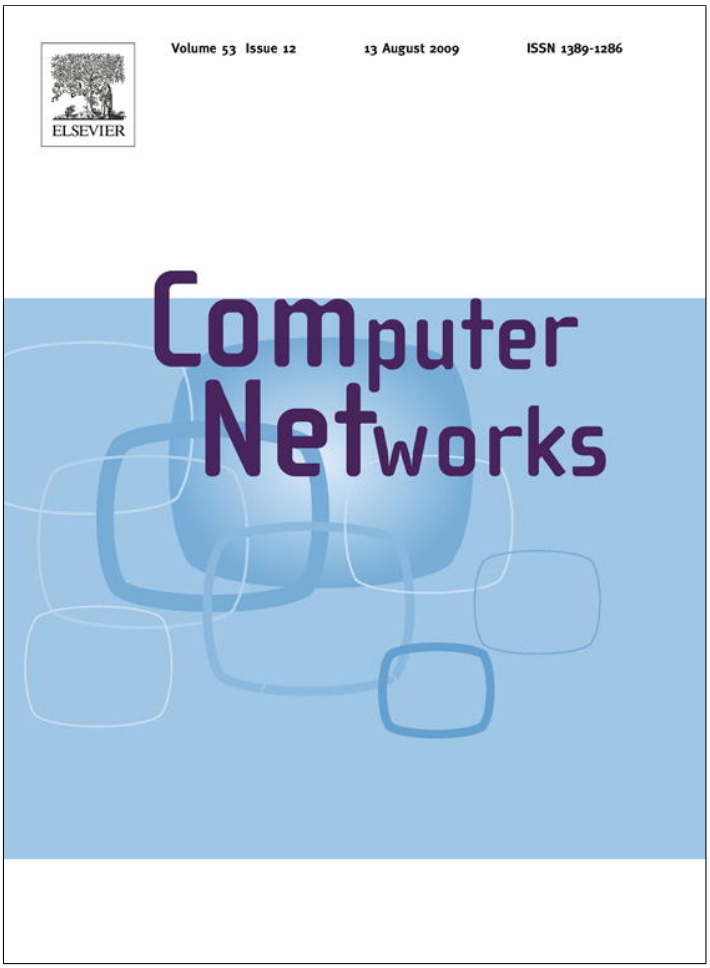


Computer Networks 53 (2009) 2140–2152

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

On the reliability of large-scale distributed systems – A topological view

Yuan He [a,*], Hao Ren [b], Yunhao Liu [a], Baijian Yang [c]

[a] Hong Kong University of Science and Technology, Department of Computer Science and Engineering, Hong Kong
[b] College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China
[c] Department of Technology, Ball State University, Muncie, IN 47306, United States

ARTICLE INFO

Article history:
Received 12 August 2008
Received in revised form 25 March 2009
Accepted 25 March 2009
Available online 1 April 2009

Responsible Editor: A. Capone

Keywords:
Cut vertex
Peer-to-peer
Reliability
Detection
Distributed method

ABSTRACT

In large-scale, self-organized distributed systems, such as peer-to-peer (P2P) overlays and wireless sensor networks (WSN), a small proportion of the nodes are likely to be more critical to the system's reliability than 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 not available. 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 the key performance gains are measured and verified through trace-driven simulations.

1. Introduction

Large-scale distributed systems, such as Peer-to-peer (P2P) overlays 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 at any time they want, 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. Also, nodes in a distributed system are usually not equally important from the reliability point of view. For example, Saroiu et al. [1] observed that after randomly removing 30% of the nodes from a 1771 nodes Gnutella network, 1106 of the remaining 1300 nodes stayed connected, and were likely to exchange and share information. In contrast, if 4% of carefully selected nodes are removed from the system, the overlay was fragmented into hundreds of pieces. Other researchers also noticed the existence of critical nodes or critical links in the context of P2P or sensor network systems [2–4].

Fig. 1a illustrates a topology of a portion of a P2P overlay network or a WSN. The dashed arrows depict the service/information flow. Clearly, the failure of any white node in Fig. 1a will not have much impact on the overall reliability of the system. However, if the red node P fails, the consequence is serious. The motivation of our work is to answer the following question: if we cannot afford to protect every node, will it be effective just protecting the critical nodes? For example, in a P2P system, if we can identify node P is critical, we may add edge AB or AC to protect the network from being partitioned by the crash/departure of P, as illustrated in Fig. 1b. As a result, the network is still connected and the service will not be interrupted when P leaves. If it is a sensor network, we may give critical node P a higher priority in competing channels or release node P from unnecessary sensing jobs to save its power and hence results in a longer lifetime.

We model a distributed system, such as a P2P or a WSN, by an undirected graph $G = (V, E)$ where the vertex set V

* Corresponding author. Tel.: +852 98179865.
E-mail addresses: heyuan@cse.ust.hk (Y. He), renhao1973@gmail.com (H. Ren), liu@cse.ust.hk (Y. Liu), byang@bsu.edu (B. Yang).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.03.012

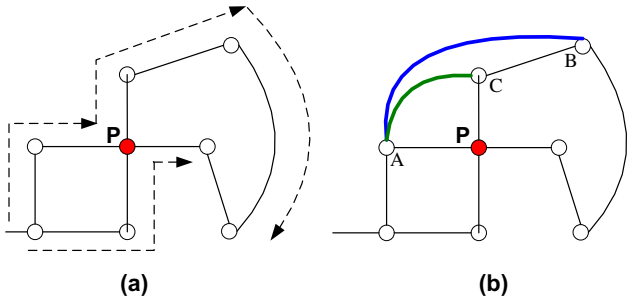


Fig. 1. Cut vertex in a P2P overlay or a WSN.

represents units such as hosts or sensors, and the edge set E represents physical links. Intuitively, all the cut vertices [5,6] in a graph are critical nodes. For ease of expression, we use the 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 [6]. However, collecting global information in a dynamic, large-scale distributed system is non-trivial. While existing approaches usually depend on the global knowledge, they usually incur huge traffic overhead [7] and are impractical in large scale distributed systems. In this paper, we propose a novel approach to identify cut vertices in a distributed manner. A tool is offered for each node to determine whether or not it is a critical node. Corresponding actions are then proposed to protect those critical nodes to achieve higher reliability.

