Fig. 5), the two initially independent blocks are merged into one block. Consequently, all nodes in the graph get 2-connected and node *P* becomes a normal node.

3.5. Overhead

We evaluate the traffic overhead by counting the messages delivered due to the cut vertex detection in a P2P system. Clearly, the adaptive detection does not incur any traffic overhead. For the active detection, suppose the system has n nodes, let c be the average degree of nodes and let t be the initial *TTL* value. Compared with the amount of arrival messages and connection requests, the amount of probe messages is much more because they are passed in the fashion of peer-to-peer flooding. Therefore the total traffic cost of the active detection is dominated by the cost of forwarding probe messages.

Note that a node will not forward the probe message if it has already sent an arrival message back to the corresponding candidate. We define the set of nodes which may be traversed by the same connection number of a candidate as the traversal set of that connection. Then the traversal sets of different connections of a candidate cannot overlap. As a result, the total traffic cost of probe messages should be $O(n^2c/2)$, where $nc/2$ is the number of edges in the whole graph. On the other hand, the traffic cost is also limited by the initial *TTL* value. It can never exceed $O(nc^t)$.

Therefore the total traffic cost of forwarding probe messages should be $min(O(nc^t), O(n^2c/2))$. For large-scale distributed systems, the value of c is usually much smaller than that of n. The inequality $c^t \leq n$ holds when the initial *TTL* value t is limited to save traffic cost. Thus we can conclude that the total traffic cost of active detection is $O(nc^t)$.

The storage overhead of adaptive detection on a node is dominated by the number of received flooded messages that come from different nodes. Let m_1 denote the size of an entry in the *MsgList*. Then the storage overhead on a node is $O(m_1c^t)$. Similarly, let m_2 denote the size of an entry in the *connection list*. The storage overhead of active detection on a node is $O(m_2c^t)$.

For real distributed systems, c is usually fairly small, such as 3, 4 or 5. We can use a small initial *TTL* value to limit the traffic cost of active detection, as well as to limit the storage overhead of both adaptive and active detections. The simulation results in Section 5 also demonstrate that a small value of t provides decent accuracy.

4. Correctness

Theorem 1: *Two blocks in a graph have at most one joint vertex.*

Lemma 1: *Let vertices A, B, C and D be four vertices in graph G. A, B and C are in the same block, while A, B and D are in the same block. Then A, B, C and D are in the same block, too.*

Theorem 2: *If there are two disjoint paths between two vertices in a graph, the two vertices are in the same block.*

Theorem 3: *A vertex is a cut vertex if and only if it is the joint of multiple blocks.*

Due to the page limit, we do not present the proofs here. Please refer to our technical report for detailed proofs (http://www.cse.ust.hk/~heyuan/Recu080118.pdf).

Consider an undirected graph $G = (V, E)$ to represent an overlay network, where V is the set of overlay nodes and E is the set of the edges of the overlay network. We illustrate the correctness of our distributed approaches.

Assume that the network topology is static and the initial *TTL* value of each flooding message is set to infinity. As long as there is a path between two nodes, the path will be detected by either the adaptive or the active detection. Therefore the overlay topology can be sketched if the detection lasts for an adequately long period of time. According to Theorem 3, if the neighbors of a node are finally merged into one single block, the node is not a cut vertex; otherwise, the node is a cut vertex. Thus the correctness of our distributed approaches is proved.

As for the accuracy of our approach, it is well known that given an appropriate topology construction, a query from one node to another in the overlay could be achieved in $O(lnN)$ hops, where N represents the size of the overlay [22]. Moreover, it has been widely recognized that most practical networks appear to be small-world or power-law networks, where the complexity of search is $O(lnN)$ or even $O(ln(lnN))$ [23]. Therefore, the length of a path between two nodes is usually within the logarithmic order of the size of the overlay. Through the real trace simulations in the next section, we also show that setting the initial *TTL* value to a small constant is sufficient to obtain a fairly high accuracy.

On the other aspect, without global information in our scheme, a tiny portion of nodes might be falsely identified as cut vertices. One possible case is that these nodes lie in very long cycles in the graph while such cycles are not detected by our distributed approaches with limited *TTLs*. However, such nodes are rare and those neglected long cycles probably make no sense in practical applications. Another possibility is due to the variations of the transmission delays among different nodes. Even though nodes might be falsely identified as cut vertices in our proposal, applying cut vertex neutralization to those false positive nodes will generate no harm to the system.

