Optimizing Overlay Topology by Reducing Cut Vertices

Xiaomei Liu[1], Li Xiao[1], Andrew Kreling[1], Yunhao Liu[2]

[1]Department of Computer Science and Engineering, Michigan State University
[2]Deptartment of Computer Science, Hong Kong University of Science and Technology
{ liuxiaom, lxiao, krelinga}@cse.msu.edu, liu@cs.ust.hk

Abstract – **Overlay networks provide base infrastructures for many areas including multimedia streaming and content distributions. Since most overlay networks are highly decentralized and self-organized, cut vertices may exist in such systems due to the lack of centralized management. A cut vertex is defined as a network node whose removal increases the number of network components. Failure of these nodes can break an overlay into a large number of disconnected components and greatly downgrade the upper layer services like media streaming. We propose here a distributed mechanism, CAM, which efficiently detects the cut vertices before they fail and neutralizes them into normal overlay nodes with slight overhead so that the possibility of network decomposition is minimized after they fail. We prove the correctness of this algorithm and evaluate the performance of our design through trace driven simulations.**

1 Introduction

Features such as high flexibility and easy deployment enable overlay networks to provide support to a large variety of Internet applications, including multimedia streaming, online gaming, and publish/subscribe systems. Due to the lack of widespread IP multicast, multimedia streaming relies on the overlay infrastructure to implement the quick audio/video data distribution using overlay multicast [1-4]. In order to improve scalability and relieve the burden on servers, P2P overlay architectures are introduced in the media streaming (P2P streaming), the online gaming (P2P online gaming), and publish/subscribe services [5-9]. In these systems, end hosts self-organize into an overlay structure and the services are deployed along the overlay.

In order to provide qualified services, many applications require overlays to guarantee reliability and avoid network failures. For instance, the outgoing streaming may be disrupted when network failures occur [5]. In comparison with general nodes, the failure of "critical" nodes such as central servers is more likely to lead to network failures. Most overlay networks are highly decentralized. They remove the potential "critical" nodes caused by the service requirement since resources and services are provided by each node in the system. This, however, cannot remove "critical" nodes induced by the network topology. Overlay nodes are highly self-organized. They make connections either with some randomly selected nodes or via locally defined algorithms. In both cases, there is no centralized control to manage the network topology and thus the presence of topological "critical" nodes is unavoidable. S. Saroiu et al. [10] show that the failure of small amounts of high-degree nodes can efficiently "shatter" the overlay network, which makes the network highly vulnerable in the face of well-constructed, targeted attacks. In this paper, we discuss the influence of another type of "critical" nodes, cut vertices, in overlay networks.

Consider a network as an undirected graph. Cut vertices are such nodes whose deletion will create new components in the original graph. For a connected graph (component), removing cut vertices partitions the graph. In this paper, "graph", "component", and "vertex" are concepts defined in graph theory: a "graph" is used to represent a network; a "component" is a connected graph; and a "vertex" is another name for a node. Vertex and node will be used interchangeably in the remainder of this paper.

Traditional methods of detecting cut vertices require the global information of the network topology. These approaches work well if the network topology is not changed frequently and the scale of the network is from small to medium. This is not the case in most of the overlay networks. The end systems that compose an overlay network come and go very frequently [5, 10, 11]. This leads to high resilience of the overlay. Furthermore, the scale of an overlay network is expected to be huge, from several thousands to millions of nodes [2, 12]. A third factor is that most overlay networks lack the centralized control to maintain the global topology information due to their fully distributed feature.

In this paper, we propose CAM (*Connection Adjacency Matrix*): a fully distributed mechanism to detect cut vertices. Based on the CAM algorithm, each node in the system periodically sends out probe messages and decides whether it is a cut vertex based on the received feedback. CAM is composed of three stages: cut vertex detection, cut vertex computation, and cut vertex neutralization. At detection stage, a cut vertex candidate sends out component detection messages for each of its connections. If any two detection messages of different connections meet with each other, an arrival message is sent back to the message issuer. At computation stage, the candidate constructs a CAM graph in which nodes represent the connections of the candidate. If it receives an arrival message of two connections, the candidate will add an edge to the corresponding nodes in the CAM graph. The candidate decides whether it is a cut vertex based on its CAM graph. At the neutralization stage, CAM normalizes the detected cut vertex to a non-cut vertex.

