Tie-set Based Fault Tolerance for Autonomous Recovery of Double-Link Failures

Kiyoshi Nakayama, Kyle E. Benson, Vahe Avagyan, Michael B. Dillencourt, Lubomir F. Bic, Nalini Venkatasubramanian
Department of Computer Science, University of California, Irvine, CA 92697-3425, USA
Email: kiyoshi.nakayama@ieee.org, {kebenson, vavagyan}@uci.edu, {dillenco, bic, nalini}@ics.uci.edu

***Abstract*—In this paper, we propose a mechanism for coping with double-link failures in an autonomous and distributed manner. We call it Tie-set Based Fault Tolerance (TBFT) because it utilizes *tie-sets*, which represent a set of the edges comprising a loop within the graph that represents the network. An autonomous distributed control method based on dividing a network into a set of tie-sets, whose union covers every edge in the network, has been verified to be more effective than traditional tree-based restoration techniques in case of single link failure. The proposed method efficiently and gracefully handle double-link failures and also decrease the communication overhead incurred during network configuration. We demonstrate these results by simulating and comparing TBFT with the traditional approach of using Rapid Spanning Tree Protocol (RSTP).**

***Keywords*-graph theory, tie-set, loop, double-link failures, fault tolerance, autonomous recovery**

I. Introduction

Double-link failures happen when two links failed simultaneously within a distributed system. While it is extremely unlikely for two links to truly fail at precisely the same time, double-link failures include two single-link failures that occur in such rapid succession that the system cannot determine which occurred first. These events are likely to transpire during geographically correlated physical phenomena such as natural disasters (e.g. earthquakes), cosmic events (e.g. solar storms), and deliberate attacks (e.g. low-altitude nuclear detonations). This type of failure is more difficult for a distributed system to recover than a single-link case.

Many methods have been proposed to solve link failure problems by routing the messages in a fault tolerant manner. Spanning Tree Protocol (STP) [1] is a technique that builds a spanning tree within a mesh network of connected Ethernet switches and disables those links that are not part of the spanning tree, leaving a single active path between any two network nodes. Rapid Spanning Tree Protocol (RSTP) [2] provides significantly faster spanning tree convergence when failures happen [3], [4]. The authors of [5], [6] show that RSTP is applicable to recover failures in ring topology as well.

Starting from SONET rings [7], [8], various failure recovery schemes have been developed based on Pre-configured cycles (p-cycle) by applying cycle covering schemes such as [9], [10], [11]. These techniques allow fast recovery and do not require significant capacity redundancy. Some mechanisms based on p-cycles have been proposed to protect span and node failure [12], networks from multiple sequential [13], simultaneous link failures [14], and both at once [15]. While protection methods based on p-cycles have led to sophisticated techniques and solutions, "preconfigured" cycles cannot dynamically restructure the formation of cycles. This becomes crucial when solving multiple-link failures, especially in large-scale systems and when one cycle contains more than one failure.

Similar to p-cycles, tie-sets logically divide the graph representation of the physical network into a set of cycles with dynamic configuration in a "distributed" manner. A core idea of distributed control method to handle a single-link failure based on tie-set graph theory [16], [17] was introduced in [18], [19]. The tie-set based restoration method for single link failure was a simple approach and was not able to handle over one point link failure as the recovery scheme of more than one failure requires a much more sophisticated analysis, problem formulation, and synchronization mechanisms. Tie-set Based Fault Tolerance, so called TBFT in this paper, demonstrated a superiority over the traditional tree-based technique with carefully designed detection and synchronization techniques. Furthermore, the strength of this technique can be found in its decentralized nature and stability in case of failure as it can cope with failures within overlapping loops. Therefore, TBFT is also expected to demonstrate a substantial stability and fast recovery when multiple link failures occur at the same time.

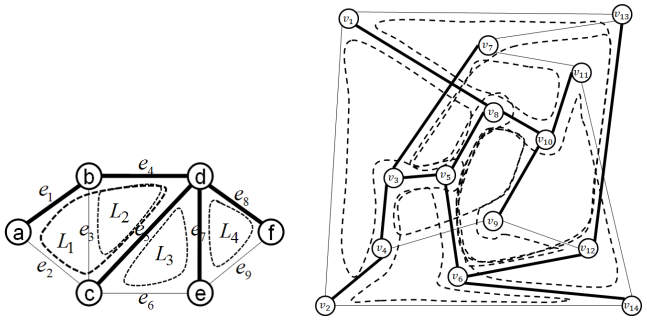
In this paper, we propose a Tie-set Based Fault Tolerance (TBFT) algorithm for double-link failures, which is a novel approach against the scheme using Dijkstra as in RSTP. The remainder of the paper is organized as follows. Section II-A discusses tie-set graph theory for the benefit of the reader. Section III presents the improved algorithm for configuring the network with tie-set information. Section IV contains the algorithm we developed for recovering from double-link failures. In section V, we present our simulation results and compare them with those of RSTP. In the final section, we conclude the paper and discuss future works.

II. Tie-sets and State Information

In this section, we provide a brief review of tie-set graph theory and state information of a node.

