International Journal of Advanced Trends in Engineering and Technology (IJATET)
Impact Factor: 5.665, ISSN (Online): 2456 - 4664
(www.dvpublication.com) Volume 2, Issue 2, 2017

MULTIPATH ROUTING AND DUAL LINK FAILURE RECOVERY IN IP FAST REROUTING

M. Mohanapriya* & P. Kavitha**
Department of Computer Science, Sri Ramakrishna College of Arts and Science, Coimbatore, Tamilnadu

Cite This Article: M. Mohanapriya & P. Kavitha, "Multipath Routing and Dual Link Failure Recovery in IP Fast Rerouting", International Journal of Advanced Trends in Engineering and Technology, Volume 2, Issue 2, Page Number 164-170, 2017.

Abstract:

In this approaches for disjoint multipath routing and fast recovery in IP networks that guarantee recovery from arbitrary two link failures. Developing the first known algorithm to construct three edge-independent spanning trees, which has a running time complexity of $0(V^2)$. The property of these trees is that the paths from a source to the destination on the trees are mutually link-disjoint. We illustrate how the three edge-independent trees rooted at a destination may be employed to achieve multipath routing and IP fast recovery. In implement different ways of employing the trees. The routing of packets is based on the destination address and the input interface over which the packet was received. If the trees are employed exclusively for multipath routing, then no packet overhead is required. If the trees are employed for failure recovery, then the overhead bits will range from 0 to 2 bits depending on the flexibility sought in routing. In this project evaluate the performance of the trees in fast recovery by comparing the path lengths provided under single- and dual-link failures with an earlier approach based on tunneling.

Key Words: Independent Spanning Trees, Ip Fast Reroute, Multilink Failure Recovery & Multipath Routing.

1. Introduction:

As mentioned before, multipath routing can provide a range of benefits. In the section we describe how these benefits are achieved, and give an overview of the main elements in multipath routing protocols.

Benefits of Multipath Routing:

Fault Tolerance – Multipath routing protocols can provide fault tolerance by having redundant information routed to the destination via alternative paths. This reduces the probability that communication is disrupted in case of link failure. More sophisticated algorithms employ source coding to reduce the traffic overhead caused by too much redundancy, while maintaining the same degree of reliability. This increase in route resiliency is largely depended on metrics such as the diversity, or disjointness, of the available paths. We delay the discussion on disjoint routes until the next section.

Load Balancing – When a link becomes over-utilised and causes congestion, multipath routing protocols can choose to divert traffic through alternate paths to ease the burden of the congested link.

Bandwidth Aggregation – By splitting data to the same destination into multiple streams, each routed through a different path, the effective bandwidth can be aggregated. This strategy is particular beneficial when a node has multiple low band width links but requires a bandwidth greater than an individual link can provide. End-to-end delay may also be reduced as a direct result of larger bandwidth.

Reduced Delay – For wireless networks employing single path on-demand routing protocols, a route failure means that a new path discovery process needs to be initiated to find a new route. This results in a route discovery delay. The delay is minimised in multipath routing because backup routes are identified during route discovery.

There is a growing need for developing efficient endto- end protocols for the Future Internet, specifically ones that can exploit multipath routing. One of the key technical challenges identified in the following:

"The outstanding technical issue with transport-based multipath is how to distinguish flows to ensure their routes diversify as soon as they enter the internetwork."

Multipath routing (MPR) is an effective strategy to achieve robustness, load balancing, congestion reduction, and low power consumption. Disjoint multipath routing provides increased security and bandwidth compared to non disjoint multipath routing as link- or node-disjoint paths are employed. Despite the advances in multipath routing research, the use of multipath routing in IP networks is mostly limited to equal-cost multi paths (ECMP). Recently, some sophisticated routers offer multipath routing, however they are limited to two kinds: 1) source-based forwarding, which provides only single-path routing for a source; and 2) forwarding port selection on a per-packet basis, which leads to high variance in the end-to-end delay, and may lead to significant throughput reduction for TCP traffic. Thus, we need an efficient mechanism to route traffic over multiple paths, ideally disjoint, in order to avoid contention for bandwidth.