The rest of the paper is organized as follows. In the next section, we review the related work. Section 3 describes the CAM algorithm. This is followed by the proof of the correctness of the algorithm in Section 4. Section 5 discusses the simulation methodology and the performance of CAM. We conclude the work in Section 6.

2 Related Work

Cut vertex is an important concept that has been introduced in graph theory and studied extensively. Existing algorithms to detect cut vertices in graph theory need to collect overall topology information of the network and construct a depth first search

(DFS) tree including all the network nodes. One category of such algorithms detects cut vertices by checking the sub-tree of each node in the DFS tree [13]. These algorithms are composed of the following operations. First, construct a DFS tree from any node. Then, for each node *v* in the DFS tree except the root, check the neighbors that *v*'s descendents connect with. If none of the neighbors of *v*'s descendents are *v*'s ancestors, *v* is a cut vertex. The root is a cut vertex if and only if it has more than one neighbor.

Another approach is to group the graph nodes into several bi-connected components [13]. A bi-connected component is a component that cannot be disconnected by deleting any vertex within the component. For any two vertices in a bi-connected component, there exist at least two disjointed paths between them. It is obvious that all the edge vertices that connect any two bi-connected components are cut vertices. In this approach, distinguishing disjointed paths and forming the bi-connected components require the traversal over the whole network. For instance, Sharir's algorithm for finding the bi-connected component builds a DFS tree involving all nodes in the graph [14].

Node failures caused by network resilience are widely noticed by the research community. Most recently proposed algorithms for overlay networks have addressed the problems [5, 15-19]. However, these algorithms treat all nodes the same way and do not pay special attention to "critical" nodes, whose failure may create more serious problems in the network than what these algorithms are able to handle.

High degree nodes can also be "critical" with respect to network topology. P. Keyani et al. [20] proposed a mechanism to modify the P2P overlay topology and reduce the number of high degree nodes. Their method focuses on how to reduce but not detect high degree nodes and is invoked when an attack occurs. The long convergence time of this method may make it impractical in a large-scale overlay network.

3 CAM: Distributed Cut Vertex Discovery

Consider an overlay network as a graph. The basic idea of CAM is to check whether this graph is still connected after a node is removed. If the graph is partitioned, the node is a cut vertex; otherwise, it is not a cut vertex. Recall that CAM is composed of three stages: cut vertex detection, cut vertex computation, and cut vertex neutralization. We present details of each stage in the rest of this section.

3.1 Cut vertex detection

A node in the system cannot be a cut vertex if it has zero or one connection. Otherwise, the node considers itself as a cut vertex candidate and initializes a cut vertex detection process. Before the detection, the candidate assigns a unique numerical identifier, starting with 1, to each of its connections/edges (we use the terms "connection" and "edge" interchangeably in the rest of this paper). This identifier is called the *connection number* of the connection. For example, if a candidate has *n* connections, it will label them from *1* to *n*.

At the beginning of the detection, the candidate sends a *component probe message* to each of its neighbors. The message contains the candidate's IP address, a timestamp, a TTL threshold, and the connection number of the edge that connects this neighbor with the candidate. Each node in the system has a *connection list*. There is one entry for each candidate in the connection list with the format of <candidate IP address, timestamp, connection number 1, connection number 2, …>. The node deals with the received message based on the information stored in the connection list. Upon receiving a message, one of the following situations may arise:

- The node has already received the message, or the message is old. The node simply just drops the message.
- There is no entry for the candidate that issues this message. The node creates an entry for it.
- The timestamp in the received message is newer than the one stored in the corresponding connection list entry. The candidate replaces the old time stamp and connection numbers stored in the connection list with the new ones.
- The timestamp of a recently received message is the same as the one stored in the corresponding connection list entry but the connection number of the message is not the same. The node adds the new connection number to the corresponding entry and sends an *arrival message* back to the candidate. Each arrival message contains two or more connection numbers and a timestamp. A node does not send any arrival messages until it receives probe messages from at least two different connection numbers.