A. Introduction to Tie-set Graph Theory

For a given bi-connected and undirected graph $G = (V, E)$ with a set of vertices $V = \{v_1, v_2, ..., v_n\}$ and a set of edges $E = \{e_1, e_2, ..., e_m\}$, let $L_i = \{e_1^i, e_2^i, ...\}$ be a set of all the edges that constitutes a loop in G. The set of edges L_i is called a *tie-set* [21]. Let T and $\overline{T}$ respectively be a spanning tree and



(a) In a planar graph (b) In a non-planar graph

Fig. 1. Examples of a fundamental system of tie-sets

a cotree of G, where $\overline{T} = E - T$. For $l \in \overline{T}$, $T \cup \{l\}$ includes one tie-set. Focusing on a subgraph $G_T = (V, T)$ of G and an edge $l = (a, b) \in \overline{T}$, there exists only one elementary path P_T whose origin is b and terminal is a in G_T. Then, a *fundamental circuit* that consists of the path P_T and the edge l is uniquely determined as $C(l) = (a, l = (a, b), P_T(b, a))$. The tie-set $L(l)$ corresponding to $C(l)$ is referred to as a *fundamental tie-set*. It is known that $\mu = |\overline{T}|$ fundamental tie-sets exist in G, and they are called a *fundamental system of tie-sets* $\mathbb{L}_B = \{L_1, L_2, ...L_\mu\}$. A fundamental system of tie-sets guarantees that it covers all the vertices and edges even in a bi-connected and non-planar graph G as shown in Fig. 1.

B. State Information of a Node

Each node v_i mainly has three types of information as state information as follows:

1) Incident Links $E(v_i)$: Link information connected to v_i.

2) Adjacent Nodes $V(v_i)$: Node information connected through incident links of v_i.

3) Tie-set Information $\mathbb{L}(v_i)$: Information of fundamental tie-sets to which v_i belongs. When a fundamental tie-set L_i contains e_i that includes v_i in its two vertices, it is defined that v_i belongs to L_i and has information of L_i.

III. Distributed Algorithm for Tie-set Information Configuration (DATIC)

As described in II-B, each node has information of fundamental tie-sets to which the node belongs so as to solve any problems within some loops. Although a distributed algorithm for recognizing Tie-set Information was proposed in [18], we propose a more efficient Distributed Algorithm for Tie-set Information Configuration (*DATIC*).

A. Mechanism of DATIC

Communication paths are represented by a set of links of a spanning tree T, which we call "tree links". On the other hand, other links are considered as a set of links of a cotree $\overline{T}$, which we call "cotree links". We can create a tree by exploiting a spanning tree algorithm (STA) [22].

As in Fig.1, thick edges stand for tree links, whereas thin edges stand for cotree links.

We define a *Find Tie-set Message (FTM)* to obtain the information of a fundamental tie-set (Tie-set Information in state information of a node). FTM contains the following information:

- *EdgeTable:* A set of links through which a FTM passed
- *NodeTable:* A set of nodes through which a FTM passed

To obtain Tie-set Information $\mathbb{L}(v_i)$, each node v_i conducts DATIC using FTMs described as follows.

Initialization: Each node v_i checks its incident links $E(v_i)$. For any cotree link $l = (v_i, v_j) \in E(v_i)$, v_i negotiates with v_j to decide which node conducts DATIC for Tie-set Information of $L(l)$ based on Node Priority such as Physical Address of a node.

Let v_o be a node that executes DATIC to construct Tie-set Information of $L(l)$ where $l \in \overline{T}$. v_o creates a FTM, and then sends the FTM to the node v_j, where v_j is included in a set of two vertices of a cotree link $l = (v_o, v_j)$. When sending a FTM to v_j, v_o adds node information of v_o to *NodeTable* and link information $e(v_o, v_j)$ to *EdgeTable*.

From here, let v_r be a node that receives a FTM. After receiving a FTM, v_r executes different procedure by the following cases.

Case 1: $v_r \neq v_o$ If v_r ($\neq v_j$) is a leaf[1], v_r discards the FTM. If v_r is not a leaf or v_j, v_r copies the FTM and sends the copied FTM to adjacent nodes that are connected through tree links of $E(v_r)$. When sending a copied FTM to an adjacent node v_a that is connected through a tree link, v_r adds node information of v_r to *NodeTable*, and adds link information $e(v_r, v_a)$ to *EdgeTable*.

Case 2: $v_r = v_o$ In this case, the FTM has passed through a fundamental tie-set $L(l)$ in a network. The information of *EdgeTable* and *NodeTable* included in the FTM is stored in v_o. v_o also notifies the Tie-set Information of $L(l)$ to the nodes that are included in *NodeTable* of the received FTM by just passing the FTM around on a tie-set $L(l)$.

If failure occurs in the middle of DATIC, the node that detects the failure broadcasts failure notification messages, which correspond to Bridge Protocol Data Units (BPDUs) in Spanning Tree Protocol, in order to notify the root node of it. Then, the root creates a spanning tree again followed by DATIC above.

B. Complexity Analysis of DATIC

Here, we discuss the Communication Complexity and Time Complexity of DATIC from the perspective of a distributed algorithm [22].