The highlights of our proposal 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*: no need to 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*: use relatively limited information to estimate the probability of being a cut vertex. We argue that a 100% precise algorithm is not necessary as long as all the cut vertices are detected and the number of wrongly identified cut vertices is small. Furthermore, we can make corrective decisions 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 our discussion, we focus on the P2P systems in this paper. The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the distributed approach. We then discuss the properties of our approach in Section 4, and illustrate our simulation methodology and 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 to monitor, model and detect failures [8–16], and various schemes and mechanisms have been developed to improve the availability and the system resilience to provide more reliable services [17–22].

Dumitriu et al. [18] sort the patterns of Denial-of-Service attacks and observe node behavior in P2P file sharing systems. Misbehaving "supernodes" and power-law network topology are considered the contributing factors to the resilience of such systems. Instead of identifying vulnerable peers, they propose counter-strategies to provide immunity to the attacks by sacrificing performance in the absence of such attacks. In [8,13], novel techniques are proposed for network anomaly detection and traffic measurement. This school of approaches focuses mainly 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 [17–22], but they treat all the nodes equally without distinguishing "critical" nodes. Indeed, critical nodes do exist in all sorts of distributed systems, such as critical nodes in service [23], in data or resources [18] and in topology. We believe critical nodes 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 the classical 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 [6]. Both algorithms require global information of the network topology. As we know, it is extremely difficult, if not impossible, to obtain such topology information in large-scale distributed systems.

Ahuja et al. [24] propose an algorithm for finding cut vertices in asynchronous networks, which is based on the distributed DFS algorithm proposed in [25]. Those two algorithms share a common feature with our algorithm in the sense that they also do not require global knowledge of the network topology. The applied scenarios of them, however, are static asynchronous networks. Network dynamics (such as frequent joining in and departure of nodes, unexpected failure, and changes in network connections), which are the nature of P2P systems and WSNs nowadays, are not taken into account. For instance, one run of the algorithm in [24] costs $2n$-2 time steps, where n is the number of nodes in the network. If there is any change in the network topology during the running period, the output of the algorithm might become incorrect. Differing from the aforementioned proposals, the scheme proposed in this paper addresses the issue of finding cut vertices in distributed systems like P2P overlays and WSNs. As demonstrated by our trace-driven simulations

in Section 5, our scheme remains accurate in dynamic network topologies. Moreover, its traffic overhead is fairly low. We will present a comparison analysis in Section 3.5.

Keyani et al. [26] address distributed recovery from attacks in P2P systems. They propose to modify the P2P overlay to reduce the number of high degree nodes. High degree nodes are more vulnerable to attacks, which is 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 range. Keyani et al. [26] 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, those relay nodes they defined are very likely to be cut vertices. In comparison with the work in [26], our scheme concentrates more on cut vertex detection and topology optimization. Moreover, our scheme can be applied to both P2P overlays and WSNs.

Although many studies on optimizing overlay topologies have been proposed in P2P systems [27–31], the goal of these studies is mainly to reduce the search overhead instead of improving the system reliability. Ramaswamy et al. [32] propose a mechanism for detecting and constructing clusters in a peer-to-peer system. But their approach cannot be applied to detect cut vertices. To the best of our knowledge, the scheme we propose in this paper is the first distributed approach to detect cut vertices in large-scale distributed systems.

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 between peers/sensors, we then design a zero-overhead adaptive detection method to identify critical nodes. The accuracy of the detection can be reinforced by an active detection method with a fairly low cost. Both 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 cut vertex neutralization method is then introduced to improve the system reliability. The last part of this section addresses the overhead of our scheme.

3.1. Definitions and principles

As stated in Section 1, we model a distributed system, such as a P2P 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 [5,6].

Component: A component is a maximally connected subgraph.
Cut vertex: A cut vertex is a vertex of a graph such that removal of the vertex causes an increase in the number of connected components.
Bridge: A bridge, or cut edge, is an edge whose removal disconnects a graph.
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 block is 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 separates 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. In Fig. 2a, the cut vertex (red point) is the joint of a 2-connected subgraph and a bridge; in Fig. 2b, the cut vertex is the joint of two 2-connected subgraphs; in Fig. 2c, 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 the others hop by hop. The forwarding path of a flooding message also contains the connectivity information from the starting node to the current node. We propose 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 of flooding in a distributed system.

(1) A node forwards the flooding message it received to all its neighbors except the node where the message came from.
(2) Each flooding message is assigned a globally unique message ID. In real P2P overlays and WSNs, a node is often assigned a globally unique ID. Without global knowledge of the network, this can be realized in

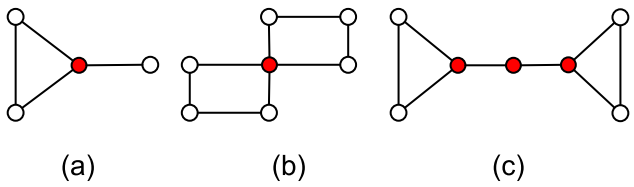


Fig. 2. Three cases of cut vertex.

various manners, e.g. by computing a cryptographic hash of the node's IP address (in P2P overlays) or its geographic locations (in WSNs). The message ID can thus be uniquely assigned, based on the starting node ID and the time the message is generated.

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

Now let us examine the example illustrated in Fig. 3. In this figure, node *P* is a cut vertex that connects three blocks (particularly vertex *P* and vertex *R* together with the bridge between them form one block). The removal of node *P* will cause the 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, ID of the 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 execution of 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*.

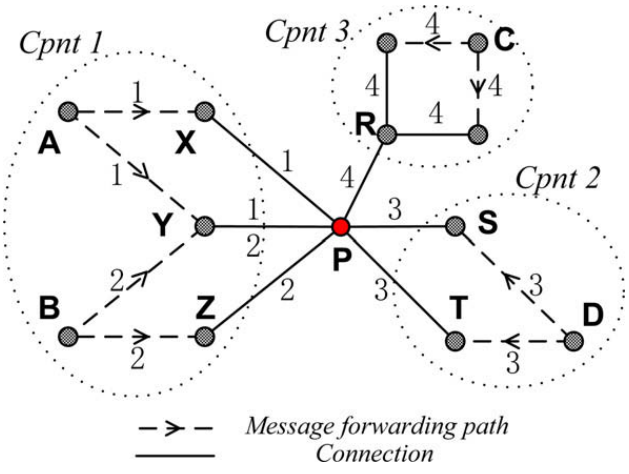


Fig. 3. Example of adaptive detection.