5. Performance Evaluation

We evaluate the impact of cut vertices, the accuracy of detection, the traffic overhead and the effect of cut vertex neutralization through intensive real trace experiments. Further experiments are conducted on the correlation between accuracy of detection and overlay dynamics. Aiming at mitigating the negative impact of cut vertices, our scheme is proved to be highly accurate, lightweight and greatly beneficial to the reliability of large scale distributed systems.

5.1. Experimental Methodology

Table 1. Information of selected traces

Trace	Size	Average degree	Number of Cut Vertices
1	10101	2.41	1315
2	13086	6.7	1865
3	15142	5.1	2075
4	25702	4.9	4169
5	40036	3.88	6448

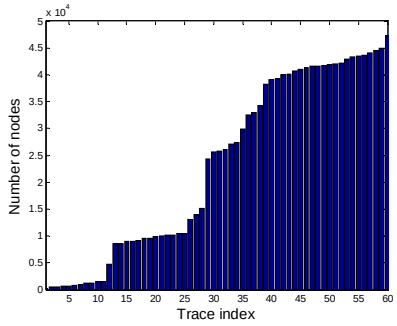

Figure 6. Size of traces

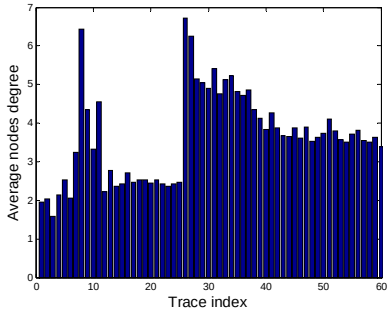

Figure 7. Average node degree

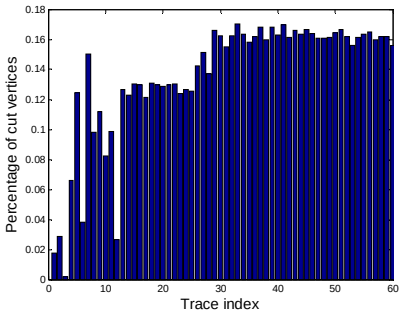

Figure 8. Proportion of cut vertices

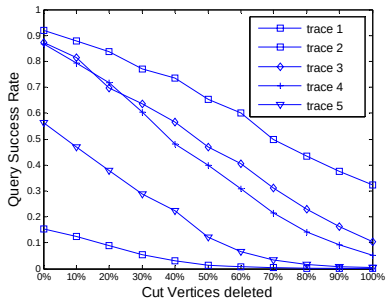

Figure 9. Impact on efficiency

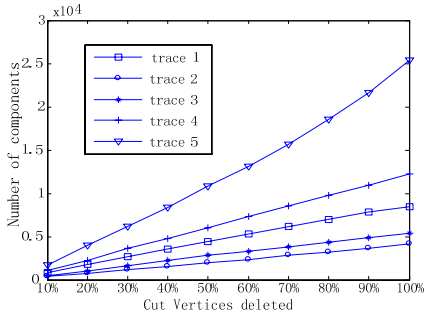

Figure 10. Impact on connectivity

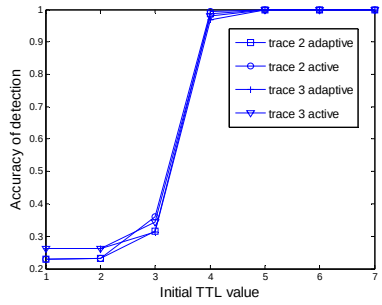

Figure 11. Accuracy of Detection

The overlay topology in our experiments is generated with the DSS Clip2 traces that were collected from Dec. 7th 2000 to June 15th 2001. We can provide the traces upon request.

Altogether we have 60 traces, and 13 of them are neglected because they contain small quantities of nodes and scarce connections. Nodes in the other 47 traces vary from around 8000 nodes to more than 45000 and the average degree varies from 2.4 to 6.7. We select 5 traces as the representative for the subsequent experiments, as listed in Table 1. Relevant information of all 60 traces is illustrated in Figs. 6-8.