A node forwards the message to all its neighbors except the message sender if the following conditions are met: this is a "new" message with the latest timestamp; the node did not issue any arrival message for the cut vertex candidate who issued this message; and the message's TTL has not expired.

Here we illustrate cut vertex detection shown in Figure 1. The candidate is a cut vertex. The initial CAM TTL value equals 3. Nodes reduce the TTL value by 1 each time right before they forward the probe messages to their neighbors. The connections to the candidate are labeled 1,2,3, and 4, respectively. In Figure 1(b), the candidate sends a probe message for each connection to nodes B, D, E, and G. Note that the TTL value has already been reduced by one by the candidate before it sends the probe messages to its neighbors. In Figure 1(c), nodes B, D, E, and G forward the received probe messages to other neighbors.

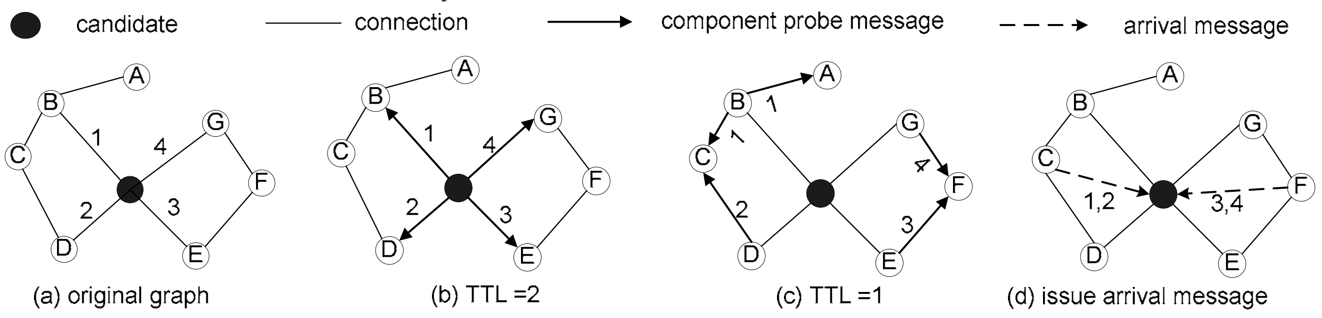
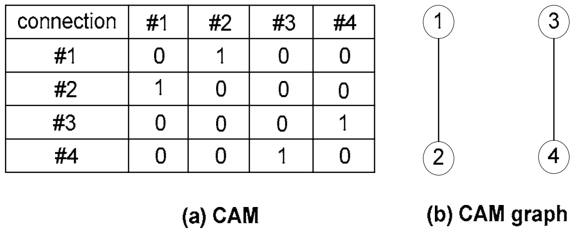


Figure 1 Cut vertex detection in CAM



connection	#1	#2	#3	#4
#1	0	1	0	0
#2	1	0	0	0
#3	0	0	0	1
#4	0	0	1	0

Figure 2 CAM and CAM graph

At this point, nodes C and F received probe messages from two distinct connection numbers. In Figure 1(d), node C sends back to the candidate an arrival message with connection numbers 1 and 2. Node F sends back to the candidate an arrival message with connection numbers 3 and 4.

3.2 Cut vertex computation

Each candidate maintains an arrival list and a $|c|$-by-$|c|$ binary matrix, where $|c|$ is the number of connections the candidate had when the CAM algorithm started. The format of the arrival list is similar to the connection list: <IP address, timestamp, connection number 1, connection number 2, …>. The IP address here is the IP address of the node that sends the arrival message back to the candidate. The binary matrix is called the candidate's *connection adjacency matrix* or *CAM*, whose row/column numbers represent the connection numbers of the candidate's connections.

If an arrival message including connection number x is received by the candidate from a network node v, the candidate will add x to the corresponding entry of v in the arrival list. For an entry (x, y) in CAM, where x is the row number and y is the column number, if the corresponding connection number of x and y can be found in the same entry of the arrival list, the value of this CAM entry is set to 1. Otherwise, the value is set to 0. In other words, if any node has sent back to the candidate an arrival message containing connection numbers x and y, a 1 is placed in the (x, y) and (y, x) entry of the candidate's CAM. After waiting an expected time out period, the candidate interprets its CAM as an adjacency matrix representation of an undirected graph, whose vertices are corresponding to the candidate's connections. This graph is called the candidate's *CAM graph*. An edge exists between node x and node y in the CAM graph if and only if the value of the CAM entry (x, y) is 1. If the candidate's CAM graph has more than 1 component, the candidate is a cut vertex. The CAM and the CAM graph of the candidate nodes in the previous example are shown in Figure 2.