1) Time Complexity of DATIC: Analyzing the Time Complexity of DATIC is simple as a FTM is passed through one cotree link and several tree links until a FTM is stored in an original node that sent the FTM or discarded at a leaf node. Therefore the maximum number of times a FTM will be forwarded is equal to D, the diameter of the graph, and therefore the Time Complexity of DATIC is $O(D)$ as FTMs are sent simultaneously.

[1]A leaf x exactly has one tree link in its incident links and other incident links are cotree links.

2) Communication Complexity of DATIC: The Communication Complexity is the total number of messages produced by the entire distributed system to configure Tie-set Information. Only one node a or b of each $l = (a, b) \in \overline{T}$ sends a FTM on l, and there are μ such links in the graph, where μ is the nullity of the graph, as described in Section II-A. Because the FTM is copied and sent along only tree links, it will be copied at most $|V| - 1$ times. Therefore, DATIC completes its procedures with Communication Complexity of $O(\mu|V|)$.

IV. Tie-set Based Fault Tolerance (TBFT) for Double-Link Failures

In this section, we analyze the theoretical aspect of double-link failures on the basis of tie-set graph theory and propose an algorithm to cope with double-link failures.

A. Problem Formulation for Double-Link Failures with TBFT

A *double-link failure* is defined as two physical or logical links failing at the same time, or at least close enough in time that the system cannot tell the difference. Let e_i^f and e_j^f be two failed links where failures happen on e_i and e_j ($e_i \neq e_j$), respectively. A set of two failed links is denoted as $E^f = \{e_i^f, e_j^f\}$. Let $\mathbb{L}(e_i^f)$, $\mathbb{L}(e_j^f)$ respectively denote the classes of tie-sets that contain failed links e_i^f, e_j^f.

Here, we further denote $\mathbb{L}^f$, $\mathbb{L}(e_i^f, e_j^f)$, and $\overline{\mathbb{L}}(e_i^f, e_j^f)$ as follows:

$$\mathbb{L}^f = \mathbb{L}(e_i^f) \cup \mathbb{L}(e_j^f) \tag{1}$$

$$\mathbb{L}(e_i^f, e_j^f) = \mathbb{L}(e_i^f) \cap \mathbb{L}(e_j^f) \tag{2}$$

$$\overline{\mathbb{L}}(e_i^f, e_j^f) = \mathbb{L}(e_i^f) \oplus \mathbb{L}(e_j^f) \tag{3}$$

$\mathbb{L}^f$ is the class of all the tie-sets that include link failure(s). $\mathbb{L}(e_i^f, e_j^f)$ is the class of tie-sets that include both of the two failed links. $\overline{\mathbb{L}}(e_i^f, e_j^f)$ represents the class of tie-sets that share exactly one failed link with tie-sets that have two failed links. By these definitions, the following equation holds true.

$$\mathbb{L}^f = \mathbb{L}(e_i^f, e_j^f) \cup \overline{\mathbb{L}}(e_i^f, e_j^f) \tag{4}$$

Double-link failures with TBFT are classified into two major cases depending on the locations of the two failed links.

1) Double-Link Failures are Independent $\mathbb{L}(e_i^f, e_j^f) = \emptyset$: When double-link failures happen on two failed links E^f where $\mathbb{L}(e_i^f, e_j^f) = \emptyset$, it is defined that double-link failures are *independent* in TBFT. In this case, double-link failures can be considered as a pair of single-link failures that can be solved with the method for single-link failure. According to TBFT, one tie-set $L_i^f \in \mathbb{L}(e_i^f)$ with the failed link e_i^f handles the failure by replacing the failed link e_i^f with the co-tree link in the tie-set L_i^f. Similarly, tie-set $L_j^f \in \mathbb{L}(e_j^f)$ recovers the failure on the link e_j^f by replacing e_j^f with the cotree link in L_j^f. Fig.2 demonstrates an example of the behavior of TBFT in this case. The double-link failures are handled independently in different tie-set L_i^f and L_j^f in an autonomous manner.

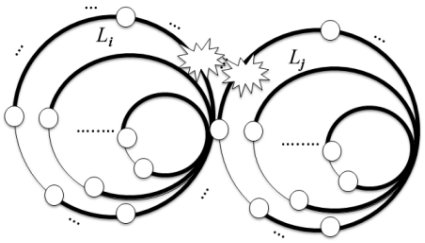
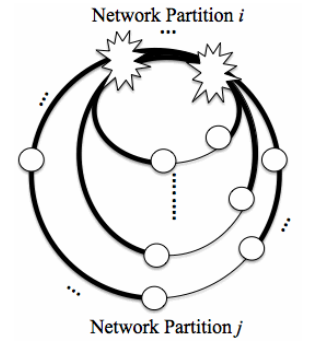
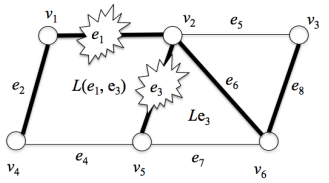


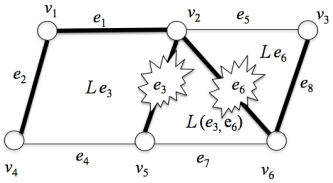
Fig. 2. The case where double-link failures are independent: $\mathbb{L}(e_i^f, e_j^f) = \emptyset$