Fig. 4 provides the pseudo code of the function *On_Flooding*(), which explains the actions of a node when it receives a flooded message.

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 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 candidate of cut vertex 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 *connection-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* (time to live) is a pre-configured number of hops that the message can be forwarded, and *connection-index* denotes the index of the connection via which the candidate sends the probe message. 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 a probe message, one of the following situations may arise.

(1) The node has already received the message, or the *timestamp* of the message is older than that stored in the corresponding connection list entry. The node just drops the message.

(2) There is no entry for the candidate that issues this probe message. The node creates a new entry for it.

(3) The *timestamp* in the received message is newer than that stored in the corresponding connection list entry. The candidate replaces the old *timestamp* and *connection-indexes* stored in the connection list with the new one.

```
/* msg: a flooding message;
msg.TTL: a pre-configured number of hops that msg can be forwarded
msg.MID: the message ID of msg;
P: the node receiving msg; */
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;    // Forward msg if it is new and not expired yet;
  }
  else    // P has received the 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, msg.MID );
    // Merge the neighbors who forward the same msg to node P into one block
    MergeBlocks(P.BlockSet );
    // Drop msg after the merging operation.
    P drops msg;
  }
}
```

Fig. 4. Pseudo code of function *On_Flooding*().

(4) The *timestamp* of the received message is the same as the one stored in the corresponding connection list entry but the *connection-index* of the message is not the same. The node adds the new *connection-index* to the corresponding entry and sends an arrival message back to the candidate. The arrival message therefore contains the *ID* of the current node, two or more *connection-indexes*, and the *timestamp* stored in the entry. A node does not send any arrival messages until it receives at least two probe messages with different *connection-indexes*.

For ease of illumination, we still use the previous example in Fig. 3 while removing unrelated nodes and symbols, as shown in Fig. 5. Upon receiving the probe messages from the candidates, a node takes corresponding actions according to the information stored in its *connection list*. There are mainly four options and the pseudo code is shown in Fig. 6.

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*, and then take corresponding actions as described above. After that the *TTL* value is reduced by 1 and the probe messages are forwarded to the downstream nodes.

In Fig. 5, node V receives two probe messages from the same candidate P with the same *timestamp* but through different connections (connection 2 and connection 3). According to the algorithm in Fig. 6, node V generates an arrival message containing node V's *ID*, two *connection-indexes* (2 and 3) where the probe messages come from, and the *timestamp* in the corresponding connection list entry (i.e. the time when the two probe messages are generated). 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,

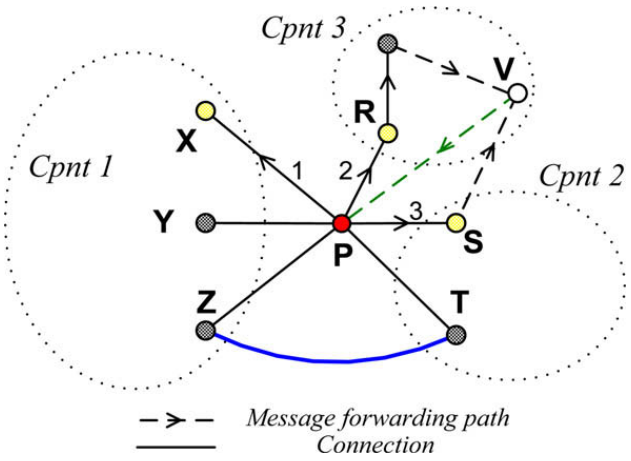


Fig. 5. Example of active detection.

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 are 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 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.

```
/* V: the node receiving a probe message;
list: node V's connection list;
msg: the probe message node V receives;
msg.ID: the ID of the candidate node that issues msg;
msg.TTL: a pre-configured number of hops that msg can be forwarded;
msg.timestamp: the time msg is generated;
msg.connection-index: index of the connection via which the candidate sends msg;
arr-msg: the arrival message generated by node V.*/

On_Probe()
{
  if (msg.ID not in list)
  {
    V creates a new entry for msg.ID; // Creates a new entry for a new msg;
    if (msg.TTL>0)
      V forwards msg;
  }
  else if (V has received the same msg before || msg.timestamp<list.(msg.ID).timestamp)
  {
    V drops msg; // Drop msg if it is out of date or has been received before;
  }
    else            // msg.timestamp≥list.(msg.ID).timestamp
    {
      if (msg.timestamp>list.(msg.ID).timestamp)
      {
        // Update the corresponding connection index entry;
        list.(msg.ID).timestamp= msg.timestamp;
        Replace list.(msg.ID).connection-indexes with msg.connection-index;
        if (msg.TTL>0)
          V forwards msg;
      }
      else   // msg.timestamp=list.(msg.ID).timestamp
      {
        // msg is sent to V via different connection-indexes;
        Add (msg.connection-index) into list.(msg.ID).connection-indexes;
        // arr-msg is generated and sent back to the probing candidate node;
        V generates arr-msg;
        arr-msg.ID=V.ID;
        arr-msg.timestamp = list.(msg.ID).timestamp;
        list.(msg.ID).connection-indexes -> arr-msg.connection-indexes;
        V sends arr-msg to msg.ID;
      }
    }
}
```

Fig. 6. Pseudo code of the function *On_Probe*().

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. Traffic overhead

We evaluate the traffic overhead by counting the messages delivered due to the cut vertex detection in a P2P system. Note that flooding is usually adopted in P2P overlays and WSNs as the basic mechanism for query and data dissemination [1,2]. The traffic overhead of flooding does exist as an element of system running, even if there is no adaptive detection. Hence the adaptive detection does not incur any additional traffic overhead because it only utilizes the information extracted from the existing flooding messages. For the active detection, suppose the system has n nodes, let c be the average node degree 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 P2P flooding. Therefore the total traffic overhead 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 are traversed by the same connection number of a candidate as the traversal set of that connection. And the traversal sets of different connections of a candidate will not overlap. As a result, the total traffic overhead of probe messages is $O(n^2c/2)$, where $nc/2$ is the number of edges in the whole graph. On the other hand, the traffic overhead is also limited by the initial *TTL* value. It can never exceed $O(nc^t)$. Therefore the total traffic overhead of forwarding probe messages is $min(O(nc^t), O(n^2c/2))$. For large-scale distributed systems, the value of c is usually much smaller than that of n. The inequality $c^t \leqslant n$ holds when the initial *TTL* value t is limited to save traffic cost. Thus we can conclude that the total traffic overhead of active detection is $O(nc^t)$.