5.2. Impact of Cut Vertices

We first take a look at the impact of losing the critical vertices in terms of query efficiency and connectivity. In Fig. 9, we measure the success rate of flooding-based queries, setting the *TTL* value at 5. It is well known that query efficiency largely relies on how well the nodes in the overlay are connected. Though cut vertices usually account for less than 10% of the overlay, deleting them often badly reduce the success rate of queries. For a typical example, trace 3, in Fig. 9, the original query success rate is 90%. When the cut vertices are deleted step by step, the query success rate rapidly drops. And after all the cut vertices are deleted (x=100%), the query success rate drops to only 10%.

Losing cut vertices also results in many isolated components in each overlay. As shown in Fig. 10, deleting a small portion of those vertices significantly increases the number of components. In other words, removal of cut vertices causes the overlay to be fragmented. For example, when half of the critical nodes leave, an overlay can break into thousands of disconnected components.

5.3. Accuracy of Detection

As stated in Section 2, all the cut vertices in a graph can be found by the traditional DFS algorithm. Based on the results of the DFS algorithm, we evaluate the accuracy of our proposed adaptive and active algorithm. Accuracy here refers to the percentage of real cut vertices in the output. For the adaptive detection, the output is the set of nodes identified as cut vertices after all the nodes have issued a flooding-based query. For the active detection, the output is the set of nodes identified as cut vertices after all the nodes have executed the active detection once. The preset *TTL* value in the experiment corresponds to the scope of flooding in the adaptive detection while it limits the range of probe messages in the active detection.

Curves in Fig. 11 demonstrate that both the adaptive and the active detection have high accuracy even when the initial *TTL* value is as small as 4 (more than 98% are correct). More interestingly, without introducing any extra traffic overhead, the adaptive detection performs almost as well as the active detection does. This finding also enables us to adopt a hybrid strategy (a combination of two kinds of detection) when identifying cut vertices to achieve high accuracy with low traffic overhead.

Additional experiments are conducted on trace 3. Setting the initial *TTL* value at 5, we change the proportion of nodes that do flooding-based queries in an adaptive detection period. The curve in Fig. 12 shows that as long as a small portion of nodes issue flooding-based queries, the adaptive detection can be precise enough. For example, when only 3% of nodes issue flooding-based queries, the resulting accuracy is as good as 90%. Considering the contrast in traffic overhead, this result further proves the advantage of the adaptive detection over the active detection.

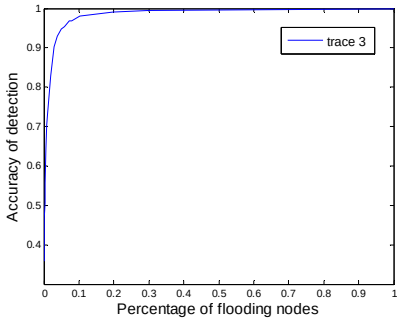

Figure 12. Accuracy of Detection

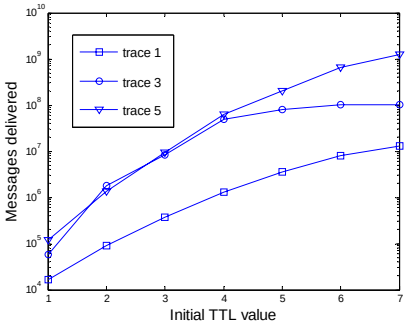

Figure 13. Cost of Active Detection

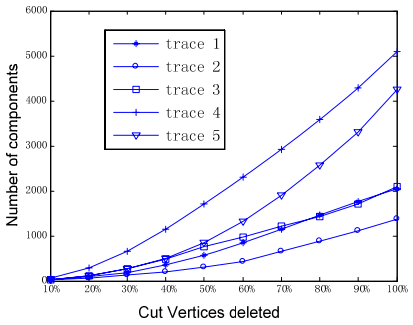

Figure 14. Effect of Neutralization (1)

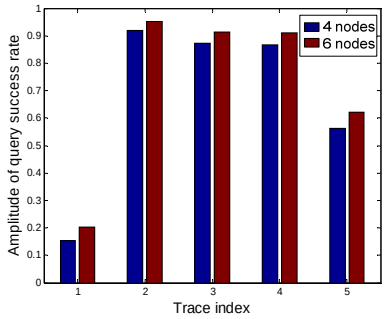

Figure 15: Effect of Neutralization (2)

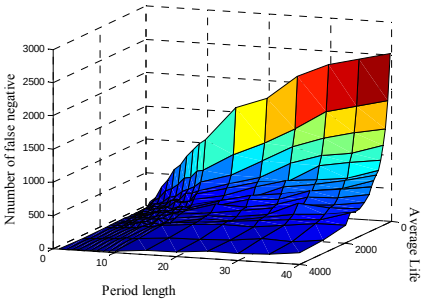

Figure 16: Accuracy vs. Dynamics (1)