(a) $\overline{\mathbb{L}}(e_i^f, e_j^f) = \emptyset$



(b) $|\overline{\mathbb{L}}(e_i^f, e_j^f)| = 1$

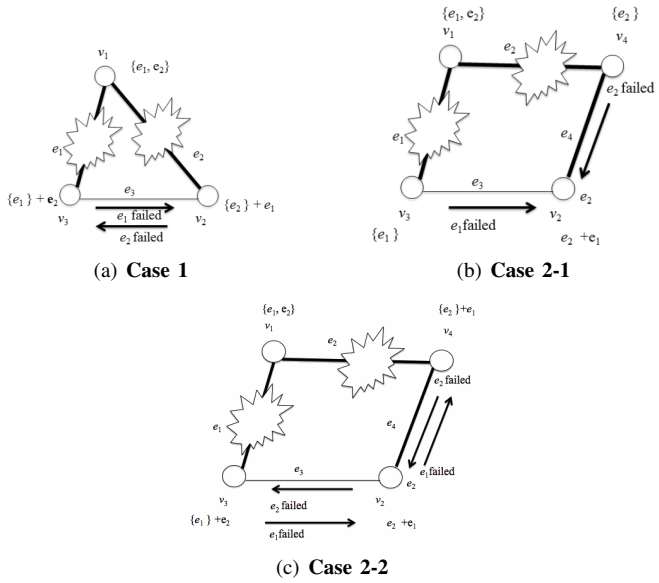


(c) $|\overline{\mathbb{L}}(e_i^f, e_j^f)| = 2$

Fig. 3. The case where double-link failures are dependent: $\mathbb{L}(e_i^f, e_j^f) \neq \emptyset$

2) Double-Link Failures are Dependent $\mathbb{L}(e_i^f, e_j^f) \neq \emptyset$: When double-link failures happen on two failed links E^f where $\mathbb{L}(e_i^f, e_j^f) \neq \emptyset$, it is defined that double-link failures are *dependent* in TBFT. In this case, there is at least one tie-set that contains both of the two failed links E^f. There are two sub-classes focusing on $\overline{\mathbb{L}}(e_i^f, e_j^f)$.

- $\overline{\mathbb{L}}(e_i^f, e_j^f) = \emptyset$
  In this case, double-link failures cannot be restored. As $\mathbb{L}^f = \mathbb{L}(e_i^f, e_j^f)$, all the failed tie-sets $\mathbb{L}^f$ contain both of the two failed links. The whole system is divided into two network partitions, where there is no way to recover this type of double-link failures as messages cannot be redirected between separated network partitions.
- $\overline{\mathbb{L}}(e_i^f, e_j^f) \neq \emptyset$



(a) **Case 1** (b) **Case 2-1**

(c) **Case 2-2**

Fig. 4. Examples of Failure Information Sharing within a Tie-set

In this case, double-link failures can be restored. As at least one tie-set that shares only one failed link among failed tie-sets $\mathbb{L}^f$ is available so systems can choose a tie-set $L^r \in \overline{\mathbb{L}}(e_i^f, e_j^f)$ to recover one of the double-link failures and then restore another failure subsequently.

An example of double-link failures where $\overline{\mathbb{L}}(e_i^f, e_j^f) = \emptyset$ is described in Fig. 3(a). As seen in Fig. 3(a), the system is divided into two partitions i and j. Naturally, no technique can possibly cope with this type of failure.

Examples of double-link failures where $\overline{\mathbb{L}}(e_i^f, e_j^f) \neq \emptyset$ are described in Fig. 3(b) and 3(c). In Fig. 3(b) where $\overline{\mathbb{L}}(e_i^f, e_j^f) \neq \emptyset, |\overline{\mathbb{L}}(e_i^f, e_j^f)| = 1$, there is a tie-set $L_{(e_1,e_3)}$ in which two failed links conflict to use the cotree link of the tie-set. The tie-set with only one failed link L_{e_3} must recover its failed link first. This resolves the conflict of failed links in the tie-set with two failed links. In Fig. 3(b), $\mathbb{L}(e_1) = \{L_{(e_1,e_3)}\}$, $\mathbb{L}(e_3) = \{L_{(e_1,e_3)}, L_{e_3}\}$, and $\overline{\mathbb{L}}(e_1, e_3) = \{L_{e_3}\}$. In this example, the tie-set L_{e_3} should be responsible for the recovery of the failed link e_3. $L_{(e_1,e_3)}$ should be responsible for the recovery of e_1.

In Fig. 3(c) where $\overline{\mathbb{L}}(e_i^f, e_j^f) \neq \emptyset, |\overline{\mathbb{L}}(e_i^f, e_j^f)| = 2$, there are two tie-sets that share exactly one failed link with a tie-set that has two failed links. One of the tie-sets with only one failed link is responsible for the recovery of its failed link. In the Fig. 3(c), as $\mathbb{L}(e_3) = \{L_{e_3}, L_{(e_3,e_6)}\}$, $\mathbb{L}(e_6) = \{L_{(e_3,e_6)}, L_{e_6}\}$, and $\overline{\mathbb{L}}(e_3, e_6) = \{L_{e_3}, L_{e_6}\}$, tie-sets L_{e_3} and L_{e_6} recover the failed links e_3 and e_6 separately.