On the other hand, the Internet is prone to link failures on an everyday basis, be it due to planned maintenance or unplanned outages. As the data rates increase, the amount of data lost due to temporary service disruption increases. To ensure fast recovery from failures, the rerouting schemes must have the following characteristics: 1) proactive recovery—whereby the backup forwarding ports are calculated *a priori*; 2) local

recovery initiated by the node next to the failed link, rather than the source; and 3) local recovery from a link failure without the knowledge of other failures, in case of multiple link failures.

Traditional routing in Internet Protocol (IP) networks involves computing a forwarding link for each destination referred to as the *primary (preferred) forwarding link*. When a packet is received at a node, it is forwarded along the primary forwarding link corresponding to the destination address in the packet. To recover from the failure of the primary forwarding link, a node must reroute the packet over a different link, referred to as the *backup forwarding link*. The backup forwarding link at different nodes in the network must be chosen in a consistent manner to avoid looping.

1.1 Significance of Three Edge-Independent Trees:

The implication conjecture thought to be closed in and opened up again by our work in remains an open problem at this time. Based on our experience in developing the counterexample, it is not apparent if a general approach to compute edge-independent spanning trees can be derived just from the corresponding vertex-independent spanning trees without the knowledge of how the vertex-independent spanning trees were constructed. In this paper, we provide an algorithm to construct three edge-independent spanning trees in three-edge-connected graphs. Given that a generic algorithm for deriving edge-independent trees from vertex-independent trees is not known as of this writing, the construction of three edge-independent trees in itself is a significant contribution due to the applications of three edge-independent spanning trees in networks, as explained below.

The computation of three edge-independent trees is key to deriving the fundamental results in *network tomography*. Network tomography refers to the area of networking where individual link characteristics are inferred by observing end-to-end path characteristics. For example, in we showed that three-edge connectivity is a necessary and sufficient condition for identifying additive link metrics using measurement cycles, where all cycles traverse a given measurement node. The polynomialtime algorithm to construct linearly independent cycles/paths between a given set of measurement nodes relies on the construction of three edge-independent trees rooted at the measurement node. Moreover, given that the linearly independent cycles/paths were constructed using trees, we may completely solve for all the link metrics in linear time without having to perform matrix inversion. In a follow-up paper, we compute the maximum achievable link rank in an arbitrary topology and identify the set of links that result in rank deficiency. The proof of identifying links within certain network components, again, relies on the construction of three edge-independent trees. Note that there are several works in the literature that attempt to compute linearly independent cycles in a brute-force approach, without having a knowledge of what is the maximum achievable link rank for a given placement of monitor nodes. The above works provide the fundamental theoretical knowledge in network tomography, which is made possible only through the construction of three edge-independent trees.

Problem Statement:

Multipath routing (MPR) is an effective strategy to achieve robustness, load balancing, congestion reduction, and low power consumption. Disjoint multipath routing provides increased security and bandwidth compared to non disjoint multipath routing as link- or node-disjoint paths are employed. The Internet is prone to link failures on an everyday basis, be it due to planned maintenance or unplanned outages. As the data rates increase, the amount of data lost due to temporary service disruption increases. To ensure fast recovery from failures, the rerouting schemes must have the following characteristics: 1) proactive recovery—whereby the backup forwarding ports are calculated a priori; 2) local recovery initiated by the node next to the failed link, rather than the source; and 3) local recovery from a link failure without the knowledge of other failures, in case of multiple link failures. Traditional routing in Internet Protocol (IP) networks involves computing a forwarding link for each destination, referred to as the primary (preferred) forwarding link. When a packet is received at a node, it is forwarded along the primary forwarding link corresponding to the destination address in the packet. To recover from the failure of the primary forwarding link, a node must reroute the packet over a different link, referred to as the backup forwarding link. The backup forwarding link at different nodes in the network must be chosen in a consistent manner to avoid looping.