3.3 Cut vertex neutralization

The process by which a cut vertex normalizes itself to a non-cut vertex is called *cut vertex neutralization*. Cut vertex neutralization is relatively trivial after cut vertices are detected: all that needs to be done in this process is to merge the disconnected components of a node's CAM graph into one connected component. Due to the page limitation of this paper, we illustrate the neutralization process by a simple neutralization mechanism. More sophisticated neutralization mechanisms can be constructed based on this basic one.

Consider a detected cut vertex v that has n CAM graph components $C_1, C_2, C_3, \ldots C_n$. At the beginning of neutralization, v randomly chooses one node from each component. Assume v selects $p_1, p_2, p_3, \ldots p_n$ from $C_1, C_2, C_3, \ldots C_n$. Based on the arrival list, v chooses overlay network nodes $o_1, o_2, o_3, \ldots o_n$, which have sent to v arrival messages of connections $p_1, p_2, p_3, \ldots p_n$ respectively. Then v sends a *connection message* to $o_1, o_2, o_3, \ldots o_n$ to indicate how the nodes should connect to each other, e.g., o_1 connects to o_2; o_2 connects to o_3; and so on. After constructing the new connections, the candidate is normalized to a non-cut vertex.

The network topologies of the aforementioned examples after cut vertex neutralization are shown in Figure 3.

3.4 Traffic overhead analysis

CAM improves the network reliability in two types of cost: the network traffic cost at the detection and the neutralization stages, and the computing cost at the cut vertex computation stage. Compared to the network traffic cost, the cost of the local computation based on a two-dimensional matrix is trivial for today's powerful end systems. Therefore, we focus on the traffic cost of CAM. Given an overlay network with n nodes, let c be the average number of connections a node can have, and let t be the TTL threshold in the CAM algorithm. If we ignore the difference of logical link cost, the traffic cost of CAM can be evaluated as follows.

At the cut vertex detection stage, a node will not forward any component probe message after it sends an arrival message back to the candidate. This means each node only forwards one component probe message for one specific connection during a CAM operation for one cut vertex candidate. We also know that, in CAM, component probe messages are not forwarded back to its (message) incoming neighbors. Therefore, if we name the set of nodes traversed by messages of the same connection number as the *traversal area* of that connection, the traversal area of different connections will not overlap. This suggests that the detection traffic cost of one candidate cannot be greater than $nc/2$. As the traversal area of each connection is also limited by t, the detection traffic cost should be $min(O(nc^t), O(n^2c/2))$.

Together with the fact that each node sends one arrival message in the worst case and every node in the system considers itself as a cut vertex candidate, the total traffic cost of the detection stage should be $min(O(nc^t+n), O(n^2c/2+n))$. At the neutralization stage, since each candidate at most sends out c neutralization messages, the traffic cost is $O(nc)$. The total traffic cost of CAM is $min(O(n(c^t+c+1)), O(n(nc+2c+2)/2)$. As c (in the order of ten) is generally much smaller than n (in the order of tens of thousands or millions), the total traffic cost of CAM can be simplified as $min(O(nc^t), O(n^2))$.

We need to adopt a small TTL value to reduce the cost of probing. In Section 5.2 , we will see that probe messages with a TTL value of two can achieve an accuracy rate of about 80% in a P2P system. In addition, as the average number of connections of most traces is from 3 to 4, c^t is trivial compared with n. Therefore, the traffic cost that all nodes execute CAM once in a P2P system is $O(n)$.