B. Failure Information Sharing within a Tie-set

In order to implement this algorithm, there must be a communication mechanism that allows nodes within a failed tie-set L_i^f that includes failed link(s) to share information about the failure. Therefore, we design a procedure to share the double-link failure information within a failed tie-set L_i^f. Let a *Failure Information Sharing Message (FISM)* be a message that includes failure information that a node detected. A FISM includes the information about the failed link $e_f \in E^f$, tie-sets to which the failed link belongs $\mathbb{L}(e_f)$, and an address table of $L_i^f \in \mathbb{L}(e_f)$.

This procedure should consider the following points.

- A node with a failed link e_f sends FISMs to its neighbors within each $L_i^f \in \mathbb{L}(e_f)$.
- The neighbor(s) that receive a FISM store the information about the failure and transfer the FISM further within the tie-set L_i^f. If a node receives another FISM about a second failure, it just adds the new information to the already stored information and also sends FISMs to its neighbors within each $L_i^f \in \mathbb{L}(e_f)$.

After this procedure, all the connected nodes in L_i^f obtain information about failed links.

Fig. 4(a) shows that v_1 already knows both failed links on e_1 and e_2 by failure the detection mechanism described later in the next section. v_2 knows there is a link failure on e_2; then, v_2 passes a FISM ("e_2 failed") to its neighbor v_3. Similarly, v_3 passes a FISM ("e_1 failed") to v_2. Both of them receive and store the FISMs. In Fig. 4(b), only v_1 and v_2 have the information of double-link failures. Then, v_2 shares the information to v_3 and v_4 as shown in Fig. 4(c).

C. Failure Detection Details

Here, let us describe the algorithm in more detail. In order to recognize link status, each node v_i sends a *SYN* message to an adjacent node v_a in a certain direction and receives an *ACK* response from an adjacent node $v_a \in V(v_i)$. The direction may be decided by the sorted address table stored in Tie-set Information of V_i, such as *NodeTable*. This message passing should be done periodically within some time period t_{syn} to check if the link to the adjacent node v_a is alive. After v_i sends a *SYN* message to v_a, v_i expects to receive an *ACK* response back from v_a during a time period t_{ack}. This procedure enables failure detection on a link connected to v_a.

The length of the time period t_{syn} and t_{ack} depends on the topology and physical characteristics of a distributed system. Defining the time period t_{syn} and t_{ack} is to be an issue that should be carefully considered when designing a switch.

In the failure detection procedure, after v_i sends a *SYN* message to v_a, two cases are possible as follows:

1) When v_i receives a response from v_a during time period t_{ack}: In this case, the link between v_i and v_a is undamaged, since an *ACK* message comes back to v_i from v_a.

2) When v_i does not receive a response from v_a during time period t_{ack}: In this case, the time period t_{ack} has passed and v_i does not receive an *ACK* response from v_a. The node v_i concludes that the link to v_a has failed. The same conclusion is made by node v_a if it does not receive a *SYN* message from node v_i either. After that, both v_i and v_a start sending FISMs to available neighbors within the tie-sets $\mathbb{L}(e(v_i, v_a))$. Let v_f be a node that detects the failed link $e_f \in E^f$. If several tie-sets that include e_f exist for v_f, they are informed about the failure as well. When a node receives a FISM, it waits a time

period t_f to receive another FISM about the any other failures. On this step, two cases are possible as follows:

- Case 1: A node does not receive a FISM about another failure during time period t_f. In this case, no additional failure happens or failures happen in different tie-sets. In case of double-link failures, this case corresponds to the case where $\mathbb{L}(e_i^f, e_j^f) = \emptyset$.
- Case 2: A node receives a FISM about another failure during time period t_f. This case indicates that both failures happen in the same tie-set. This case corresponds to the case where $\mathbb{L}(e_i^f, e_j^f) \neq \emptyset$. A node updates stored information about those failed links.

After time period t_f, all nodes in tie-sets $\mathbb{L}^f$ have the information about failures. In the case of simultaneous double-link failures, a tie-set is divided into two isolated parts. Both failed links have two connected nodes that are aware of failure. This means that each part of the tie-set has nodes that inform about both failures. The next step is to decide which tie-set recovers which failure.

In order to assign responsibilities to recover link failure(s) among all tie-sets with failure(s) $\mathbb{L}^f$, each node analyzes the failure information and makes a decision on its own. After a time period t_f, a node v_i has all information about failures. Therefore, a node v_i is able to analyze all tie-sets that have at least one failed link and designate a tie-set for recovery of each of the failures. Tie-sets that have only one failed link are designated to recover it. Tie-sets with double-link failures are designated to recover links that are not overlapped with other tie-sets.

The node v_i takes responsibility to recover a failed link $e_f \in E^f$ only if both of the following conditions hold true:

1) Node v_i is connected to the cotree link of L_i^f that contains e_f.
2) L_i^f is designated for recovery of e_f.

If a failed link e_f has more than one designated tie-set, a node v_i will take responsibility for recovery if it is in the tie-set with the smallest number of nodes. If ties happen, any of the tie-sets can be chosen to recover through some deterministic method of prioritization, such as the lowest numbered MAC address among all nodes. Hence, each tie-set recovers a failure independently from other tie-sets, which means there is no need to notify about the decision made to recover a particular failure as well as about the completion of the recovery.