2. Related Works:

2.1 Expansion to a Cubic Graph:

This step is to transform the given graph into a three-vertex connected graph. The expansion to a cubic graph, where every vertex in the graph has degree three, makes it easier to understand the construction procedure and facilitates the computation of the edge-independent trees, which will be evident in Section IV-E. Any graph where vertices have degree greater than three may be expanded to a cubic graph in a trivial manner. However, the expansion procedure must also guarantee that the cubic graph is three-vertex-connected in order to be able to construct the sequence of paths with special properties (as outlined above in Step 2). We achieve this by iteratively expanding every node in the original graph into a set of nodes that have degree three, while retaining the three-edge connectivity of the network. We finally show that a three-edge-connected cubic graph is also three-vertex-connected (see Lemma 1 in the Appendix).

Fig. 3 shows the procedure to expand a given 3E-2V graph into a three-vertex-connected cubic graph. The algorithm works by expanding a vertex with degree into subvertices, denoted by through. The subvertices are internally connected as a path using additional edges. The edges connected to vertex are spread across the vertices such that subvertices and have two edges connected to them, while all other subvertices have exactly one edge connected to them. The choice of edges to be connected to vertices and are based on the connectivity of the graph in the absence of vertex.

Consider an intermediate stage of the graph where some vertices may have already been expanded. Call this graph. Assume that is three-edge-connected and there exists a vertex with degree higher than three. We now expand vertex. The algorithm works by removing the vertex and all the edges connected to it. The resultant graph is then divided into two-edge-connected (2E) components. Observe that the graph formed by the 2E-components is one-edge-connected. If the resultant graph has only one 2E component, then it must have at least three edges connecting the component to the vertex. These edges may be distributed to the subvertices in any manner as long as each subvertex has degree exactly three. If the resultant graph has more than one 2E-component, then every leaf component6 must have at least two edges connecting to vertex. In addition, every component that is connected to exactly two other components must have at least one edge connected to vertex. We select two leaf components, say and. From each of these two leaf components, we select two edges. We attach the first edge from to and the second edge from to We repeat this for. Thus, vertices and have exactly degree three, and the two selected leaf components, and, have one edge attached to each and. Any remaining edges connecting to may be assigned to the subvertices in an arbitrary order provided each subvertex has degree exactly three.

Procedure Cubic Expansion

1) For every vertex n whose degree d is more than three, do:
   a) Remove vertex n and all the edges connected to it. Let the resultant graph be $\mathcal{G}_{int} \setminus \{n\}$
   b) Sub-divide the vertex into $d-2$ sub-vertices, denoted by n_1 through n_{d-2}. Connect vertex n_i to n_{i+1}, where $i = 1, 2, \ldots, d-3$.
   c) Divide $\mathcal{G}_{int} \setminus \{n\}$ into 2E-components.
   d) If $\mathcal{G}_{int} \setminus \{n\}$ has only one 2E-component, then assign the edges connecting this component to n arbitrarily such that; vertices n_1 and n_{d-2} have two edges connected, while all other sub-vertices have one edge connected.
   e) If $\mathcal{G}_{int} \setminus \{n\}$ has more than one 2E-component,
      i) Select two arbitrary leaf components, say C_1 and C_2. Select two edges from each of these leaf components. Connect these edges to sub-vertices n_1 and n_{d-2} such that one edge from each of these two components is connected to each n_1 and similarly to n_{d-2}.
      ii) Connect any remaining edges from any of the components to n in any order such that the degree of every sub-vertex is exactly three.

Algorithm to expand a given three-edge- and two-vertex-connected graph into a three-edge-connected cubic graph

2.2 Constructing Augmenting Cycles/Paths:

Given a three-connected cubic graph and a root vertex, we decompose the network into a sequence of paths. The first path is a cycle that starts and ends at vertex. Every other path starts and ends at distinct vertices. The cycle and the paths satisfy the following properties.