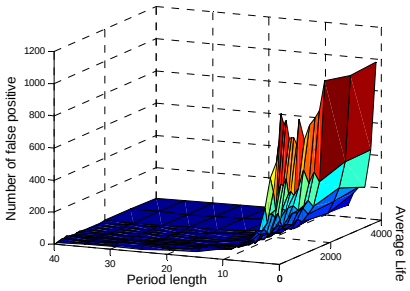

Figure 17: Accuracy vs. Dynamics (2)

5.4. Traffic Overhead

The major workload in the adaptive detection is information processing on each node. Considering the powerful processing capacity of modern processors, the computation overhead appears to be trivial. Hence, this discussion focuses on the traffic overhead incurred by the active detection, as shown in Fig. 13. Note that in our design, active detection is required only after a node cannot identify itself through adaptive detection, especially when we adopt hybrid identification strategies. According to the previous experiment results, most of the cut vertices can be identified by the adaptive detection. Therefore, only a tiny portion of nodes have to execute the active detection. On the other hand, the average cost on each node is acceptable. For example, when *TTL*=5 in trace 1, the active detection has an accuracy higher than 90% and the average traffic overhead is around 100 messages per node, without regard to the potential benefit of the adaptive detection.

5.5. Effect of Cut Vertex Neutralization

We evaluate the effects of cut vertex neutralization through two groups of experiments.

In the first group of experiments, we measure the number of components increased after deleting a certain amount of cut vertices, as shown in Fig. 14. Comparing the curves in Fig. 14 with those in Fig. 10, we find the remarkable benefit of cut vertex neutralization. After cut vertex neutralization, the destructive effect of deleting cut vertices is reduced to only 20%~30% of the original level.

In the second group of experiments, we measure the query success rate after cut vertex neutralization. Randomly taking 4 or 6 nodes as the query targets, Fig. 15 depicts the amplitude by comparing the query success rates before and after cut vertex neutralization. As we can see from the histograms, cut vertex neutralization improves the query efficiency by 20%~90%.

From the experiment results above, we conclude that cut vertex neutralization greatly improves the reliability of large scale distributed systems.

5.6. Accuracy vs. Dynamics

Since the information processed by the adaptive detection is periodically extracted from the flooding messages, the accuracy of the adaptive detection should have some correlation with the overlay dynamics. In the last two experiments, two metrics are adopted to measure the accuracy of the adaptive detection with dynamic overlay: the number of false negatives and the number of false positives. False negatives represent nodes which are cut vertices but not identified as cut vertices. False positives represent nodes which are not cut vertices but identified as cut vertices. Both numbers are counted as errors in the output but have different impacts.

When false negative nodes exist, the adaptive detection will inevitably miss some cut vertices (i.e. false negative nodes). After the subsequent cut vertex neutralization, the overlay connectivity is still vulnerable to the failure of cut vertices. False positive nodes, on the other hand, introduce unnecessary additional connections without any negative effect on the connectivity.

We count the number of false negatives and the number of false positives as we regulate the degree of overlay dynamics and the detection frequency. As we can see in Fig. 16 and Fig. 17, the average life span varies from 100 to 4000 sec while the period length of the adaptive detection varies from 1 to 40 sec. Fig. 16 shows that the number of false negatives has close correlation with both the detection frequency and the degree of overlay dynamics. Especially when the average life span is short, the detection accuracy becomes much more sensitive to the detection frequency. On the contrary, Fig. 17 shows that the overlay dynamics have little impact on the number of false positives. And the latter keeps at a fairly low level unless we execute detection too frequently.

As an observation result, we find it is reasonable to set comparatively low frequency of the adaptive detection. The output is highly accurate unless the overlay gets extremely unstable. More frequent detection is necessary in more dynamic environments. However, since the adaptive detection incurs zero traffic overhead, our distributed approaches keeps effective and lightweight under various scenarios.

6. Conclusion

The connectivity among the nodes basically determines the reliability of a large-scale distributed system. It is observed that a small portion of nodes are often more critical to the system reliability than the others. Removal of critical nodes usually destroys the topology connectivity and incurs substantive extra traffic within the systems.