D. Detection and Recovery Algorithms

Let v_f be a node that detects a link failure and e_f be a failed link. Distributed algorithm for link failure consists of two parts: the algorithm for a node v_f that detects a failure on link e_f (Algorithm 1) and the algorithm that is executed by a node v_i that received a FISM (Algorithm 2).

We could come up with several ideas to pick a tie-set up where route switching is executed in Step 2 in Algorithm 2; the total value of link weights of a tie-set, the number of hops to a cotree link, the size of a tie-set, among others depending on the nature of a network. The number of hops to a cotree link is considered in this paper.

Algorithm 1 Distributed algorithm in a failed node v_f

STEP 1: *Blocking physical ports connected to* e_f
v_f blocks its physical port connected to e_f.
STEP 2: *Selecting failed tie-sets* $\mathbb{L}(e_f)$
v_f selects tie-sets $\mathbb{L}(e_f) = \{L_i^f\}$ from Tie-set Information $\mathbb{L}(v_f)$ where $e_f \in L_i^f$, and includes information of failed tie-sets $\mathbb{L}(e_f)$ in a FISM.
STEP 3: *Sharing information about the failed link* e_f
v_f sends a FISM with information about the failed link e_f and failed tie-sets $\mathbb{L}(e_f)$ to adjacent nodes in each tie-set $L_i^f \in \mathbb{L}(e_f)$.

Algorithm 2 Distributed algorithm in a node v_i that received a FISM

STEP 1: *Receiving a FISM*
if v_i receives a FISM from v_a **then**
Store the failure information about e_f and $\mathbb{L}(e_f)$.
Send a copied FISM to the next node $\neq v_a$ in the tie-set L_i^f.
Wait a time period t_f in order to receive a FSIM about another failure.
end if
STEP 2: *Analyzing failure information to make decision*
Using failure information after a time period t_f, do the following:
if L_i^f includes only e_f **then**
Designate L_i^f to recover e_f.
else $\{L_i^f$ includes double link failures $E^f\}$
if $e_f \in E^f$ belongs only to L_i^f **then**
Designate L_i^f to recover e_f.
end if
end if
After designating responsibility to L_i^f, do the following:
if L_i^f is designated to recover e_f and v_i is connected to cotree link of L_i^f **then**
Take responsibility to recover e_f.
end if
STEP 3: *Opening physical ports connected to cotree link*
if v_i takes responsibility to recover e_f **then**
Open port to cotree link.
end if

V. Simulation and Experiments

We created a simulation with Java to verify the behavior of the recovery method for link failure suggested in this paper and compared TBFT with RSTP on behalf of existing technologies because of its general use. A tool which demonstrates RSTP is created in reference to IEEE standards 802.1D [2]. In configuring a network, links are set to be undirected so that data can flow bi-directionally. In addition, the network is designed to be redundant, in other words, bi-connected, so as to be and able to cope with failures. For node configurations, each node was assigned input ports and output ports, a message

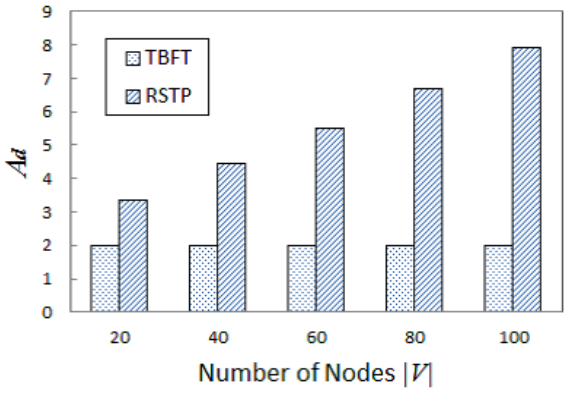


Fig. 5. The average number of times of route switching

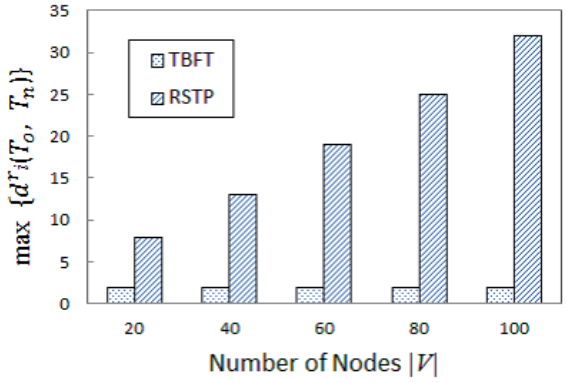


Fig. 6. The maximum number of times of route switching

buffer, and a processor. A common buffering method is used in a simulated node, where all messages received through input ports go to the message buffer. The processor takes each message from the message buffer via a polling method. After each message is processed, the message is sent to other nodes through appropriate output ports unless it is discarded or this node is its final destination.

A. Route Switching Points

One of the serious problems in RSTP is that topology of tree greatly changes when failure happens in the vicinity of a root bridge requiring many times of route switchings. Naturally, many times of route switchings lead to instability in communications.