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 overhead of active detection. The simulation results in Section 5 also demonstrate that a small value of t provides decent accuracy. Meanwhile, the size of all the probe messages sent in the active detection is identically 4 bytes. The per-node traffic overhead of active detection can thus be confined to a constant. Recall that the adaptive detection does not incur any traffic overhead. The aggregate traffic overhead on each node is constant, using the scheme in this paper. Compared with the algorithm in [24] which has a maximum total traffic overhead of $4m - n$ (m is the number of links in the network) and the length of each message is bounded by $2logn + 2$ bits, our scheme has fairly low traffic overhead while preserves high accuracy of detection.

4. Correctness

Theorem 1. *Two blocks in a graph have at most one common 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.*

Proofs of the above theorems and lemma [5,6] are easy and omitted.

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

Proof. The theorem to be proved is separated into two propositions. We prove them respectively. □

(1) A vertex is a cut vertex if it is the joint of multiples blocks.Without loss of generality, we assume that vertex P in graph G is the joint of two blocks: *block 1* and *block 2*. Suppose vertex A is in *block 1* and vertex B is in *block 2*. Thus vertex A and vertex P are connected, so are vertex B and vertex P. A, B and P are in the same component of graph G. By removing vertex P and its incident edges from G, we get a graph G'. In G', vertex A and vertex B are no longer connected. Otherwise, A and B are 2-connected in the G (one path including P and the other excluding P), which indicates that A and B are in the same block of graph G. Hereby we come to the contradiction. Consequently, A and B are in different components of graph G'.P is a cut vertex. Proposition (1) is proved.

(2) If a vertex is a cut vertex, it is the joint of multiples blocks.

First, a vertex with 0 or 1 degree cannot be a cut vertex. So a cut vertex must have at least two neighbors.

Suppose vertex P is a cut vertex in graph G. Without loss of generality, we assume the removal of vertex P generates two components – *cpnt1* and *cpnt2*. According to the definition of cut vertex, vertex P has at least one neighbor in each component. Suppose vertex A is a neighbor of P in *cpnt1* and vertex B is a neighbor of P in *cpnt2*. A and B are in different blocks. Otherwise A and B are 2-connected which contradicts that they belong to different connected components.. Thus P is the joint of two blocks. One block contains A and the other contains B. In addition, vertex P is the unique joint of the two blocks according to Theorem 1. Proposition (2) is proved.

From (1) and (2), Theorem 3 is proved. ¶

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 illuminate 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 the adaptive detection. Similarly, the active detection will always detect the path if the initial *TTL* of each probe message is set to infinity. Therefore the overlay topology can be sketched if the detection lasts for a long enough 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 [33–35]. 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 $O(ln(lnN))$ [36–38]. 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. This is mainly because 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. Even if these nodes are falsely identified as cut vertices, executing cut vertex neutralization to them will not generate any harm to the system.

Moreover, our scheme can be applied to both P2P overlays and WSNs which adopt flooding as the basic mechanism for query and data dissemination [1,2]. According to the analysis in Section 3.5, the total number of messages is $O(nc^t)$. The average number of messages processed by each node is $O(c^t)$ while c (the average node degree) and t (the initial *TTL* value) are usually very small constants. Thus the storage and processing cost on each node is confined to a constant, which is acceptable for both resource-constrained wireless sensor motes and powerful PCs.

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 a number of real trace experiments. Further experiments are carried out on the correlation between the accuracy of detection and overlay dynamics. The experimental results demonstrate that our scheme is highly accurate, lightweight and can greatly enhance the reliability of large-scale distributed systems.

5.1. Experimental methodology

The overlay topology in our experiments is generated with the DSS Clip2 traces [39] that were collected from December 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 45,000 and the average node 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. 7–9.

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. 10, we measure the success rate of flooding-based queries, setting the *TTL* value at 5. It is well known that query

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

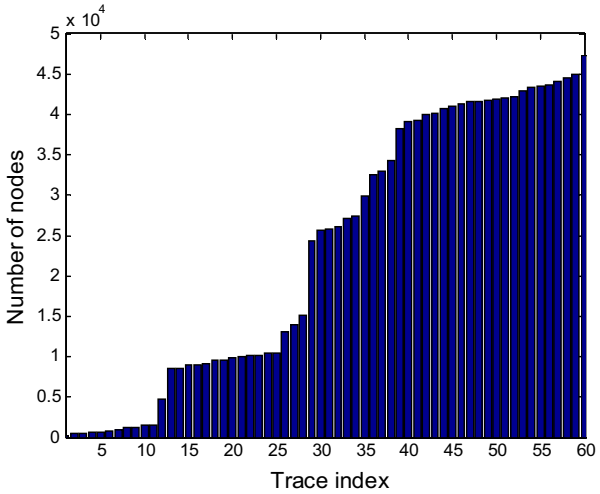


Fig. 7. Size of traces.

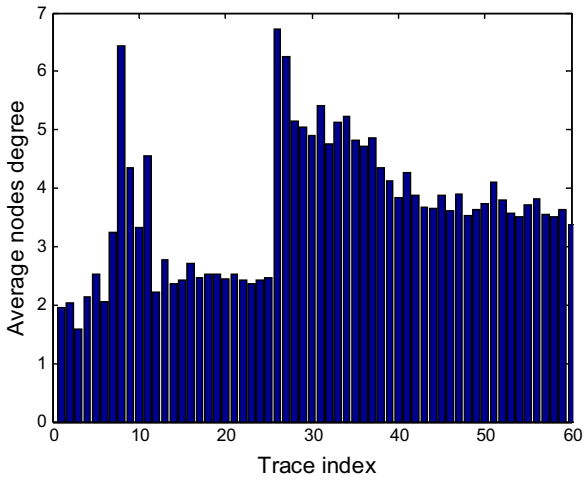


Fig. 8. Average node degree.