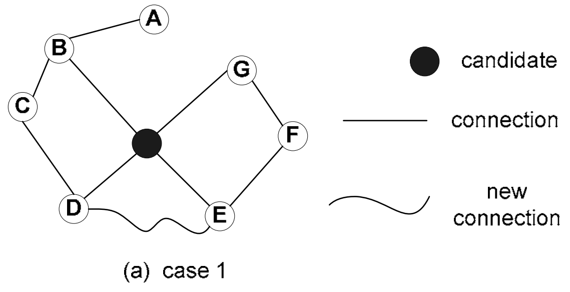


Figure 3 Cut vertex neutralization

4 Proof of Correctness

In this section, we present the proof of correctness for the CAM algorithm, starting with the definition of the system model and followed by the proofs.

4.1 System model and definitions

Consider a connected 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. Assume that the network topology is static and the network has unlimited resources. Thus, each node can issue the probe message with the TTL value set to infinity. In practice, we trade the accuracy for the traffic cost and set the TTL to a small value.

Definition 4.1: Given a graph G (V, E), a cut vertex candidate v_c refers to a vertex that tries to decide whether it is a cut vertex. The number of edges/connections that v_c has is referred to as $|c|$.

Definition 4.2: Given a graph G (V, E), a vertex v_p, and one of its connections $e_c = (v_p, v_q)$, assuming connection number of e_c is c. v_q is the neighbor of v_p that is connected by e_c. Let us remove v_p together with all its edges from G and get a new graph G' (V - v_p, E - $\{(v_p, v_i) \mid v_i \in V, (v_p, v_i) \in E\}$). The reachable set of e_c, $RS(e_c)$, is the set of vertices that can be reached in G' by a breadth first search (BFS) initiated by v_q. This search process is denoted as BFS(e_c). In other words, RS(e_c) contains the set of all vertices that can be reached by v_q via connection e_c. It is also important to note that there exists a vertex $v_q \in$ RS(e_c) such that $(v_p, v_q) \in$ E.

4.2 Proofs

Lemma 1: If there is an edge that connects two vertices in the candidate's CAM graph that represent connections e_a and e_b of graph G, then $RS(e_a) = RS(e_b)$.

Proof: The fact that there is an edge between the vertices representing connections e_a and e_b in the candidate's CAM graph implies that the values of entry (a, b) and (b, a) must be 1 in the CAM. This implies that there must exist some node v_p in the network that has received component probe messages for connections e_a and e_b. Since v_p received component probe messages for connections e_a and e_b, it is obvious that node v_p is traversed by both BFS(e_a) and BFS(e_b). According to the operation of BFS algorithm, after traversing node v_p, BFS(e_a) can reach all the nodes that BFS(e_b) can reach and vice versa. Therefore, RS(e_a) = RS(e_b).

Lemma 2: If the vertices that represent connections e_a and e_b are in the same component of the candidate's CAM graph, RS(e_a) = RS(e_b).

Proof: This is trivial given Lemma 1.

Lemma 3: If the vertices that represent connections e_a and e_b are not in the same component of the candidate's CAM graph, RS(e_a) $\neq$ RS(e_b).

Proof: We prove this by contradiction. Assume that the vertices that represent connections e_a and e_b are not in the same component of the candidate's CAM graph, but RS(e_a) = RS(e_b). Then there must exist at least one node that has not been reached by probe messages containing connection numbers a and b but can be reached by both BFS(e_a) and BFS(e_b). According to the operation of BFS, this only happens when the TTL has expired before the probe message arrives at the specific node. This contradicts the earlier assumption that the TTL values of probe messages are infinite. Therefore, the lemma is true.

Lemma 4: If RS(e_a) = RS(e_b) for any $1 \leq a, b \leq |c|$ of a cut vertex candidate v_c, then v_c is not a cut vertex.

Proof: RS(e_a) = RS(e_b) suggests that any node $v_i \in$ RS (e_a) must $\in$ RS(e_b). RS(e_a) = RS(e_b) for all $1 \leq a, b \leq |c|$ suggests this happens in any two reachable sets of v_c. In addition, for any nodes v_i and $v_j \in$ RS(e_a), there exists a path from v_i to v_j that does not include v_c. Therefore, removing v_c from G leaves one connected component. By definition, v_c is not a cut vertex.

Lemma 5: If RS(e_a) $\neq$ RS(e_b) for any $1 \leq a, b \leq |c|$, then v_c is a cut vertex.