1) The removal of vertices in the path keeps every other vertex not added in this path or earlier paths connected with each other.

2) Every vertex in the path is connected to at least one vertex that is not added to this path or earlier paths. The first condition implies the *nonseparating* nature of the path. The second condition may be met in several ways. The algorithm in [22] simply decomposes the network into a sequence of nonseparating cycle and paths that also satisfy the second property above in a specific manner.7 For the sake of completeness and ease of understanding of the rest of the paper, we briefly discuss the path augmentation technique employed by [22] to construct three vertex-independent trees rooted at , but tailored to the cubic graph expansion. This is shown in Fig. 5. It is important to note that we are not interested in computing three vertex-independent trees. We obtain the cycle and paths from [22] and modify these paths to obtain cycle/paths in the original graph.

1) Select an arbitrary neighbor of r, say u. Remove the edge r–u.
2) Compute a cycle starting and ending at r and avoiding u such that: (1) the removal of the cycle keeps every other vertex not in the cycle connected to u; and (2) every vertex in the cycle has one neighbor that is not added to the cycle.
3) $i = 0$.
4) If $\bigcup_{j=0}^{j=i} P_j$ does not include all vertices in $\mathcal{V}^*$, do:
   a) $i \leftarrow i+1$.
   b) Compute a path P_i avoiding u, i.e.: (1) every vertex in the path P_i has one neighbor in $\mathcal{Q} \setminus \bigcup_{j=0}^{j=i} P_j$; and (2) every vertex in $\mathcal{Q} \setminus \bigcup_{j=0}^{j=i} P_j$ remains connected to u.
   c) Repeat.
5) Stop.

Figure outlines the steps involved in the path augmentation procedure. We identify any arbitrary neighbor of, say, and remove the edge - . We then decompose this graph into a sequential ordering of paths. The first path is a cycle as it will start and end at. At every stage, the path starts and ends at two distinct vertices that

are already part of some earlier paths and traverses at least one new vertex. The last path augmented will have as the only new vertex added.

3. Challenges and Contributions:

3.1 Multipath Evaluation:

We now evaluate the performance of the three link-independent trees when employed for multipath routing in addition to being used for fast recovery from arbitrary dual-link failures. To evaluate our scheme, we benchmark it against the popular technique ECMP. For the simulation, we build a shortest directed acyclic graph (DAG) rooted at each destination. Every node has one (or more) forwarding neighbors on the ECMP DAG and splits both the incoming traffic and the traffic that it is sourcing uniformly among these equal-cost next-hops. For the traffic model, we assume that every node in the network sends 1 unit of traffic to every other node. For the three link-independent trees, traffic is equally divided among the three trees by the source node. At intermediate nodes, incoming traffic on a particular tree is forwarded along the same tree.

With the simulation setup described above, we now evaluate the traffic carried by every directed link in the network in the topologies considered. 14 shows the histograms of link loads for different networks. The links are ordered (in descending order) based on the traffic carried by them when ECMP is employed. We observe that links typically carry more traffic with independent trees compared to ECMP. This is not surprising as the three-trees approach provides disjoint paths for all nodes at the expense of longer path lengths compared to ECMP. In all these networks, the total bandwidth carried in the network is approximately 70% more than that with ECMP.

When we consider traffic for a specific destination (or in scenarios of having skewed or partial traffic matrices), the distribution of traffic on the links could be quite different. Shows the ordered link loads for destination node 1 in the ARPANET network. As we see in this example, the maximum load carried by a link in ECMP could be higher than that with independent trees.11 since the path lengths using the three trees are longer than the shortest path, most links carry higher traffic when compared to ECMP.

To the best of our knowledge, the edge-independent spanning tree is the only scheme that can achieve both multipath routing and fault tolerance from arbitrary dual-link failures simultaneously. Another aspect where the fully disjoint paths come in handy is against intrusion detection since any compromised link would only be seeing one third of the traffic from any given source in the network. Contrast this with ECMP where there is no guarantee of disjointedness12 of the paths from a source to a destination node.