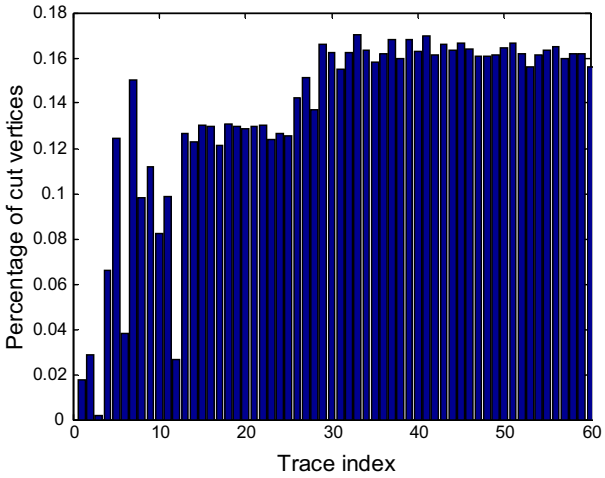


Fig. 9. Proportion of cut vertex.

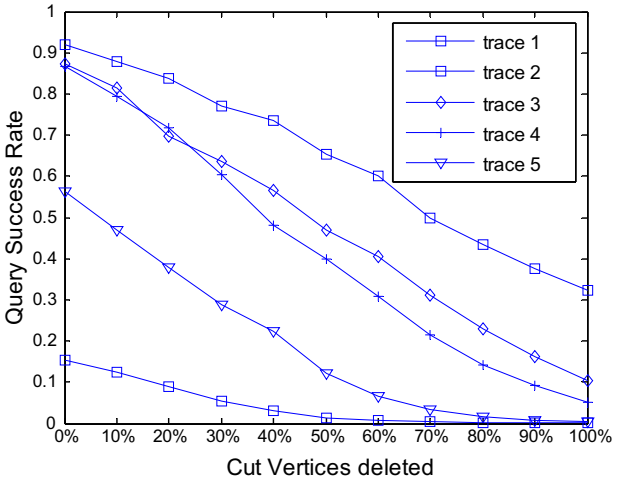


Fig. 10. Impact on query efficiency.

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

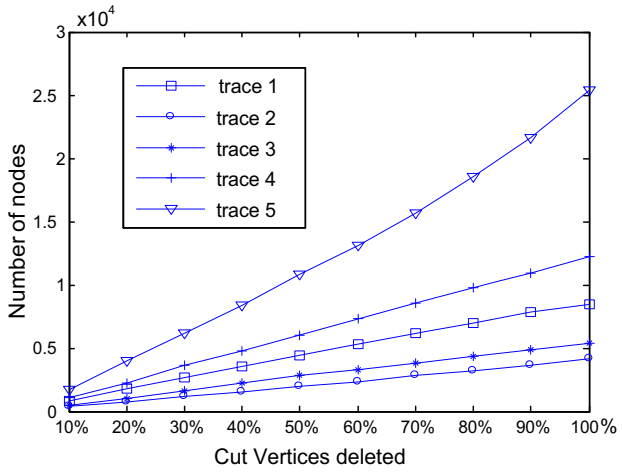


Fig. 11. Impact on connectivity (1).

the success rate of queries. For a typical example, trace 3, in Fig. 10, 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. 11, 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 thousand of disconnected components. Fig. 12 depicts the affected scope of cut vertices. The *Y*-axis represents the percentage of nodes in the fragmented components to all the nodes with connections in the original overlay. Clearly, each cut vertex will affect many other normal peers in the system.

5.3. Accuracy of detection

As stated in Section 2, all the cut vertices in a graph can be found by the traditional sequential DFS algorithm introduced in [6]. Based on the results of the DFS algorithm, we

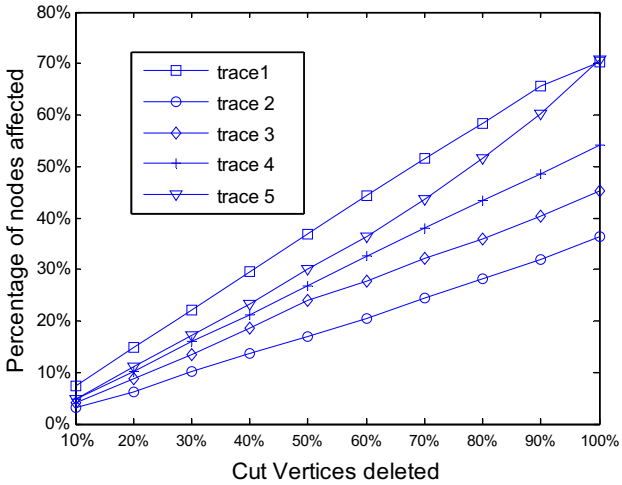


Fig. 12. Impact on connectivity (2).

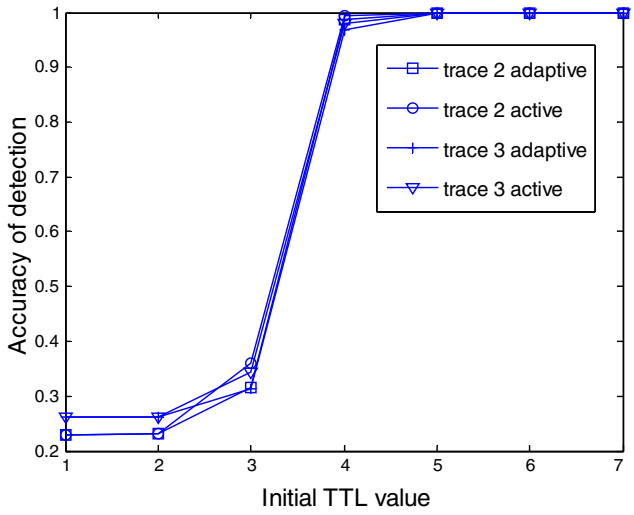


Fig. 13. Accuracy of detection (1).

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, and it limits the range of probe messages in the active detection.

Curves in Fig. 13 demonstrate that both the adaptive and the active detection have high efficiency even when the initial TTL value is as small as 4. More exciting to us, the adaptive detection always performs very close to the active detection but incurs zero traffic overhead. Such results encourage us to adopt hybrid strategies (a combination of two kinds of detection) when identifying cut vertices. As a result, we can achieve high accuracy with low traffic overhead at the same time.