Proof: Let G' be the graph that is formed by removing v_c and all its edges in G. RS(e_a) $\neq$ RS(e_b) implies that there does not exist an edge $(v_i, v_j) \in G'$ where $v_i \in$ RS(e_a) and $v_j \in$ RS(e_b). This implies that RS(e_a) and RS(e_b) are separate components of G'. By the definition of RS, the removal of v_c from G results in the number of components of G increasing by at least 1. Therefore, v_c is a cut vertex.

Theorem 1: If the candidate's CAM graph has more than one component, it is a cut vertex.

Proof: The candidate's CAM graph has more than one component. According to Lemma 3, we know that the RSs associated with the connection numbers whose vertex representations belong to different components in the CAM graph are not equal. Due to Lemma 5, we can easily conclude v_c is a cut vertex.

Theorem 2: If the candidate's CAM graph has one component, it is not a cut vertex.

Proof: From Lemma 2, we can deduce that RS(e_a) = RS(e_b) for any $1 \leq a, b \leq |c|$ when the candidate's CAM graph has one component. From Lemma 4, we can conclude v_c is not a cut vertex.

5 Performance Evaluation

We deployed a series of trace-driven simulations to evaluate the performance of CAM. Note that cut vertices exist in most overlay networks. We evaluate CAM based on P2P systems for two reasons. First, P2P systems are very popular today. Many applications including multimedia streaming and online gaming may be either based on P2P system or adopt a P2P based architecture. P2P traffic overwhelms web traffic on the Internet and has become the major consumer of Internet Bandwidth [21]. Second, the P2P system is a representative of the overlay networks: it is composed of self-governed end systems; it is autonomous and open; there is no central control server in a P2P

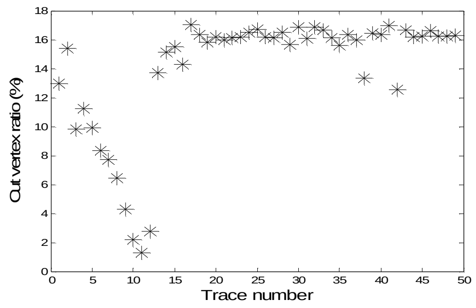


Figure 4 Cut vertex ratio of the traces

system; nodes can join and leave the system at any time; and collecting and maintaining overall topology information in a P2P system is hard, if not impossible.

5.1 Simulation setup

We used the DSS Clip2 traces that were collected from Dec. 7th 2000 to June 15th 2001 in our simulation. DSS Clip2 traces were available on http://dss.clip2.com, but are not available now. We can provide the traces to those who are interested upon request. There are 48 traces in this collection. The network sizes of the traces range from 225 to 47245. The average connections per node (node degree) of the traces are from less than 1 to 5. The cut vertex ratio for each trace ranges from about 1% of trace 7 to 17% of trace 17 as shown in Figure 4. It is obvious that the cut vertex ratio of traces collected after March 19th 2001 are more stable compared to previous ones, when the largest change in P2P software happened.

We have evaluated in the simulation the accuracy of CAM and its influence in the overlay topology. We checked the cut vertices in each trace using both traditional DFS based algorithm and CAM algorithm to check the accuracy of CAM. We then gradually removed the cut vertices and measured the overlay topology at the same time to check the influence of CAM in the overlay topology.

5.2 Accuracy of CAM detection

Three metrics are introduced for evaluating the *accuracy* of CAM: *CAM accuracy rate (CAR)*, *CAM false positive rate (CFPR)*, and *CAM false negative rate (CFNR). CAR* shows how many vertices detected by CAM as cut vertices are cut vertices. CFPR and CFNR show the error types made by CAM. CFPR shows how many nodes that are not cut vertices but are identified as cut vertices. CFNR shows how many nodes that are cut vertices are not detected as cut vertices. Assume that V is the set of all the vertices in a network, C is the set of all the cut vertices, and K is the set of all the vertices that are identified by CAM as cut vertices. The definitions of aforementioned metrics are given as follows:

$$CAR = \frac{|K \cap C|}{|K|} \tag{1}$$

$$CFPR = \frac{|(V - C) \cap K|}{|V - C|} \tag{2}$$

$$CFNR = \frac{|C \cap (V - K)|}{|C|} \tag{3}$$

We have checked the accuracy of all 48 traces and shown the average values of these three parameters in Figure 5. From these results, we can observe that the accuracy rate is over 70% even when the TTL value of CAM is 1. The accuracy rate is almost 100% when the CAM TTL value is set to 4. With the increase of the CAM TTL value, the CFPR keeps reducing and drops to almost zero when the TTL equals 4. At the same time, the CFNR remains as zero in all cases. This suggests that CAM can always successfully identify all the cut vertices, and CAM errors mainly result from the false alarms CAM reports when it considers non-cut vertices as cut vertices if the TTL is not large enough.