3.2 Forwarding Tree Selection in a Protection Graph:

Consider a packet destined to node d with address d0 encounters a failure at node x, where the default forwarding link x–y. Node x stacks a new header to the packet with the destination address as yg. The packet may now be transferred either along the red or blue tree. There are two approaches to select the default tree over which the packet is routed.

The first approach is referred to as the red tree first (RTF), where every packet is forwarded along the red tree. Upon failure of a red forwarding link in the protection graph, the packet will be forwarded along the blue tree. When a blue forwarding link fails, the packet is simply dropped as it indicates that the packet has already experienced two link failures1.

The second approach is referred to as the shortest tree first (STF), where a packet is forwarded along that tree which provides the shortest path to the root of the tree. As the packets are first forwarded on the shortest tree, the packets experience lower delays under single link failure scenarios. While the red tree may offer the shortest path for node x in the protection graph Gyg, the blue tree may offer the shortest path for another node x0 in the same protection graph, where x; x0 2 Nyg. A packet that is forwarded on the red (blue) tree will be re-routed to the blue (red) tree upon a red (blue) forwarding link failure. The limitation of this approach is that it may result in perennial looping if more than two links fail in the network. Unlike the RTF approach, where a packet to be forwarded on the blue link implies that it has already experienced two link failures, the STF approach does not provide any implicit indication on the number of failures experienced by the packet. We will employ an additional bit that denotes the number of failures the packet has encountered in a protection graph. When forwarded on the shortest-path tree, the bit is set to 0. Upon the failure of the forwarding link on the first tree, the packet is forwarded on the other tree with the bit set to 1. Upon failure of a forwarding link in the protection graph, a packet is dropped if the bit is set to 1.

3.3 Forwarding Search Token:

A node that has the path search token attempts to augment a path through each of its neighbor. The node then forwards the token to those eligible neighbors, traversing the ordered list in the reverse direction (opposite to the order in which the SEARCH messages were initiated), one at a time. An eligible neighbor is one for which the variable flag new node added is set to true. Such an order reversal for passing the token helps maintain a consistent global ordering in a distributed manner across all the nodes in the network. A node that receives a TOKEN changes its state from CYCLE to TOKEN, starts the path search along each of its neighbors, and forwards the token to its eligible neighbors.

Once the tokens are returned by all neighbors, the node sets its state to FINISH and returns the token to the node from which it first received the token. The token finally reaches the drain, indicating that all nodes in the network are in the FINISH state, at which point the algorithm terminates.

4. Methodology Overview:

4.1 Dataset and Experiment Setups:

Theorem 1: The expansion procedure retains the three-edge connectivity property of the graph and eventually results in a three-vertex-connected graph.

Proof: Note that as the original graph is two-vertex-connected, removal of vertex keeps connected. After an expansion at any stage, the graph still retains the 2V property. This can be checked by seeing that the removal of any vertex keeps the graph connected. Thus, it is sufficient to show that the algorithm retains three-edge connectivity at every vertex expansion stage. We show that after every vertex expansion, the removal of any edge will still leave the graph two-edge-connected. We define a property called overlap on the edges connected to the expanded vertices of n. Consider only the leaf components in $\mathcal{G}_{\text{int}} \setminus \{n\}$. Divide these components into two arbitrary groups, say D_1 and D_2. The smallest index of the expanded vertex n that D_i is connected to is referred as $\min_i$ and the largest index of the expanded vertex n that D_i is connected to is referred as $\max_i$. We define the interval of connection of the two groups as $(\min_1, \max_1)$ and $(\min_2, \max_2)$. We say that D_1 and D_2 overlap if these two intervals overlap (or one is completely contained in the other). If the two intervals are disjoint, then we say that the two groups do not overlap.