Additional experiments are conducted on trace 3. Setting the initial *TTL* value to 5, we change the proportion of nodes that do flooding-based queries in an adaptive detection period. The curve in Fig. 14 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 factor of traffic overhead, this result further demonstrates the advantage of adaptive detection over active detection.

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 is negligible. Hence, this discussion focuses on the traffic overhead incurred by the active detection, as shown in Fig. 15. Note that in our design, active detection is required only after a node cannot determine whether it is a cut vertex through adaptive detection, especially when

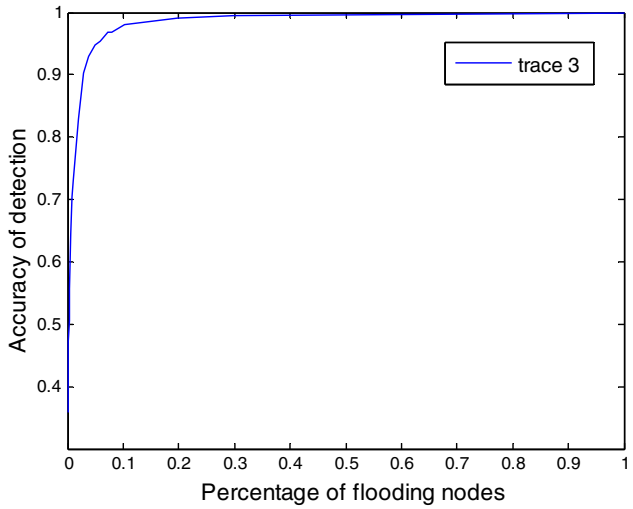


Fig. 14. Accuracy of detection (2).

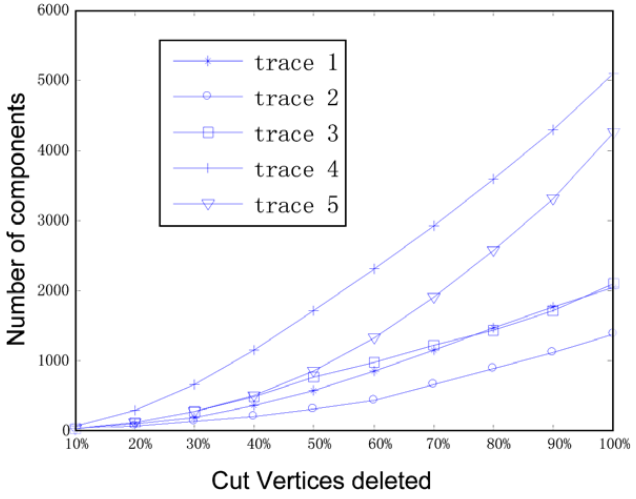


Fig. 16. Effect of neutralization (1).

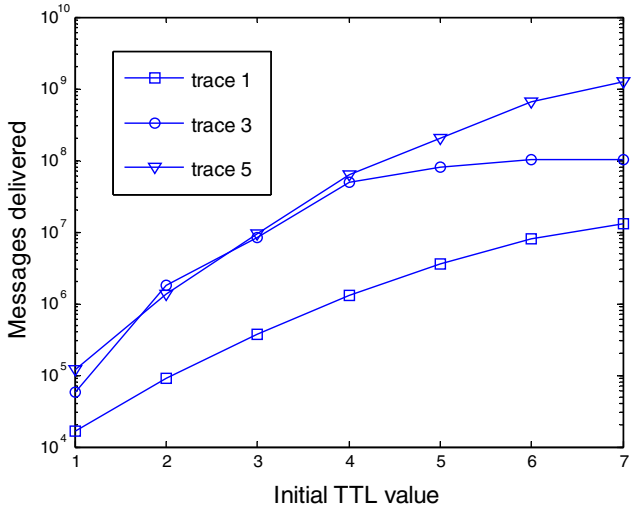


Fig. 15. Message incurred by active detection.

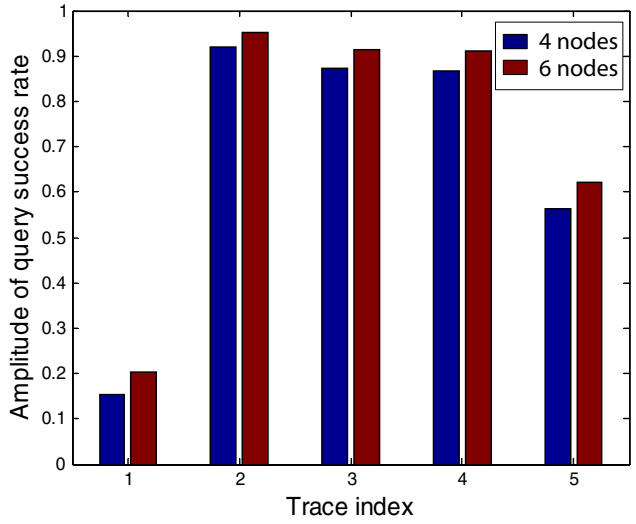


Fig. 17. Effect of neutralization (2).

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. For each node that runs active detection, the average cost is also very limited. For example, when *TTL* = 5 in trace 1, the active detection has an accuracy higher than 90% while the average traffic overhead is around 100 messages per node, without regard to the preceding fact that adaptive detection has already identified most of the cut vertices.

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. 16. Comparing the curves in Fig. 16 with those in Fig. 11, we find a remarkable benefit of cut vertex neutralization. With 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 (the blue[1] bar) or 6 (the red bar) nodes as the query targets, Fig. 17 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

[1] For interpretation of color in Figs. 1-3,5,7-23, the reader is referred to the web version of this article.