5.3 Influence on the overlay topology

We present here how CAM can affect the network topology. The metrics used for evaluating the influence of CAM in the overlay topology include the *number of components* and *ratio of new cut vertices*. The ratio of the new cut vertices refer to the cut vertices induced by the failure of nodes in the system. Due to the page limitation, we only representatively present the simulation results of trace 30, which has a relative large network size of 41,589, a typical average node degree of 3, and was collected after March 19th, 2001.

We compare the number of components in Figure 6. It shows that CAM greatly decreases the number of components induced by cut vertex failure. The ex-cut vertices here refer to the cut vertices in the overlay before cut vertex failures occur. The largest reduction is observed when CAM TTL equals 1. The reduction is rather close when CAM TTL is 2, 3 or 4. From Figure 5, we know CAM achieves the lowest CAR when TTL is set to 1. In fact, almost all nodes would be reported as cut vertices by CAM when TTL is set to 1, and thus after executing neutralization, the entire overlay topology would become a mesh. With the results shown in Figure 5 and aimed at reducing the detection cost, we recommend to set CAM TTL as 2 or 3.

Figure 7 shows how many new cut vertices might be produced when cut vertex failures occur. Note that CAM normalizes all the cut vertices to normal overlay nodes, and thus all the cut vertices in the network after CAM are the "new" cut vertices. When the TTL value is increased from 1 to 3, the generation rate of "new" cut vertices is also increased. However, the generation rates of TTL of 3 and TTL of 4 are very close. We also noticed that CAM with a TTL larger than 1 induces more

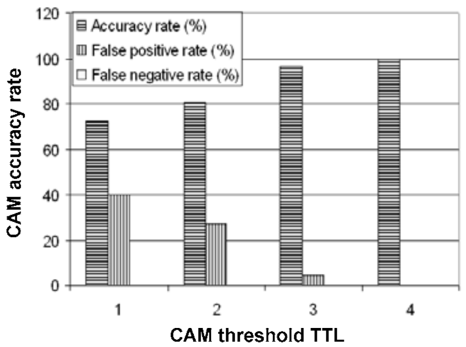


Figure 5 Accuracy rate of CAM

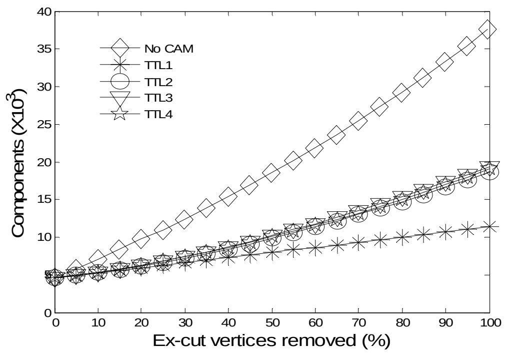


Figure 6 Component number increases after cut vertex failures

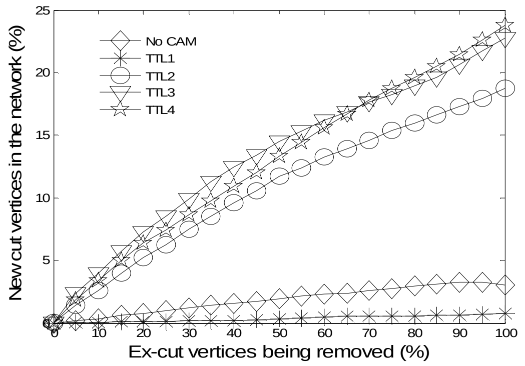


Figure 7 New cut vertices after ex-cut vertex failures

new cut vertices than in an overlay without deploying CAM. One possible reason is that the nodes connected with the new connections made by CAM, which prevent networks from partitioning, became new cut vertices.