Let $\mathcal{I}_n$ denote the edges that connect two subvertices of n. Consider a edge ℓ. We have two possible scenarios: 1) $\ell \notin \mathcal{I}_n$; or 2) $\ell \in \mathcal{I}_n$. In both scenarios, we show that the graph after edge removal is two-edge-connected.

Case 1: Since edge $\ell \notin \mathcal{I}_n$, we consider three subcases. First, let ℓ be within one of the two-edge-connected components C_i. Before n is expanded, we have a 3E-2V connected graph. Any vertex v in C_i has three link-disjoint paths to n. Now removal of ℓ in C_i can affect at most one of these three disjoint paths. Hence, v still has at least two edge-disjoint paths to n. Therefore, v has two edge-disjoint paths to any n_i after expansion since n_1 to n_{d-2} is a chain of newly added edges. This argument may be used for any vertex in $\mathcal{G}_{\text{int}} \setminus \{n\} \setminus \{l\}$. Since the existence of edge-disjoint paths is transitive, there are two edge-disjoint paths between any two vertices, and hence the graph remains two-edge-connected.

4.2 Implementation:

The implement the randomly directed exploration protocol on the same simulation framework as the previous protocol. Since the randomly directed exploration protocol relies on a local network topology, the random graph model cannot server for its simulations. Instead, the take the unit-disk graph as the sole network scenario. The choose a constant node degree and select as the priority range of the protocol. As a result, there are an average 2.5 neighbors in the priority zone of a node.

Module Description:

User: We design a dynamic peer to peer topology. Peers are interconnected and pass the data directly with each other peers by using maintain the connection details in dht. Peers have connection with other peers. Distributed hash table (dht) has peer details like peer name, ip address, port number and link specification. User enters to the network at the time all details of peer register into dht.

Link Calculation: Sender sends the data to receiver through intermediate peers. First sender peer collect the available peer details and calculate available paths between senders to receiver. Among that available paths sender calculate the shortest path for data transmission. And also determine node failure details.

Vertex Conjecture: Vertex Conjecture algorithm started to reconfiguration process when node failure make disjoint problem. Disjoint is nothing but one node split from network without its knowledge. That means one node suffers due to someone relive from network. At that time dht use this algorithm and calculate most in degree node in the network and connect this suffering node with most in degree node.

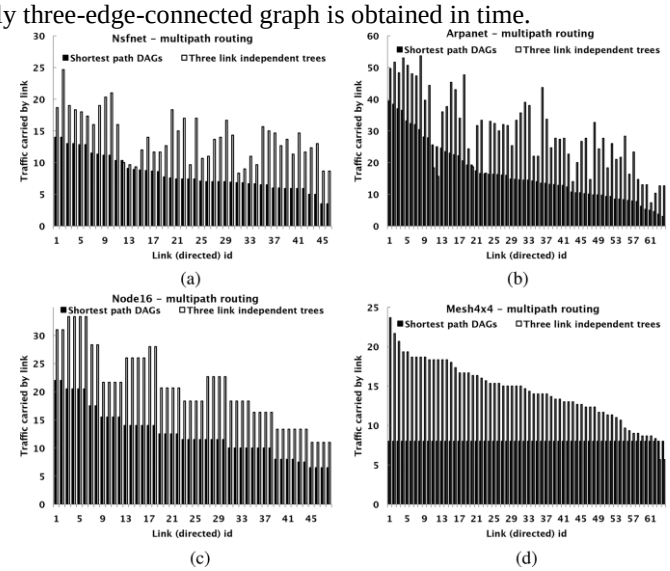
Data Transmission: All receiving packets are store in to the buffer and reorder the packets. Receiver receives the packets with secure manner. Data show using message receive box. Data are received by different path in different transaction of sender to receiver. Receiver also does not know about the route of the each data transaction. It makes more security for packets

7.2.1 Experiments:

A. Complexity Analysis:

Computing a minimally three-edge connected graph may be achieved in two steps. First, we compute a three-edge-connected *sparse* spanning subgraph of [33]. The number of edges in the sparse graph is guaranteed to be at most. Second, we reduce the sparse spanning subgraph to a minimally three-edge-connected graph. We consider one edge at a time and check if the edge may be removed without affecting the three-edge connectivity

of the spanning subgraph, which may be achieved in time for every edge. As the number of edges in the sparse subgraph is the minimally three-edge-connected graph is obtained in time.