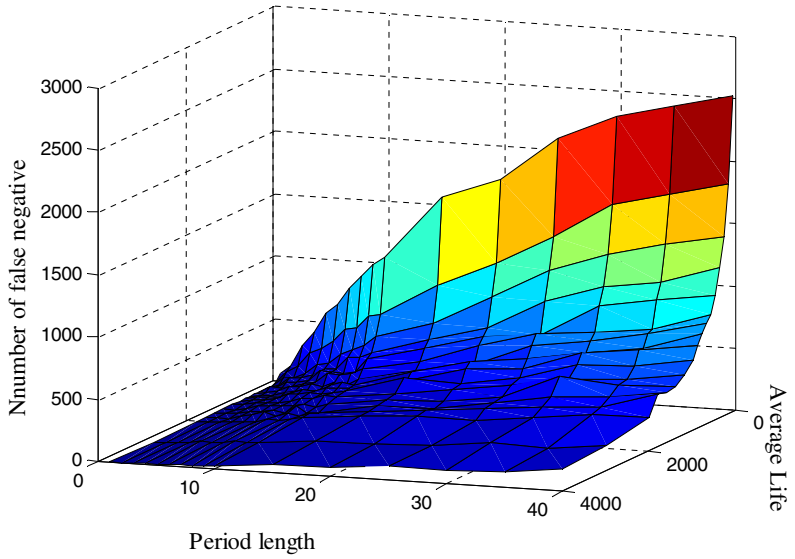


Fig. 18. Accuracy vs. dynamics (1).

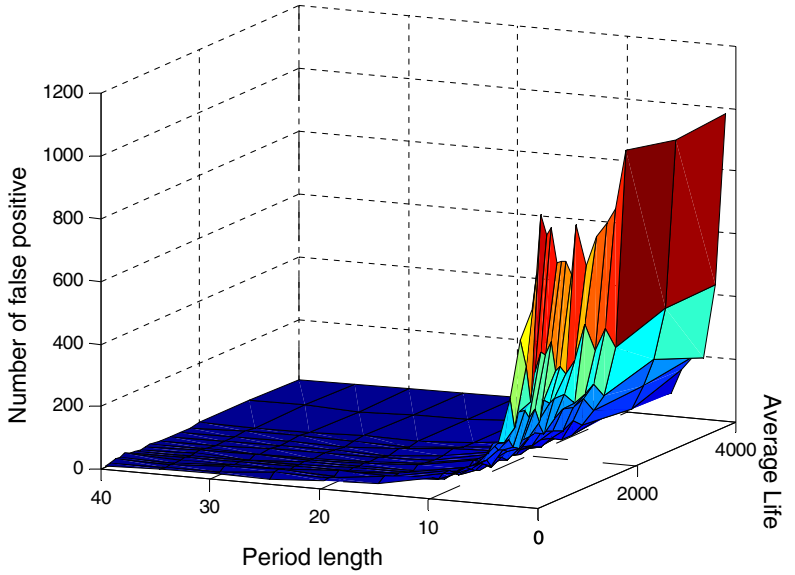


Fig. 19. Accuracy vs. dynamics (2).

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) and the overlay connectivity after cut vertex neutralization is still vulnerable. Nevertheless, false positive nodes cause no harms to the system reliability. They only require additional connections than necessary.

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 Figs. 18 and 19, the average life span varies from 100 to 4000 s while the period length of the adaptive detection varies from 1 to 40 s. Fig. 18 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. 19 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 relatively 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 approach achieves a good balance between efficiency and accuracy under a number of scenarios.

6. Conclusion

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

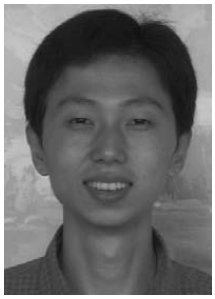
We propose a distributed approach to identify cut vertices. The proposed scheme is composed of three parts: adaptive detection, active detection and cut vertex neutralization. The adaptive detection utilizes the common flooding messages and achieves zero-overhead detection. As a complement to the adaptive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

Acknowledgement

This work is supported in part by NSF China Grants No. 60673166, 60736013, 60673179, and NSFC/RGC Joint Research Scheme N_HKUST614/07.