On the basis of these points, experiments to measure the number of times of route switchings required to restore double-link failures are conducted to compare against RSTP. A spanning tree that represents communication paths before double-link failures is denoted as T_o, and a renewed tree that represents communication paths after link failures is denoted as T_n. The number of route switching points can be quantified using the distance between T_o and T_n. The distance is defined as follows:

$$d(T_o, T_n) = |T_o - T_n| \tag{5}$$

Let $d_{(i,j)}(T_o, T_n)$ be the distance when link failures occur on tree links $\{e_i, e_j\} \subseteq T, (e_i \neq e_j)$. In total, ${}_\rho\mathrm{C}_2$ of double-link failures can be conceivable where $\rho(= |T|)$ is a rank of a graph. Let $d^r_{(i,j)}(T_o, T_n)$ be the distance when double-link failures can be restored; the set of failed edges E^f does not correspond to the case where $\mathbb{L}(e^f_i, e^f_j) \neq \emptyset$ and $\overline{\mathbb{L}}(e^f_i, e^f_j) = \emptyset$.

If R combinations of restored points exist, then the average of the number of route switching points A_d is defined as follows:

$$A_d = \frac{\sum d^r_{(i,j)}(T_o, T_n)}{R} \tag{6}$$

For a given bi-connected and undirected graph $G = (V, E)$, a graph G is created at random with the number of nodes $|V|$ ranging from 20 to 100. A tree is output by giving link costs at random, and executing the Spanning Tree Algorithm (STA). Fig.5 is the experimental results that show the average A_d of the number of route switching points. As shown in Fig.5, RSTP requires about more switchings than TBFT on average when double-link failures happen, while TBFT needs only two time switchings. In addition, A_d shows modest upward tendency in proportion to the size of a network.

Fig.6 shows the maximum times of route switchings ($\max\{d^r_i(T_o, T_n)\}$) for each given graph whose condition is the same as the experiment to measure A_d. While TBFT constantly needs two switchings, RSTP requires much more switchings than the proposed method. This is because RSTP greatly changes its tree topology in case of failure in the vicinity of a root bridge. Failure near a root node is the biggest problem in operation of RSTP. The remarkable tendency of augmentation of the number of times of route switching is seen in a large-scale network. In that case, throughput degrades seriously as well as convergence time greatly increases making an entire network unstable.

B. Communications Overhead

Under the same conditions, we measured the communications overhead, which is the number of total messages that are used in restoring double-link failures. Communications overhead directly affects the stability of a network as individual messages may change the physical communications port states, which temporally interrupts data transfer between bridges. Furthermore, transmitting these messages decreases the amount of bandwidth available to any applications running on the distributed system. As these applications may be mission-critical, which is a reasonable assumption given that we are concerned with rapid fault recovery, they should be disrupted as little as possible even by rare events such as link failures.

Let $N_m(e_i, e_j)$ be the total number of messages used in restoring double-link failures when link failures occur on tree links $\{e_i, e_j\} \subseteq T, (e_i \neq e_j)$. The message used in RSTP when recovering failures is called Bridge Protocol Data Unit (BPDU), whereas TBFT uses FISMs to recover failure. Let $N^r_m(e_i, e_j)$ be the total number of messages when double-link

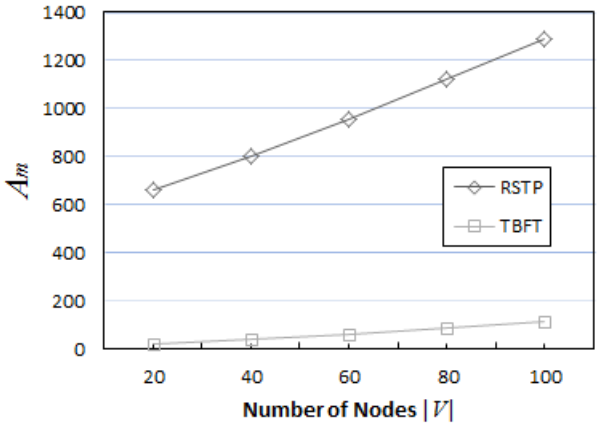


Fig. 7. The average communications overhead

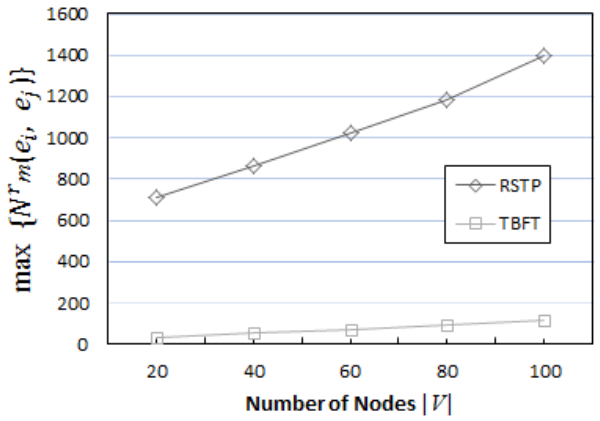


Fig. 8. The maximum communications overhead

failures can be restored. Then, the average number of total messages A_m is expressed as follows:

$$A_m = \frac{\sum N^r_m(e_i, e_j)}{R} \tag{7}$$

Fig.7 and 8 show the average number of total messages A_m and the maximum number of total messages $N^r_m(e_i, e_j)$. As shown in Fig.7 and 8, TBFT greatly reduces the total number of messages to restore double-link failures over RSTP. Therefore, the recovery mechanism will have less visible impact on the performance of the system than RSTP.