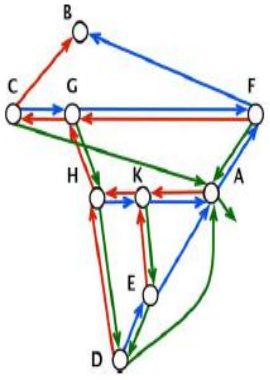
Traffic carried by each directed link in the networks (a) NSFNET, (b) ARPANET, (c) Node16, and (d) Mesh 4 4 under the two different multipath routing schemes.

The complexity of the cubic expansion procedure is . For every vertex, we compute the depth-first-search (DFS) numbering after removing the vertex to identify the two-edge-connected components. DFS numbering has a complexity , which is performed for every vertex. Note that the number of edges in a minimally three-edge-connected graph is linear in the number of vertices, hence the complexity is only o(v).

The complexity of constructing the paths in $\mathcal{Q}$ is $O(V^*E^*)$ [22]. The number of edges in $\mathcal{Q}$ is linear $(3/2)$ in the number of vertices, hence the complexity is $O(V^{*2})$. The number of vertices in the cubic graph V^* is bounded by $(V+(3V-6))$ by virtue of the cubic expansion procedure on the minimally three-edge-connected graph. Hence, the complexity of computing the paths in $\mathcal{Q}$ is $O(V^2)$.

Our procedure to segment the paths only consists of traversing each edge in the paths in $\mathcal{Q}$ one at a time. Hence, it takes only $O(E^*)$. Using the discussion above, this takes $O(V)$.

Therefore, the overall time complexity to compute three-edge-independent spanning trees is $O(V^2)$.



$S_0 : B - C - G - F - B$

$S_1 : G - H - K - A - F$

$S_2 : A - E - K$

$S_3 : E - D - H$

Conclusion and Feature Works:

We have provided an algorithm to construct three edge-independent spanning trees. This partially answers an outstanding problem in graph theory and also has several applications in networking and tomography. It remains to be seen if we can generate ideas that will work for arbitrary edge connectivity or will settle the implication conjecture on independent spanning trees. Based on our experience in developing the algorithm for this paper, it is not apparent if a general approach to compute edge-independent spanning trees can be derived just from the corresponding vertex-independent spanning trees, without the knowledge of how the vertex-independent spanning trees were constructed.

We show how the three edge-independent trees can be used for routing in an IP network. We develop a routing scheme that is capable of disjoint multipath routing using only the destination address in the packet header. We also develop three routing approaches using the trees such that the routing table entries are limited to at most four per node and very minimal packet overhead. All of the routing schemes developed are guaranteed to withstand any arbitrary two-link failures in the network. Through simulations, we show that the path lengths obtained by using the three trees is close to that obtained with the tunneling approach, even though the latter employs seven routing table entries per node and does not provide multipath routing capability.

Keeping backup configurations in the network makes protection of multicast traffic much easier. Protecting multicast traffic from node failures is a challenging task, since state information for a whole multicast

subtree is lost. By maintaining a separate multicast tree in each backup configuration, we believe that very fast recovery from both link and node failures can be achieved.