References

[1] S. Saroiu, P. Gummadi, S. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking (MMCN) San Jose, CA, USA, 2002.
[2] F. Stann, J. Heidemann, R. Shroff, M.Z. Murtaza, RBP: Robust broadcast propagation in wireless networks, in: Proceedings of ACM SenSys, Boulder, Colorado, USA, 2006.
[3] Xun Wang, Sriram Chellappan, Phillip Boyer, Dong Xuan, On the effectiveness of secure overlay forwarding systems under intelligent distributed DoS attacks, IEEE Transactions on Parallel and Distributed Systems 17 (7) (2006) 619–632.
[4] W. Zhang, G. Xue, S. Misra, Fault-tolerant relay node placement in wireless sensor networks: problems and algorithms, in: Proceedings of IEEE INFOCOM, Arizona State University, Tempe, USA, 2007.
[5] F. Harary, Graph Theory: Addison-Wesley, Reading, 1969.
[6] F. Buckley, M. Lewinter, A Friendly Introduction to Graph Theory, Prentice-Hall, New Jersey, 2002.
[7] X. Liu, L. Xiao, A. Kreling, Y. Liu, Optimizing overlay topology by reducing cut vertices, in: Proceedings of ACM NOSSDAV, Newport, RI, 2006.
[8] B. Krishnamurthy, S. Sen, Y. Zhang, Y. Chen, Sketch-based change detection: methods, evaluation, and applications, in: Proceedings of ACM SIGCOMM Internet Measurement Conference (IMC), Miami Beach, FL, USA, 2003.
[9] G. Liu, C. Ji, Scalability of network-failure resilience: analysis using multi-layer probabilistic graphical models, IEEE/ACM Transactions on Networking 17 (1) (2009) 319–331.
[10] K. Xu, Z. Duan, Z. Zhang, J. Chandrashekar, On properties of internet exchange points and their impact on AS topology and relationship, Lecture Notes in Computer Science 3042 (2004) 284–295.
[11] J. Liu, X. Zhang, B. Li, Q. Zhang, W. Zhu, Distributed distance measurement for large-scale networks, International Journal of Computer and Telecommunications Networking 41 (2003) 177–193.
[12] L. Xiao, K. Nahrstedt, Reliability models and evaluation of internal BGP networks, in: Proceedings of IEEE INFOCOM, Hong Kong, 2004.
[13] S. Kim, A.L.N. Reddy, Image-based Anomlay detection technique: algorithm, implementation and effectiveness, IEEE Journal on Selected Areas in Communications 24 (2006) 1942–1954.
[14] F. Xing, W. Wang, Modeling and analysis of connectivity in mobile ad hoc networks with misbehaving nodes, in: Proceedings of IEEE ICC, Istanbul, Turkey, 2006.
[15] D. Leonard, Z. Yao, V. Rai, D. Loguinov, On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks, IEEE/ACM Transactions on Networking, 15(5), (2007).
[16] N. Hu, L.E. Li, Z.M. Mao, P. Steenkiste, J. Wang, Locating internet bottlenecks: algorithms, measurements, and implications, in: Proceedings of ACM SIGCOMM, Portland, Oregon, USA, 2004.
[17] S. Birrer, F.E. Bustamante, Resilience in overlay multicast protocols, in: Proceedings of the 14th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Monterey, CA, 2006.
[18] D. Dumitriu, E. Knightly, I. Stoica, W. Zwaenepoel, Denial of service resilience in peer-to-peer file sharing systems, in: Proceedings of ACM SIGMETRICS, Banff, Alberta, Canada, 2005.
[19] K.T. Law, C.S. Lui, K.Y. Yau, You can run, but you can't hide: an effective methodology to traceback DDoS attackers, in: Proceedings of the 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Fort Worth, TX, USA, 2002.
[20] Y. Lin, B. Liang, B. Li, Data persistence in large-scale sensor networks with decentralized fountain codes, in: Proceedings of IEEE INFOCOM, Anchorage, AK, USA, 2007.
[21] S. Lee, Y. Yu, S. Nelakuditi, Z.L. Zhang, C.N. Chuah, Proactive vs. reactive approaches to failure resilient routing, in: Proceedings of IEEE INFOCOM, Hong Kong, 2004.
[22] B.Y. Zhao, L. Huang, J. Stribling, A.D. Joseph, J.D. Kubiatowicz, Exploiting routing redundancy via structured peer-to-peer overlays, in: Proceedings IEEE ICNP, Atlanta, GA, USA, 2003.
[23] J. Mirkovic, G. Prier, P. Reiher, Attacking DDoS at the source, in: Proceedings of IEEE ICNP, Paris, France, 2002.
[24] M. Ahuja, Y. Zhu, An efficient distributed algorithm for finding articulation points, bridges, and biconnected components in asynchronous networks, in: Proceedings of the Ninth Conference on Foundation of Software Technology and Theoretical Computer Science, Bangalore, India, 1989.
[25] Y.H. Tsin, Some remarks on distributed depth-first search, Information Processing Letters 82 (2002) 173–178.
[26] P. Keyani, B. Larson, M. Senthil, Peer pressure: distributed recovery from attacks in peer-to-peer systems, in: Proceedings of IFIP Workshop on Peer-to-Peer Computing, Pisa, Italy, 2002.
[27] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making Gnutella-like P2P systems scalable, in: Proceedings of ACM SIGCOMM, Karlsruhe, Germany, 2003.
[28] C. Tang, P.K. Mckinley, On the cost-quality tradeoff in topology-aware overlay path probing, in: Proceedings of IEEE ICNP, Atlanta, GA, USA, 2003.
[29] S. Ratnasamy, S. Shenker, S. McCanne, Towards an evolvable internet architecture, in: Proceedings of ACM SIGCOMM, Philadelphia, PA, USA, 2005.
[30] A. Ali, A. Mathur, H. Zhang, Measurement of commercial peer-to-peer live video streaming, in: Proceedings of Workshop in Recent Advances in Peer-to-Peer Streaming, Waterloo, Ontario, Canada, 2006.
[31] X. Liao, H. Jin, Y. Liu, L.M. Ni, D. Deng, AnySee: peer-to-peer live streaming, in: Proceedings of IEEE INFOCOM, Barcelona, Spain, 2006.
[32] Lakshmish Ramaswamy, Bugra Gedik, Ling Liu, A distributed approach to node clustering in decentralized peer-to-peer networks, IEEE Transactions on Parallel and Distributed Systems 16 (9) (2005) 814–829.
[33] H. Wang, T. Lin, On efficiency in searching networks, in: Proceedings of IEEE INFOCOM, Miami, FL, USA, 2005.
[34] S. Servetto, G. Barenechea, Constrained random walks on random graphs: routing algorithms for large scale wireless sensor networks, in: Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, GA, USA, 2002.
[35] C. Gkantsidis, M. Mihail, A. Saberi, Random walks in peer-to-peer networks, in: Proceedings of IEEE INFOCOM, Hong Kong, 2004.
[36] A. Iamnitchi, M. Ripeanu, I. Foster, Small-world file-sharing communities, in: Proceedings of IEEE INFOCOM, Anchorage, AK, USA, 2001.
[37] S. Milgram, The small world problem, Psychology Today 1 (1967) 61–67.
[38] T. Bu, D. Towsley, On distinguishing between internet power law topology generators, in: Proceedings of IEEE INFOCOM, New York, USA, 2002.
[39] DSS Clip2 Traces, <http://dss.clip2.com>, 2005.



Yuan He received his BE degree in Department of Computer Science and Technology from University of Science and Technology of China in 2003, and his ME degree in Institute of Software, Chinese Academy of Sciences, in 2006. He is now a PhD student in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology, supervised by Dr. Yunhao Liu. His research interests include peer-to-peer computing, sensor networks, and pervasive computing. He is a student member of the IEEE and the IEEE Computer Society.

Hao Ren received his BS degree, MS degree, and PhD degree in Department of Computer Science and Engineering from the National University of Defense Technology (NUDT) China, in 1996, 1999, and 2007, respectively. Currently he works in the Department of Computer Science and Engineering at NUDT. His research interests include peer-to-peer computing and grid computing.

Yunhao Liu (SM'06) received his BS degree in Automation Department from Tsinghua University, China, in 1995, and an MA degree in Beijing Foreign Studies University, China, in 1997, and an MS and a PhD degree in Computer Science and Engineering at Michigan State University in 2003 and 2004, respectively. He is now an assistant professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. He is also an adjunct professor of Xi'an Jiaotong University and Ocean University of China. His research interests include peer-to-peer computing, pervasive computing and sensor networks. He is a senior member of IEEE and a member of ACM.

Baijian Yang received his BS and MS degree in Department of Automation from Tsinghua University, China, in 1995 and 1998, respectively. He received his PhD degree in Computer Science from Michigan State University in 2003. He is currently an assistant professor in Department of Technology at Ball State University, Indiana, US. His research interests include distributed system, mobile computing and networking security. He is a member of IEEE and a member of ACM.