We propose a distributed approach to identify cut vertices. The proposed scheme is composed of three parts: adaptive detection, active detection and cut vertex neutralization. The adaptive detection utilizes the information from common flooding message and realizes zero-overhead detection. As a complement to the adaptive method, the active detection is conducted to further improve the accuracy of detection. It is also feasible to solely adopt the active method for fast and lightweight detection. Based on the results of the detection, cut vertex neutralization builds additional connections around the critical nodes and enhances the system reliability. We prove the correctness of our scheme theoretically and evaluate the performance with trace-driven simulations.

Acknowledgements

This work is supported in part by NSF China Grants No.60673179, No.60573140, No.60736013, and No.60573135.

References

[1] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in Proceedings of *Multimedia Computing and Networking* (*MMCN*), 2002.

[2] F. Stann, J. Heidemann, R. Shroff, and M. Z. Murtaza, "RBP: Robust Broadcast Propagation in Wireless Networks," in Proceedings of *ACM SenSys*, 2006.

[3] O. Younis, S. Fahmy, and P. Santi, "An architecture for robust sensor network communications," *International Journal on Distributed Sensor Networks* (*IJDSN*), *Special issue on localized communication and topology protocols for sensor networks*, 2005.

[4] W. Zhang, G. Xue, and S. Misra, "Fault-tolerant relay node placement in wireless sensor networks: problems and algorithms;," in Proceedings of *IEEE INFOCOM*, 2007.

[5] F. Buckley and M. Lewinter, *A Friendly Introduction to Graph Theory*, 2002.

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

[7] B. Awerbuch, "A New Distributed Depth-First Search Algorithm," *Information Processing Letters*, vol. 20, pp. 147-150, 1985.

[8] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based Change Detection: Methods, Evaluation, and Applications," in Proceedings of *ACM SIGCOMM Internet Measurement Conference* (*IMC*), 2003.

[9] S. Kim and A. L. N. Reddy, "Image-based Anomlay Detection Technique: Algorithm, Implementation and Effectiveness"," *IEEE JSAC*, 2006.

[10] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating Internet Bottlenecks: Algorithms, Measurements, and Implications," in Proceedings of *ACM SIGCOMM*, 2004.

[11] D. Dumitriu, E. Knightly, I. Stoica, and W. Zwaenepoel, "Denial of Service Resilience in Peer-to-Peer File Sharing Systems," in Proceedings of *ACM SIGMETRICS,* 2005.

[12] Y. Lin, B. Liang, and B. Li, "Data Persistence in Large-scale Sensor Networks with Decentralized Fountain Codes," in Proceedings of *IEEE INFOCOM 2007,* 2007.

[13] B. Y. Zhao, L. Huang, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz, "Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays," in Proceedings of *IEEE ICNP,* 2003.

[14] Y. Zhu and Y. Hu, "Making Peer-to-Peer Anonymous Routing Resilient to Failures," in Proceedings of *IPDPS,* 2007.

[15] T. Q. Qiu, E. Chan, and G. Chen, "Overlay Partition: Iterative Detection and Proactive Recovery," in Proceedings of *IEEE ICC,* 2007.

[16] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the Source," in Proceedings of *IEEE ICNP,* 2002.

[17] P. Keyani, B. Larson, and M. Senthil, "Peer Pressure: Distributed Recovery from Attacks in Peer-to-Peer Systems," in Proceedings of *IFIP Workshop on Peer-to-Peer Computing,* 2002.

[18] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," in Proceedings of *ACM SIGCOMM,* 2003.

[19] C. Tang and P. K. Mckinley, "On the Cost-Quality Tradeoff in Topology-Aware Overlay Path Probing," in Proceedings of *IEEE ICNP,* 2003.

[20] M. Li, W.-C. Lee, and A. Sivasubramaniam, "DPTree: A Balanced Tree Based Indexing Framework For Peer-To-Peer Systems," in Proceedings of *IEEE ICNP,* 2006.

[21] L. Ramaswamy, B. Gedik, and L. Liu, "A Distributed Approach to Node Clustering in Decentralized Peer-to-Peer Networks," *IEEE Transactions on Parallel and Distributed Systems,* 2005.

[22] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," in Proceedings of *IEEE INFOCOM,* 2004.

[23] A. Iamnitchi, M. Ripeanu, and I. Foster, "Small-world File-sharing Communities," in Proceedings of *IEEE INFOCOM,* 2001.