References:

1. Gopalan and S. Ramasubramanian, "Multipath routing and dual link failure recovery in IP networks using three link-independent trees," in Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst., Bengaluru, India, Dec. 2011, pp. 1–6.
2. J. Arkko, B. Briscoe, L. Eggert, A. Feldmann, and M. Handley, "Dagstuhl perspectives workshop on end-to-end protocols for the future internet," Comput. Commun. Rev., vol. 39, no. 2, pp. 42–47, Apr. 2009.
3. Cisco, "Cisco express forwarding," 2006 [Online]. Available: http://www.cisco.com/c/en/us/support /docs/routers/12000-series-routers/47321-ciscoef.html
4. G. Iannaccone, C. N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an IP backbone," in Proc. ACM Internet Meas. Workshop, 2002, pp. 237–242.
5. A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," in Proc. IEEE Symp. Found. Comput. Sci., 1984, pp. 137–147.
6. W. Zhang, G. Xue, J. Tang, and K. Thulasiraman, "Linear time construction of redundant trees for recovery schemes enhancing QoP and QoS," in Proc. IEEE Infocom, Miami, FL, USA, Mar. 2005, pp. 2702–2710.
7. S. Ramasubramanian, M. Harkara, and M. Krunz, "Linear time distributed construction of colored trees for disjoint multipath routing," Comput. Netw., vol. 51, no. 10, pp. 2854–2866, Jul. 2007.
8. P. Thulasiraman, S. Ramasubramanian, and M. Krunz, "Disjoint multipath routing in dual homing networks using colored trees," in Proc. IEEE Globecom Wireless Ad Hoc Sensor Netw. Symp., San Francisco, CA, USA, Nov.–Dec. 2006, pp. 1–5.
9. P. Thulasiraman, S. Ramasubramanian, and M. Krunz, "Disjoint multipath routing to two distinct drains in a multi-drain sensor network," in Proc. IEEE INFOCOM, Anchorage, AK, USA, May 2007, pp. 643–651.
10. A. Iselt, A. Kirstadter, A. Pardigon, and T. Schwabe, "Resilient routing using ECMP and MPLS," in Proc. HPSR, Phoenix, AZ, USA, Apr. 2004, pp. 345–349.
11. Reichert, Y. Glickmann, and T. Magedanz, "Two routing algorithms for failure protection in IP networks," in Proc. 10th IEEE ISCC, June 2005, pp. 97–102.
12. M. Shand and S. Bryant, "IP fast reroute framework," IETF Internet Draft draft-ietf-rtgwg-ipfrr-framework-08.txt, Feb. 2008.
13. Kvalbein, A. F. Hansen, T. Čičić, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," in Proc. IEEE Infocom, Apr. 2006, pp. 1–11.
14. F. Hansen, O. Lysne, T. Čičić, and S. Gjessing, "Fast proactive recovery from concurrent failures," in Proc. IEEE ICC, Jun. 2007, pp. 115–122.
15. S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "Proactive vs. reactive approaches to failure resilient routing," in Proc. IEEE INFOCOM, Mar. 2004, pp. 176–186.
16. J. Wang, Z. Zhong, and S. Nelakuditi, "Cam05-4: Handling multiple network failures through interface specific forwarding," in Proc. IEEE GLOBECOM, Nov. 2006, pp. 1–6.
17. S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using not-via addresses," Internet Draft draft-ietf-rtgwg-ipfrr-notvia-addresses-02.txt, Feb. 2008.
18. A. Li, P. Francois, and X. Yang, "On improving the efficiency and manageability of NotVia," in Proc. ACM CoNEXT, 2007, Art. no. 26.
19. S. Kini, S. Ramasubramanian, A. Kvalbein, and A. Hansen, "Fast recovery from dual link failures in IP networks," in Proc. IEEE INFOCOM, 2009, pp. 1368–1376.
20. K. Xi and J. Chao, "IP fast rerouting for single-link/node failure recovery," in Proc. Broadnets—Internet Technol. Symp., Sep. 2007, pp. 142–151.
21. A. Zehavi and A. Itai, "Three tree-paths," J. Graph Theory, vol. 13, no. 2, pp. 175–188, 1989.
22. J. Cheriyan and S. N. Maheshwari, "Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs," J. Algor., vol. 9, no. 4, pp. 507–537, 1988.