6 Conclusion

In this paper, we have investigated the cut vertex failure problem and proposed a fully distributed mechanism, CAM, to detect cut vertices in overlay topology and improve the overlay reliability. As many applications, such as media streaming, rely on overlay networks, improving reliability can provide better service qualities of these applications. CAM can be applied to each node locally in an overlay networks. To our knowledge, we are the first to introduce a fully distributed cut vertex detection algorithm for nodes to detect whether they are cut vertex locally.

We prove the correctness of the algorithm and also evaluate its accuracy and influence in the overlay topology by trace-driven simulations. CAM can always successfully identify all the cut vertices. The detection traffic overhead can be restricted by setting a small CAM TTL value, which may mistake a small number of non-cut vertices as cut vertices. Our simulation shows that with a TTL threshold value as small as 2, CAM can obtain a fairly good accuracy rate of 80%. The accuracy rate increases to 96% when the TTL equals 3, and 99% when TTL equals 4. After being detected, the cut vertices can be normalized to non-cut vertices. We propose a basic neutralization mechanism to normalize cut vertices to non-cut vertices in this paper. In the future, we will propose optimized mechanism to neutralize cut vertices. We also plan to demonstrate the effectiveness of CAM in other overlay network systems.

7 Acknowledgement

This work was supported in part by the US National Science Foundation under grants CCF-0325760, CCF-0514078, CNS-0549006, and CNS 0551464.

8 Reference

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *in the proceeding of ACM SIGCOMM*, 2002.

[2] Y. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early Experience with an Internet Broadcast System Based on Overlay Multicast," *in the proceeding of USENIX Annual Technical Conference*, 2004.

[3] Y. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," *in the proceeding of ACM SIGMETRICS*, 2000.

[4] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," *in the proceeding of INFOCOM*, 2003.

[5] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient Peer-to-Peer Streaming," *in the proceeding of ICNP*, 2003.

[6] A. S. John and B. N. Levine, "Supporting P2P Gaming When Players Have Heterogeneous Resources," *in the proceeding of NOSSDAV*, 2005.

[7] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On Peer-to-Peer Media Streaming," *in the proceeding of ICDCS*, 2002.

[8] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "AnySee: Scalable Live Streaming Service Based on Inter-Overlay Optimization," *in the proceeding of INFOCOM*, 2006.

[9] Y. Choi and D. Park, "Mirinae: A Peer-to-Peer Overlay Network for Large-Scale Content-based Publish/Subscribe Systems," *in the proceeding of NOSSDAV*, 2005.

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

[11] S.Sen and J. Wang, "Analyzing Peer-to-peer Traffic Across Large Networks," *in the proceeding of ACM SIGCOMM Internet Measurement Workshop*, 2002.

[12] "Limeware", *www.limeware.org,*

[13] F. Buckley and M. Lewinter, "*A Friendly Introduction to Graph Theory"*: Prentice Hall, 2002.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "*Introduction to Algorithms"*: McGraw-Hill & MIT Press, 2001.

[15] S. Yuen and B. Li, "Strategy Proof Mechanisms for Dynamic Multicast Tree Formation in Overlay Networks," *in the proceeding of INFOCOM*, 2005.

[16] C. Abad, W. Yurcik, and R. H. Campbell, "A Survey and Comparison of End-System Overlay Multicast Solutions Suitable for Network-Centric Warfare," *in the proceeding of SPIE Defense and Security Symposium/BattleSpace Digitalization and Network-Centric Systems IV*, 2004.

[17] X. Liu, Y. Liu, and L. Xiao, "Reliable Response Delivery in Peer-to-Peer Systems," *in the proceeding of MASCOTS,* 2004.

[18] Y. Liu, X. Liu, L. Xiao, L. Ni, and X. Zhang, "Location-Aware Topology Matching in Unstructured P2P Systems," *in the proceeding of INFOCOM'04*, 2004.

[19] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, "A Distributed Approach to Solving Overlay Mismatching Problem," *in the proceeding of ICDCS*, 2004.

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

[21] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy, "An Analysis of Internet Content Delivery Systems," *in the proceeding of OSDI*, 2002.