VI. Conclusion

In this paper, Tie-set Based Fault Tolerance (TBFT) is proposed for distributed autonomous restoration of double-link failures. We provided detailed descriptions of the algorithms required for configuring a network with tie-set information and for detecting and recovering from failures. We demonstrated that TBFT can realize faster recovery and more efficient operation than tree-based restoration schemes through simulation experiments and comparisons with the Rapid Spanning Tree Protocol (RSTP).

We are currently investigating whether this technique can be extended to an arbitrary number of link failures. This result, if found to be true, will be presented in future work. Another important question that we are exploring is the relation between how the initial spanning tree is chosen and the structure of the resulting tie-sets chosen by DATIC. We will continue to work towards optimizing these algorithms for eventual deployment in an emulated distributed system environment.

References

[1] Perlman, Radia (1985). *An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN* ACM SIGCOMM Computer Communication Review 15 (4): 44E53.

[2] IEEE Computer Society Sponsored by the LAN/MAN Standards Committee, *IEEE Standards 802.1D*, IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges, 9 June 2004.

[3] J. Qiu, Y. Liu, G. Mohan, K.C. Chua, *Fast Spanning Tree Reconnection for Resilient Metro Ethernet Networks*, IEEE International Conference on Communications (ICC), 2009, pp. 1 - 5.

[4] J. Qiu, G. Mohan, K.C. Chua, Y. Liu, *Handling Double-Link Failures in Metro Ethernet Networks Using Fast Spanning Tree Reconnection*, Proceedings of the 28th IEEE conference on Global telecommunications (GLOBECOM), 2009, pp. 3593-3598.

[5] M. Pustylnik, M. Vukotic, R. Moore, RuggedCom, Inc., *Performance of the Rapid Spanning Tree Protocol in Ring Network Topology* Available online.

[6] D. DesRuisseaux, *Use of RSTP to Cost Effectively Address Ring Recovery Applications in Industrial Ethernet Networks*, Presented at the ODVA 2009 Conference and 13th Annual Meeting, February 25, 2009, Howey-in-the-Hills, Florida USA.

[7] http://www.sonet.com/EDU/upsr.htm, *Understanding SONET UPSRs*, Available online.

[8] http://www.sonet.com/EDU/blsr.htm, *Understanding SONET BLSRs*, Available online.

[9] Wasem OJ., *An algorithm for designing rings for survivable fiber networks*, IEEE Transactions on Reliability 1991; 40:428-439.

[10] L. M. Gardner, M. Heydari, J. Shah, I. H. Sudborough, I. G. Tolis, and C. Xia, *Techniques for finding ring covers in survivable networks*, in Proc. IEEE GLOBECOM, San Francisco, CA, Nov. 1994, pp.1862-1866.

[11] C. Thomassen, *On the complexity of finding a minimum cycle cover of a graph*, SIAM Journal on Computing 1997; 26(3):675-677.

[12] G.Shen, and W.D.Grover, *Extending the p-Cycle Concept to Path Segment Protection for Span and Node Failure Recovery* IEEE Journal on Selected Areas in Communication, Vol. 21, No. 8, 2003.

[13] D. A. Schupke, W. D. Grover, and M. Clouqueur, *Strategies for Enhanced Dual Failure Restorability with Static or Reconfigurable p-Cycle Networks* IEEE International Conference on Communications (ICC 2004), June 2004.

[14] T. Fenga, L. Longb, A.E. Kamalb, and L. Ruana, *Two-link failure protection in WDM mesh networks with p-cycles* Computer Networks Volume 54, Issue 17, 3 December 2010, pp. 3068 - 3080.

[15] Hongsik Choi, Subramaniam, S., Hyeong-Ah Choi., *On double-link failure recovery in WDM optical networks*, IEEE International Conference on Computer and Communications (INFOCOM), 2002, Vol. 2.

[16] N. Shinomiya, T. Koide, and H. Watanabe, *A theory of tie-set graph and its application to information network management*, International Journal of Circuit Theory and Applications 2001; 29:367-379.

[17] T. Koide, H. Kubo, and H. Watanabe, *A study on the tie-set graph theory and network flow optimization problems*, International Journal of Circuit Theory and Applications 2004; 32:447-470.

[18] K. Nakayama, N. Shinomiya, *An autonomous recovery for link failure based on tie-sets in information networks*, Proc. of IEEE Symposium on Computers and Communications (ISCC), 2011, pp. 671-676.

[19] K.Nakayama, N.Shinomiya, H.Watanabe, *Distributed Control for Link Failure Based on Tie-Sets in Information Networks*, Proceedings of 2010 IEEE International Symposium on Circuits and Systems; pp.3913-3916.

[20] Swamy MNS, Thulasiraman K., *Graphs, Networks, and Algorithms*, Wiley Interscience: New York, 1981.

[21] Iri M, Shirakawa I, Kajitani Y, Shinoda S etc., *Graph Theory with Exercises*, CORONA Pub: Japan, 1983.

[22] Nancy A. Lynch, *Distributed Algorithms*, Morgan Kaaufmann Publishers, Inc.: San Francisco, California, 1996.